Profile and Trace Survey

Ian Clark

E-mail: ianclark@secondvalleysoftware.com

Last update before posting: 7^{th} September, 2008

This literature survey was started in 2002, updated in 2003 and then periodically afterwards for minor editing. The last update was in 2008. It was posted on 5^{th} March, 2011 when we started placing content on our website. It was pushed through *multicols* for two columns, and the *Excalibur* LaTeX spell checker. It is unlikely to be converted to Abobe tools or updated. The original survey was done while registered at Stellenbosch University and was unpublished. As it has no value if inaccessible, it has been placed in the public domain as far as our gathering of the relevant information for prior art. The works of individual authors might be covered by patents and copyright. References are provided. Enjoy...

Available at: www.secondvalleysoftware.com/research/pdfs/trace.pdf (684 kBytes)

Abstract

This profiling and tracing survey was part of my "background" or literature search to establish whether permanent hardware instrumentation was feasible, or if debugging via software instrumentation on a multi-core device could compensate for limited hardware visibility. The survey summarizes various tools without commenting on their suitability, and provides prior art references in the event of applying for any trace related patents, or defending any patent infringement suites. Hopefully, the survey is useful in similar efforts for performance related embedded work.

In embedded system development, at some stage even rudimentary measurements are required to ensure performance goals are met. The intention was to reuse available software trace infrastructure in the event of attempting to build a hardware based trace system, as the visualization and storage requirements are independent of the target. The instrumentation effort intended to understand complex systems by viewing trace data from various perspectives. The targets for monitoring are hard real-time distributed systems typically found in industrial automation.

Processor evaluation makes extensive use of profiling, simulation, tracing and hardware measurements. Profiling obtains average program performance for cache misses, clocks per instruction and other metrics. While simulation is flexible, it requires a detailed model and a higher skills level than taking actual hardware measurements. Tracing, or recording the program counter addresses as a program executes, allows the execution of a program path to be reconstructed.

Tracing need not be purely hardware– or software based, but generally a hybrid of both methods. Hardware measurements perturb a system the least, but are more difficult to gather for long traces. Besides being expensive and non-portable, hardware measurements are becoming increasingly difficult due to packaging, high speed requirements for acquisition, wide data– and address buses, plus limited storage in general purpose logic analyzers. Without filtering or compression, hardware tracing any 100 MHz bus of an embedded processor with a logic analyzer will generate almost a GByte [1] of data per second, causing the trace to be suspend while storing the captured data to secondary storage. Software tracing involves instrumenting either the original source code or the linked binary code. Trace compression or filtering is required for both hardware– and software generated traces for practical data handling.

Simulation and trace-driven simulation suffer from slowdowns of $10 \rightarrow 10\,000\times$ depending on detail, and are therefore not used for real-time performance evaluation. Multitasking and multiprocessor systems have additional tracing constraints as global time stamping is required to reconstruct the overall system's behavior.

[1] 32-bit address + 32-bit data + 8 control signals = 9 bytes per cycle, which is 900 MBytes per second for a 100 MHz bus. No time stamps are required for periodic sampling, however, for event triggering, time stamps are necessary which should be a 32-bit or longer to minimize counter overflows.

Contents

List of Figures

List of Tables

Chapter 1

Overview

This chapter describes the interest I have in the trace field and what motivated the desire for embedded instrumentation rather than "off-the-shelf" logic analyzers. There is always bias in any report, with this one being no different. The bias crept in over a period of several years while hoping to obtain funding or to eventually fund the development myself, and was also influenced by the systems I developed during the past thirty years. Most of these projects involved designing a SBC (Single Board Computer) from scratch or to produce something in very short deadlines, but without the authority to dictate what was to go into the design.

Before venturing into any built-in instrumentation, one needs to examine why it was so rare. My aim was to to compare hard real-time behavior to an original schedule by monitoring multiprocessor RISC (Reduced Instruction Set Computer) hardware using infrastructure from prior tracing research. A very important point of the survey is to highlight the prior art, as many trace patent claims ignored important (and widely cited) work. Anyone intending to build similar hardware would be well advised to cite prior work in case of lawsuits.

This survey examines several methods of profiling and tracing computer systems. Tracing is a means of instrumenting a system to study the execution path of a running program. From an instruction trace file, the original program path can be reconstructed for a computer with predictable memory (i.e. repeatable sequential code on a workstation that does not have access to external sensors via an analog-to-digital converter. For software instrumentation, the higher the level of detail, generally the greater the level of distortion — both in execution time slow down and code space dilation.

The first tracing and profiling articles were found on CiteSeer and Google, and later at the IEEE (Institute of Electrical and Electronic Engineers) Computer- and ACM (Association for Computing Machinery) digital libraries. From one article referencing another, the primary sources were found, however, during more recent searches on patent sites, there are so many that relate to tracing and profiling, that some of these are now included in the main chap-

ters, rather than only in the separate patents chapter. Patent search sites like *Patent Storm*, *Free Patents Online* and *Google Patents* have simplified patent searching, plus included cross references to "cited" as well as the patents that are cited by patents issued later. Semiconductor companies are also becoming more involved in large simulation efforts to verify their latest designs — IBM (International Business Machines), Sun Microsystems, Intel and AMD (Advanced Micro Devices) provide detailed papers, datasheets, and often software to simulate their processors. Their "in-house" journals reflect the well funded research required to develop new processors, which is not easily matched by small research groups.

The learning curves for any new processor were fairly steep, but the nature of a new board requires low-level debugging or "board bring-up". The other projects were "software only" on real-time kernels for a stable BSP (Board Support Package) but without any hardware assistance for debugging. The move from a predominantly hardware development environment to software prompted the search for a solution that could be used in more than one project, across architectures with minimal changes, and provide a "software logic analyzer". While this survey certainly took longer than anticipated, it is unlikely that any independent researcher would stumble across an elegant design without rigorous training. Patents are a source of solutions to numerous problems in the performance debugging field. Publications reflect years of work by groups of researchers who often provide source code for others to replicate their results or build upon their ideas. I have tried to group summaries from both the patents and publications, and make no claims to the completeness of the effort. Indeed, many a detour was taken down interesting paths, but hopefully the following pages are a useful starting point to your own efforts.

1.1 Motivation for Instrumentation

The original motivation to develop built-in instrumentation came from not being able to measure "head room" or sensitivity to additional tasks on existing systems. These systems were my own design in some cases, but rushed by market forces and a lack of a logic analyzer. They used a simple "tick based" kernel with tasks launched on prime number intervals to minimize the chances of simultaneous triggering. Other projects were longer term that had consumed buckets of money over many years and were "visited" as a contractor. It was impossible to instrument existing systems without funding, and the system packaging made no allowances for any test probes. Besides, the systems were fragile without "before" and "after" snapshots or regression tests to ensure errors were not introduced into any scheduled events.

As contract developers, we had little idea of idle time or transient behavior as critical tasks were executed within interrupt handlers. The frustration of not being able to measure these existing systems (even with a logic analyzer), while user-demands continually added more functionality onto an already brittle system prompted a search for solutions. On my own designs, the behavior was just as mysterious as I never owned a logic analyzer, but occasionally triggered an oscilloscope to measure repeated task execution times, and at least there was a RTOS (Real-Time Operating System) to make the system less brittle.

Porting a real-time kernel onto anything that does not support instrumentation makes the task unnecessarily difficult. Projects quickly get to a point where it is impossible to add anything without a scheduler or RTOS. Several years were spent porting BSPs and RTOS to embedded targets for customers. A few Linux ports also consumed many frustrating months and did not provide real-time in spite of kernel sizes of well over a megabyte.

An article Vestal (1994) presented a detailed

analysis of scheduler sensitivity, and how to modify one task while still maintaining a schedule guarantee. No measurements were made, but several hints were provided. The work was for hard real-time periodic tasks with a fixed-priority preemptive scheduler on a uniprocessor. This sparked my initial interest in this work; then how to measure or visualize the results convinced me of its importance.

I would like to use the trace data to verify a schedule, confirm claims for machine throughput (packaging machines), and debug transient failures. The level of trace detail is generally high — from branch instructions to procedure calls and exits. In commercial computer systems check pointing and replay are important for debugging, which is extremely important considering claims made that 40% of computer system failures result from software bugs costing the USA $59,5 billion annually Narayanasamy *et al.* (2006). Round figures like 40% and a rather exact amount of $59,9 billion rather than $60 billion are not very convincing, but the amounts are nevertheless very high. Complexity and the rush to deliver software are largely responsible for the huge losses. In industrial automation, the rush is no different, except if there are synchronization problems, then something bends or breaks and damaged product spills onto the floor. The skills to find these errors are not readily available on industrial equipment during initial design, commissioning, daily running or in the event of a fault. Simple tracing with time stamping can help improve matters. Depending on the depth of memory or the detail of a trace, this is essentially an embedded logic analyzer.

1.2 The Debugging Myth

Measuring the time spent on different tasks during development is difficult. Many companies ask developers to fill in weekly time sheets; I even tried to do that honestly for my own development projects, but eventually the allocation of time breaks down. The cycle of opening the editor, making a few simple changes or additions, saving, compiling and downloading to a target and then testing into the night are not easily attributed to any one category. The urge to track time disappears as the swamp fills up and you become surrounded by crocodiles, resulting in times that are not a true reflection of the testing share of a project. When papers present round figures, the results must be questioned, specially when they are 40%, 70% or give ranges like 30→70%. IBM researchers Hailpern and Santhanam (2002) estimated a range between 50→75%; in Nielsen *et al.* (2003) the range was 50→60% of the total development effort. Other "rules-of-thumb" from CASE (Computer Aided Software Engineering) textbooks are that coding takes one sixth of the time, however, most projects I contracted on never had one sixth dedicated to coding, as a six month project would imply one month of writing software and perhaps one month of testing — it was always six months of writing late into every night without formal testing, except at the end on payment milestones. When it was all over, we still had no idea of what the system was really doing — graphical charts, measurements, confirmation of a scheduler's timing constraints, head room, performance measurements — they were all missing. Many researchers have addressed these issues, but there is still no universal trace format or trace port, nor low-cost embedded instrumentation that can be "unplugged" for field deployment.

The unfortunate part of the search is that building hardware in a rapidly moving field requires a wide range of skills and expensive tools. Perhaps the answer is software, but in the absence of any RTOS publications amongst the trace research, limited hardware assistance might be necessary. Researchers use available hardware to complete publications in a reasonable time frame, unless the hardware is novel or for architectural exploration. While most people would acknowledge the importance of debugging, its attractiveness for publication is lacking after spending months of fault-finding hardware. In industry, the time to publish af-

ter completing a project is not there either, plus there is no motivation to publish without applying for a patent, as the competition is fierce.

The remaining bias in this survey reflects a move from industry to academia and back to industry. Processor manufacturers are slowly moving to make debugging less painful, but this is only to sell ever more complex devices; it is not in response to academics asking for more predictable devices.

1.3 Tracing Background

Software tracing or simulation may be sufficient for short sections of code; e.g. individual tasks, context switches, interrupt handlers and selected functions. Even for hardware tracing, when buffers are full and require saving to external secondary storage, tracing must be disrupted, or the traced target halted — which is impractical for hard real-time systems. Continuously tracing any high-speed processor to secondary storage will require filtering and data compression. Tracing systems require extensive software infrastructure. Features that were used for software tracing will certainly be useful in a hardware based system — typically which locations to trace, compression, filtering and visualization.

Simulation and Verification are certainly not small scale activities. In Lauterbach (1993), typical trace simulations were distributed across a network of fifty SPARC (Scalable Processor Architecture) ELC workstations. A few years later, Sun would use up to 1 000 SPARC processors to simulate and verify the 5,2 million transistor UltraSPARC-1 design (Yung, 1998, pg 78). The tape-out estimates for the UltraSPARC were out by eight months over a three year project (Yung, 1998, pg 28). There were over 150 engineers involved (Yung, 1998, pg 84) with a project cost of over $100 million (Yung, 1998, pg 46). The stakes are higher for processors which target games consoles. In 2004, IBM announced that Sony, Toshiba and IBM had invested $400 million in the CELL processor for the PlayStation 3 games console IBM Press Release (2004). According to Austin *et al.* (2002), the Intel Pentium 4 design team used a simulation pool that contained more than 1000 workstations.

For smaller design teams, any tracing assistance at the processor level is likely to improve delivery dates, as the elaborate infrastructure in simulation or instrumented testbeds available at the processor vendors' laboratories are unlikely to follow the device into the field.

1.4 Processors Considered

When the trace project was seriously considered (using an instrumented testbed around 2000), my embedded background was mainly MIPS with some PowerPC work. MIPS also had a fairly high academic profile with projects listed below:

MIPS2000 at Stanford and Hennesy startup,

DLX at Stanford Hennessy and Patterson (1990),

TORCH at Stanford,

SimOs at Stanford and described in §3.3.1,

DASH at Stanford,

Hydra at Stanford Hammond *et al.* (2000),

Pixie at MIPS,

DEC had several MIPS projects and ventured into a MIPS based workstation before developing their own RISC,

PRISC at Harvard by Razdan/ Smith,

FRISC at Rensselaer (Tien *et al.*, 1997, pg 240),

GAS at Illinois,

Tracing at Illinois Daigle *et al.* (1996), §6.8.1,

OneChip at Toronto Wittig (1995),

DLX tracing covered more in §6.8.6,

MIPS-like design at OpenCores,

Tracer ,

RAW at MIT Taylor *et al.* (2002); Waingold *et al.* (1997),

Vanilla Pekoe at MIT Heo (2000); Krashinsky (2001),

GARP at Berkeley Hauser (2000); Hauser and Wawrzynek (1997); Callahan *et al.* (2000),

SPIM at Wisconsin by Larus and covered in more detail in §3.3.3,

DLXview at Purdue,

MINT at Rochester Veenstra and Fowler (1994*a*),

R10000 Performance by Zagha *et al.* (1996),

WCET at SNU Korea.

PowerPC arrived after 1990 but took a while to become available to embedded designers. The sheer size of IBM, Freescale (formerly Motorola) and embedded systems market moving from 680x0 to PowerPC, resulted in increased publications for the PowerPC. ARM steadily captured the mass market by being designed into cell phones and later targeting 8-bit designs. MIPS faded from the desktop with the demise of SGI (Silicon Graphics Inc.), but they never addressed the low-end market with Flash-based μControllers or low cost, until MicroChip took out a license in 2007. Trace ports were promised by MIPS, but other than Toshiba and MicroChip μControllers, none have trace ports. As ARM devices disappeared into SoC (System-on-a-Chip) designs, debugging became a serious issue for software developers. The ARM ETM (Embedded Trace Macro) became standard for tracing, while IBM released trace ports in their PowerPC 4xx devices, a product line that was sold to AMCC (and repurchased later) who have expanded the 4xx family and retained the trace ports. It soon became apparent that a trace port and a trace port analyzer would need to be purchased. The AVR32 was introduced in February, 2006, with a trace port, as well as being able to trace into its own address space under user initiated commands Atmel Corp. (2006). The first AVR32 tests were based on an evaluation board purchased toward the end of 2006, and a 30-day trial license of the IAR tool chain. The cost of the IAR tool chain was far too high to consider any further work, however, Atmel did develop an *Eclipse* based environment with a GCC (GNU compiler collection) compiler tool chain that worked with their JTAG (Joint Test Action Group) debugger. This was used for a commercial project in 2008 and the good experience made the AVR32 a candidate for the debug target, however, harsh commercial reality shifted the debug emphasis to ARM.

1.5 Report Layout

This survey covers articles that were considered important by previous researchers and that would influence any hardware decisions for embedded instrumentation. Many sections were dedicated to software tracing as the data analysis is similar whether the collection was hardware– or software based. The survey is not limited to real-time systems as most of the articles appear to study architecture performance issues (cache), and it would have been a comparatively simple exercise to extend the studies to instrumenting a real-time kernel, however, that was not the original authors' intent. The layout is not in chronological order, but grouped by different types of tracing, and then where possible, by processor architectures.

The rest of this survey is laid out as follows; *Profiling* is covered in Chapter 2. The next four chapters follow the categories of tracing as defined by (Uhlig, 1995, pg 24), however, single stepping was omitted as it was not popular for tracing and is described it in more detail in Clark (2007).

- Chapter 3 covers *Processor Simulation*.
- Chapter 4 looks at *Instruction Modification*.
- Chapter 5 examines *Microcode Modification*.

- Chapter 6 describes *Hardware Monitors*, logic analyzers, and trace ports.
- Chapter 7 summarizes a few patents that were of interest to this work.
- Trace formats are listed in Chapter 8.

Many patents have been awarded for tracing (both hardware and software), so anyone wishing to enter the field commercially would need to be familiar with the areas covered, as they appear extremely broad, or ignore prior public knowledge. Graphical tools need to be adapted or developed to make the tremendous amounts of data useful for end-users. For the visualization side of this project, see Clark (2008).

Chapter 2

Profiling

2.1 Introduction to Profiling

Although profiling is used for performance evaluation, it is a periodic sampling of a program's execution and therefore obtains *average program performance* for cache misses, clocks per instruction and other metrics. Profiles may be time-based, event-based, use software instrumentation to profile a program as it executes, or hardware sampling of the PC (Program Counter) in a highly instrumented testbed — for example a soft-core in a FPGA (Field Programmable Gate Array).

Time-based profiling in embedded systems (and perhaps on the desktop) was to discover "hot spots" as future candidates for optimization. Profiling can be useful for code coverage in safety-critical systems to highlight areas which have not been tested or executed. To obtain performance figures from a time-based profile would require long running programs for a "statistically accurate" profile.

Event-based profiles rely on performance counters to generate interrupts on an overflow, and save internal counters for later analysis. Several modern microprocessors with built-in performance counters include: Atmel AVR32, Intel Pentium and Itanium, Cray vector processors, several IBM PowerPCs, DEC (Digital Equipment Corporation) Alpha, HP PA-8000, MIPS RM7000, MIPS R10000 and most recent MIPS cores. These counters were for profiling operating systems and provide information on cache misses, pipeline stalls, branch mispredictions, memory coherence operations and events that are invisible outside the processor, and therefore cannot be obtained by tracing.

Most of the profiling projects were for workstations and servers. Profiling of real-time systems are only covered in press releases and trivial tutorials presented by RTOS vendors, and are basically modifications of prior profiling work. A commercial package, CodeTEST, aimed at embedded systems is discussed in section 2.5.1. Another excellent reference on the advantages and disadvantages of profiling and optimization is Feigin (1999).

The layout starts with time-based profiles, next are event-based profiles, followed by software instrumentation for profiling, and lastly hardware instrumentation for sampling the PC. Within each group, general cases are discussed prior to specific tools, followed by processor related articles.

2.2 Time-Based Profiling

Time-based profiles rely on regular timer interrupts during a program's execution to sample the program counter. This is low overhead as the operating system has already stopped the process to maintain the system software clock Goldberg and Hennessy (1993). Afterward, a histogram of the functions provides insight where the program spent most of its time, and also which functions are candidates for optimization. The sampling is random in that the profiled program does not know when it will be interrupted, and if the sampling is taken over a sufficiently long period, then it should accurately reflect the program activity or identify "hot spots".

Time-based profiling systems take a snapshot of kernel structures to provide an average profile — usually taken over a fairly long run with low resolution clocks. See Danzig and Melvin (1990) why these profiles cannot be used for performance figures, even though some authors would give times to three decimal places when the UNIX clock interrupted at 60→100 Hz. If low resolution sampling at 100× a second is done on lots of small procedures, then many of them will not even appear in the profile if they execute for less than a second.

2.2.1 prof and gprof

PC sampling in UNIX `prof` and `gprof` periodically interrupted a program to increment a counter corresponding to the region containing the current PC.

`gprof` was described in Graham *et al.* (1982). The execution time monitoring consisted of three parts; the first allocated and initialized the runtime monitoring data structures before the program started. The second part was the monitoring routine called by the prologue of each profiled routine (prologue installed automatically by their compiler). The third part condensed the data structures and wrote them to disk when the program terminated. The monitoring routine took care of the various counters, and other than linking in the monitoring routines, the users only needed to ensure that augmented routine prologues were produced during compilation. `gprof` used call graphs to attribute the time spent by individual routines as well as those that were invoked by each routine. The execution overhead ranged between 5 $\rightarrow$ 30% for the program being profiled.

An advantage of linking in different monitoring routines was that the compiler did not require modification, and according to the authors, no recompilation of the program. That obviously depends on whether the precompiled object files were compiled with the augmented prologue. Another attraction of their method of handling counters inside the monitoring routines was that they were not scattered around the source code to be profiled.

For embedded targets running a small RTOS, the prologue might need to be manually placed using conditional compilation and run at the function call level. The counters were indexed in a hash table in the original article, however, for manual placement, the counter indices could be hard coded.

`gprof` was compared to `Mahler` in Wall (1989), as well as to their other late code modification tools that enabled their version of `gprof` to be more useful. The Titan experimental workstation had a user-visible clock that was used as an alternative to the timer interrupt in `gprof` for PC-sampling. The `Mahler` linker inserted code for noting the current time at key points, simplifying kernel-profiling.

The following criticism of `gprof` was taken from Keller and Urquhart (1994). There was no shared library support, thus requiring the program compilation with exclusively non-shared libraries. The system did not provide support for simultaneously profiling multiple processes, all processes which could be run had to be recompiled for routine-level profiling. The system was invasive (e.g. modified the executable code to be profiled), and required dedicating to profiling additional memory approximately half of the space of the program to be profiled. Moreover, in addition to the entire set of processes to be profiled having to be rebuilt in order to provide profiling, it was only capable of providing routine-level and no source statement or instruction level profiling, did not summarize all CPU usage but rather only that of one user program at a time, and further often required a substantial increase in user CPU time, sometimes approaching 300%, due to its invasiveness."

Particularly for embedded work, time-based profiles are not that useful as pointed out for general cases in John Jr. and Urquhart (1999), "...sampling once per interrupt gives a ragged view of the performance data. It is difficult to accurately understand what is happening in the system because the performance data is collected at random points. There is no data collected pertaining to a sequence of instructions running in the consecutive order in which they were intended to execute." Other than trying to identify "hot spots", profiling a RTOS is not as useful as a trace, however, as seen in Chapter 6, involves much more effort.

2.2.2 Profile from instrumented operating system timer

An IBM patent, Keller and Urquhart (1994) described a method of profiling programs that could account for operating system activity, shared (dynamically linked) libraries, and user code without recompiling any programs. The AIX (Advanced Interactive Executive) timer interrupt was modified to include a trace "hook" and tracing libraries that could collect data from the running processor, save the vir-

tual addresses of the program counter when interrupted, and post-process the data later to generate complete profiles. The patent title suggests that the profile was obtained from a "Non-invasive trace-driven system", however, the user did not need to recompile any programs, require special linking scripts, or otherwise be aware of any tracing activity; it was certainly not hardware based, and possibly incurred a slight increase in loading. The kernel interrupt is required for maintaining the system time, scheduling and other activities, and does not execute at a particularly high frequency (not given but unlikely under a millisecond). For high speed RISC processors running AIX, these would be capable of tens of millions of instructions per second when the patent was filed (1993), so interrupting at 100 Hz which was common for Unix systems at the time, the overhead would be light. Additionally, the profile was generated as a post-processing step afterward.

2.3 Event-Based Profiles

Instead of using a timer to generate a periodic interrupt, hardware event counters can generate interrupts on overflows or preset event masks. Event-based profiles rely on performance counter overflows (if they increment), or counters reaching zero for user loaded down-counters. Cache and performance studies that relied on internal counters were generally run on workstation class machines. Early embedded processors did not have performance counters; many did not have cache or TLB (Translation Lookaside Buffer) hardware, which explains the bias in this section for high-end devices. The groupings are the DEC Alpha, MIPS, PowerPC and Intel's Pentium and Itanium.

2.3.1 Alpha Event-Based Profiles

DCPI

See Anderson *et al.* (1997) for a detailed description of DCPI (Digital's Continuous Profiling Infrastructure), which claimed a 1→3% slow down for most workloads. The internal performance counters of the DEC Alpha generated periodic interrupts. Other tools for Alpha mentioned in the same article were `iprobe`, `dcpiprof`, `dcpicalc`, `dcpistats` and `Spike/OM`. Using their tools to identify a performance problem, they reducing a SQL (Structured Query Language) query from 180 to 14 hours

In Lindenmaier *et al.* (2000), DCPI was used to profile several SPECfp95 benchmarks, one SPECint95 benchmark and the Livermore loops to provide input for scheduling instructions. There were two programs that reported a 10% improvement, most were improved by 1% and two programs (`compress95` on a 21064 and `turb3d` on a 21164) had significant degradation due to flaws in the Balanced scheduler. The small improvements and amount of work show diminishing returns when trying to tweak code on modern processors. The machines were idle, so the effect of operating system interrupts would be interesting, particularly for real-time systems.

ProfileMe

`ProfileMe` was described in two papers from DEC/ Compaq Anderson *et al.* (1998); Dean *et al.* (1997). From the paper titles the profiling was by hardware support for out-of-order execution processors, but the hardware does not appear to have been built. (It would have to be internal to the CPU (Central Processing Unit) to gain access to the pipeline and branch prediction units). The Alpha 21064 and 21164 were already available. They were described as in-order processors, whereas the 21264 was described as a 'concrete' example of an out-of-order processor. (I am not sure whether the `ProfileMe` hardware made it into the 21364 or

follow-on processors. Compaq essentially canceled future Alpha development in June 2001).

To measure `ProfileMe` statistics, a cycle-accurate simulator of the Alpha 21264 processor was extended to gather statistics for `compress`, `gcc`, `go`, `ijpeg`, `LI`, `perl`, `povray` and `vortex`. The authors also traced each of the programs in the SPECint95 benchmarks.

The hardware described was rather complex and must be duplicated to handle instruction pairs that the authors were interested in. The profiling was not for PC sampling or event counting, but instruction sampling. A complete record of interesting events (such as cache misses and branch mispredictions), was directly associated with each profiled instruction. A wealth of additional information was collected, including pipeline stage latencies, branch history data, and effective addresses for memory operations. By additionally allowing a pair of in-flight instructions to be simultaneously profiled, other information was collected — useful concurrency levels and pipeline stage utilization.

The instruction to profile was selected by writing a random number into an incrementing counter that counted fetched instructions, and when it overflowed, the processor was interrupted. To reduce interrupt overhead, there were several copies of profiled registers stored into FIFO (First-in-first-out)s. By random sampling, a statistically meaningful estimate of program behavior was gathered.

With the DCPI profiles (as well as the `ProfileMe` hardware), information could be gathered over the entire system — executable programs, shared libraries, device-drivers plus the operating system kernel.

Although the above two papers describe transparent hardware support for instruction profiling on out-of-order processors, the hardware was internal and did not provide continuous traces.

Morph

`Morph` was a framework for collecting profiles and optimizing programs Zhang *et al.* (1997). It included *Morph Monitor*, a kernel component that collected continuous, low overhead profiling and program monitoring, and *Morph Editor* which re-optimized the transformed compiler intermediate form into an executable. *Morph Manager* managed profile information and the automatic invocation of re-optimization. Other components were *Morph Back-end* and *Post-Morph*. The back-end produced executable programs and shared libraries that included annotations required by `Morph` to support efficient late optimization. *PostMorph* was a means of extending legacy executables by inferring `Morph` annotations through static and dynamic analysis so that they could be retargeted by the *Morph Editor*. The system was written for Digital Unix on the Alpha to meet the following requirements:

- Optimization should happen on the machine where the software is used,
- Optimization should not require source code,
- Optimization should be transparent to the user.

Collecting profiles was similar to DCPI; statistical sampling of all system activity with low overhead. An important point was the assumption that each user had "their own" workstation, and that they had a private copy of the program to profile — after all, it was going to be changed by the profile of that user's interaction. The compiler optimizations were implemented as `SUIF` (Stanford University Intermediate Format) passes on a compact representation of the compiler intermediate form, as `Morph` did not require source code.

The monitor was implemented as a pseudo-device and the following kernel functions were modified; *hardclock()*, *exec()*, *mmap()*, and the *exit()* calls. The *exec* and *exit* calls allowed the kernel to save information about which application was running (and its address space),

plus when it was terminated. Eight bytes per sample at a clock interrupt rate of 1024 Hz allowed for 30 seconds of profile data with a 256 KB kernel sample buffer. The buffer was organized as a ring buffer with sampling continuing during the dump to disk. (I assume the dump activity was not recorded).

The profile samples were processed off-line during off-peak times at a rate of 60 MB per minute. This also implies that any optimizations will only be noticed (if at all), the next day. Long-term profile storage was another problem with sizes ranging from 29 $\rightarrow$ 84% for the programs in (Zhang *et al.*, 1997, Table 6). Recall that these would be per user.

Spike

Spike, Flower *et al.* (2001), was an executable optimizer that used profile information to place application code for improved fetch efficiency and reduced cache footprint. Initially developed for NT, it was later modified for Tru64 Unix on 21264 Alpha processors. Applications that were optimized included the Unix kernel, TPC-C benchmarks (30% improvements initially), Oracle and web applications. The TPC-C Oracle code spent roughly 30% of its time in the Unix kernel, while the SPECweb program spent 85% of its time in the kernel.

The profile information was taken from exact counts of basic blocks produced by simulation, or estimated counts produced by the DCPI statistical profiler. For large programs like commercial databases or the Unix kernel, there were no significant performance differences when using the much faster (to gather) DCPI data.

Spike changed the code layout to improve cache performance by reducing the cache footprint, arranging basic blocks of a procedure so that the frequently executed paths were in a straight-line, and placing frequently called procedures close to the calling procedure to minimize cache conflicts. Procedures in the kernel are accessed relative to the *global pointer* register, and branches on the Alpha have a 21-bit offset displacement limit.

Some performance figures for Icache improvements were 50% for the SPECweb96 benchmark. Cache miss data was collected using DCPI and ProfileMe. Interesting values for cache misses on the Alpha 21264A were approximately 100+ cycles for L2 misses to memory, 50+ cycles for ITB misses and 20 cycles for an Icache miss that hits in the L2 cache. These figures show how bad the high clock frequency Alpha's memory mismatch was, and also how important the cache design was to prevent the processor waiting most of the time for memory accesses. Other enhancements were placement of pages for cache improvements, prefetching data by inserting prefetch based on profile-guided stride information. On Oracle TPC-C there were approximately 70 prefetches inserted into the image which improved the performance by roughly 10% (in addition to the 31% from other Spike improvements without prefetching to give a 39% performance improvement). Stride profiling was especially effective for *equake* and *applu* with improvements of 56% and 22% respectively.

In comparing related work, the authors commented that the significantly lower profiling overhead from DCPI compared to cache simulation was necessary in building a practical product. They also claimed that Spike enables customer optimization of the Unix kernel specifically for their workloads. These would certainly need to be sophisticated customers to justify the additional expense of kernel source code or to instrument at this level. Spike inserted prefetches into the binary directly and therefore did not require any source code — particularly attractive when either the source code is unavailable or re-compilation is impractical. I am not sure if the authors included not requiring "kernel" source code.

Related to embedded work, programmers could place frequently called procedures close together in the same source file or link order. There are no sophisticated profilers for embedded targets that frequently rely on GNU (GNU is Not Unix) tools or a generic Linux kernel.

Often, the only documentation is the source code, so programmers might place functions in alphabetical order to find them easier (for human readers). Perhaps procedure level profiling might show up the hot spots where jumps are fairly far apart or frequent cache conflicts reduce performance.

Ephemeral Profiling

Ephemeral profiling was a term given to describe the instrumentation which will "come and go" as the program executes Traub *et al.* (2000). Stubs were inserting at conditional branches that unconditionally branched to instrumentation code that then determined if the branch was taken or not. After some threshold, the stub was removed. This is much like inserting software breakpoints. The work was for the 21164 Alpha with the periodic epoch set at 10 ms. The instrumentation aimed at less than 5% overhead to guide code optimization.

The comparisons were against complete profiles using `HALT` (§4.3).

Programmable Co-processor for Profiling

The co-processor was described in Zilles and Sohi (2001). Fairly similar to the `ProfileMe` work, but able to profile multiple in-flight instructions simultaneously. To evaluate their design, a cycle-accurate simulator was built of the co-processor and the algorithms implemented in the simulated co-processor's microcode. The co-processor model was included in the timing simulator derived from the Alpha version of `SimpleScalar` which simulated the main processor, a 4-way superscalar, dynamically scheduled processor, roughly modeled after the Alpha 21264. Although the experiments were performed on a 64-bit architecture, the PC information in the profiling slots was 32-bit (as the upper half was found to have very little content).

The estimates for the baseline design were half a million transistors, with 300 000 of these for CAM (Content Addressable Memory) memory arrays to provide inexpensive hash table-like functionality for lookups and matching.

The benchmarks tested were the SPEC2000 integer suite, that ran between 9– and 44 billion instructions. The simulated inputs were a 100 million instruction region to reduce the execution duration. The results were mostly within 5% of estimates and overhead below 2%.

2.3.2 MIPS Event-Based Profiles

MIPS was an early RISC which was widely used in academic research and commercially in SGI workstations. Performance counters were included in the 64-bit processors, for example, the R10000 and later devices for workstations, followed by the RM7000 and later devices used in embedded systems.

Performance Analysis using R10000 counters

The R10000 counters were described by SGI staff in Zagha *et al.* (1996). They described the motivation and placement of the counters, as well as the design effort — under 300 lines of RTL, fewer than 5,000 transistors, and two 32-bit counters which could be configured to monitor any one of 16 distinct types. There were 30 different event types that could be monitored, as two event types ("Cycles" and "Graduated instructions") may be monitored by either of the two counters.

The counters were designed to be non-intrusive, as monitoring was done in hardware instead of software. SGI provided tools to configure and monitor the counters. The kernel maintained up to 32 different 64-bit virtual counters which were available through a programming interface. Software tools described in Zagha *et al.* (1996) were `perfex`, `Speedshop`, and `Performance Co-Pilot`.

Examples of problems that were identified with `Speedshop` were: cache problems for solving three-dimensional partial differential equations on a 100 x 100 x 100 grid — speedup

was 1.3×; one line in a 45,000 line program accounted for 40% of the cache misses. By adding a shadow array, the total secondary cache miss for the entire application was reduced from 30→12%. Several other examples were given relating to multiprocessor shared memory, weather prediction and branch prediction.

2.3.3 Intel Pentium-4 and Itanium Event-Based Profiles

Intel introduced SMT (Simultaneous Multi-Threading) in the Pentium-4 or Xeon processors in the 2002 Q1 issue of the *Intel Technology Journal*. As hyperthreading would make one processor appear as two cores to the operating system, vendors needed measurement methods to tune their operating systems and applications. Although prior Intel processors provided performance counters, I have not done a detailed search as the x86 architecture was aimed at the desktop rather than embedded space. This is likely to change with multi-core and low-cost hardware so more recent devices were selected.

Intel Pentium-4 Performance-Monitoring Features

The section header is taken from Sprunt (2002*b*). The Pentium 4 was the first x86 processor to support SMT, and the performance monitoring counters were used to compare the l1_miss benchmark for single thread mode and dual thread mode. The dual threaded mode's throughput was roughly half that obtained running the same benchmark sequentially in single threaded mode. The author also commented that the Pentium 4 Xeon's SMT features can be both an advantage and a detriment to overall performance, and that to take full advantage of SMT, the operating system will have to carefully select which tasks can concurrently share a SMT processor.

2.3.4 Intel Multi-core Profiling

In Alexandrov *et al.* (2007), profiling is presented for multi-core and multi-threaded processors using the Intel Performance Tuning Utility package. The Intel PTU was based on prior work with the Intel VTune Performance Analyzer. The user-friendly graphical interface uses Eclipse for both Windows- and Linux platforms. The IA-32 dual-core, 64-bit Xeon and Itanium were supported. The paper was written for testing on an eight-core Xeon platform running Redhat Linux.

Profiling was aimed at finding bottlenecks in applications that could be executed in parallel. The differences in the internal performance counters was masked by the tool, which is available for downloading.

The interface was well thought out. Collapsible spreadsheets displayed source code with the corresponding cycle counts.

2.3.5 PowerPC Event-based Profiles

Groups within IBM have invested heavily in the profiling infrastructure of later models of the PowerPC family. The PowerPC studies have been grouped, although they were from widely different sources and different devices. Most of the IBM profiling work appears to be in patents. These were examined after searching the IEEE and ACM digital libraries, and unfortunately have not been added into this document yet.

PowerPC 604e Performance Monitoring

The 604e had several hardware counters that could measure up to 111 unique events. The PM (Performance Monitor) counters in the 604e PowerPC were more powerful than those previously described for the Alpha, R10000 and Pentium. The work was described in Levine and Roth (1997).

In 1991, early versions of the POWER processors had no monitoring support in the proces-

sor. Cards were created to attach and monitor processor bus activity. In 1992, the POWER2 processor had 22 counters integrated into four units, which were implemented as separate chips. A bit (PMM) in the MSR (Machine Status Register) controlled counting. In 1993, revision 2.0 of the 604 had on-chip monitoring with additional ability to support or prohibit counting in user– or supervisor mode. In 1995, the 604e was released with additional counters and support. In 1997 (year the paper was written), the POWER2SC shipped while the PowerPC620 was still under development.

An early application of the PM counters was to count all unaligned data accesses for legacy code, which determined that the performance could be improved by enhancing the CPU to handle an unaligned access. The 604e provided additional support to study a program's memory access patterns and interaction with a system's memory hierarchy, including SMP (Symmetric Multi-Processing) environments with events relating to the MESI (Modified Exclusive Shared Invalid) cache coherency protocol.

In the PowerPC 604e, there were four hardware counters to measure up to 111 unique events that cannot be monitored externally. Support was added to count the number of occurrences of certain classes of instructions. By identifying the amount of time spent executing a specified instruction, the number of occurrences of the instruction and the addresses of the occurrences, one could determine the expected improvement in performance for proposed changes regarding handling the instruction in either software or hardware. The ability to count matches at a specific instruction address allowed counting to start after a specific instruction had executed a specific number of times. The 604e also had the ability of counting the number of cycles spent while interrupts were inhibited or pending. There was support to determine the effectiveness of the branch-prediction logic.

The AIX interface to the performance counters maintained 64-bit counts. User accessible features included enabling/ disabling the counters, storing samples to sections of memory, setting the periodic sampling rates, synchronizing multiple counters across multiple processors, selecting the timebase register bits to trigger on, and downloading initialization code to setup the counters before enabling them.

The counter API (Application Program Interface) and paper Levine and Roth (1997), provide additional insights into processor performance evaluation. Even with hardware tracing, or built-in trace, many of the features measured with the 604e counters are unavailable on other architectures.

Xilinx FPGA Embedded PowerPC 405 Profiling

A profile was generated from trace data and described in a Xilinx application note Njoroge (2004). Several problems with the tools setup included:

- The Xilinx supplied `gprof` had problems reading PowerPC binaries.
- RISCWatch could not profile Linux due to TLB accesses not handled and corrupting the trace.
- Problems with the Agilent Trace Port Analyzer due to different version numbers not recognized by the analyzer.

An application note is not meant to describe problems but to show how to use silicon that is marketed as functional, so when Xilinx evaluation boards end up outside of the USA or Europe, designed by a different company, then you are not going to get much response from any of your incompatible tool vendors. Can you imagine being able to contact Agilent or IBM and complain about some version numbers? The Xilinx PowerPC based boards (which did not work as envisaged from marketing material), and board support packages were never updated by the vendors as Xilinx races on to the next generation of software and devices (essentially abandoning the

previous devices). The article did not mention even more fundamental problems I experienced — not being able to source level debug over the Xilinx Parallel-IV cable as promised. The tools kept crashing and were unable to load up external memory. The on-board BlockRAM was too small to put in anything meaningful — other JTAG pods were required which the marketing department forgot to mention. The software upgrade from one release of the EDK (Embedded Development Kit) to the next broke code that was previously tested. The new tools were meant to read in the old design and then convert it to the new release.

FPGAs are great for prototyping but the FPGA industry is indeed a fast moving one that does not favor backwards compatibility, and although the application note dealt with tracing the embedded 405 core in Xilinx FPGAs, the actual FPGA was not used for embedded instrumentation.

LeProf

The `leprof` profiling tool was part of the IBM MET (Microarchitecture Exploration Toolset) package, and described in Moudgill (1998). In comparison to `prof`, `gprof` and `tprof`, which slow down a program by $10 \rightarrow 20\%$, `leprof` slows down by a factor of $20 \rightarrow 100\times$. However, `leprof` provided exact branch and cache behavior information by generating a trace of a program and extracting the following profile statistics:

- for each instruction, how often it was executed.
- for each branch instruction, whether it was taken or a fall-through branch.
- for each load or store instruction, whether it would hit or miss in the data cache.

The instruction-level information for a function was determined by obtaining the start and end of a function from an executable, and then adding up the frequency of data miss statistics for all instructions in that range. The kernel could not be traced, so it was impossible to obtain kernel activity information on behalf of the program being profiled. `leprof` kept track of the number of times a function was called and the position of the caller, which allowed the programmer to see the dynamic function call graph as well as the frequency of invocations.

Moudgill (1998) included a manual section, examples of statistics and some interesting case studies. In the *Future Work* section, tighter integration with `Aria` was suggested. Both `Aria` and `Turendot` are covered in § 3.5.1 on page 35.

AS400 Profiling

The AS400 changed from a CISC (Complex Instruction Set Computer) processor to a PowerPC RISC processor in the mid 1990s. In Schmidt *et al.* (1998), the authors describe their profiling tool to aid in restructuring the operating system code to improve performance. Three types of profilers were described; sampling, trace-based and instrumenting. The authors chose the instrumenting profiler as they perceived difficulties with using a trace-based profiler. Their tools would be supplied to IBM customers, and there was no practical method of making trace tools available to them, as a real-time trace reduction method would be needed to make the trace a manageable size.

The instrumentation "hooks" incremented counters whenever a branch decision or procedure call occurred. The data was used to reorder basic blocks once the call profile was available. To speed up the enabling and disabling of instrumentation, a single bit in the condition code register was tested. The effectiveness of the work was shown in a series of benchmarks, where the average improvement was 20%.

In the *Concluding Remarks* section, another team used the profiling data to construct a code coverage tool. Although the tool could not determine whether all possible paths were exercised, it was able to indicate those procedures and basic blocks which were not tested at all.

Blue Gene Profiling

In November 2004, the IBM Blue Gene (BG/L) became the world's fastest supercomputer. The tuning and profiling was described in Martorell *et al.* (2005). The IBM Blue Gene was developed in partnership with Lawrence Livemore National Laboratory as a collection of up to 65 536 dual processor SoC devices. Each SoC contained two PPC 440 cores which each had a dual custom FPU (Floating-Point Unit), 32K instruction and data L1 cache, a small (2-KB) L2 cache and prefetch buffer, shared L3 cache, shared 4 MB embedded DRAM (Dynamic RAM), trace ports, shared high speed links for a torus interconnect (6 in, 6 out at 1,4 Gb/s each), and several counters for performance measurements.

The performance counter unit had 48 32-bit counters to monitor up to 311 events. The normal PPC 64-bit timestamp register was also available. Several performance counter related packages were ported to BG/L — HPM, a version of `perfctr` called BGLperfctr, PAPI, Paraver, and the MPItrace package. A large portion of the performance tuning was done on a simulator, BLGsim, for details of instruction stalls that could not be determined from hardware monitoring or tracing. The simulator is briefly mentioned in section 3.5.4.

There was a section dedicated to *Performance counter limitations* which mainly mentioned the floating-point monitoring limitations. The supercomputer was designed for "grand challenges" with floating-point intensive problems, so perhaps more floating-point monitoring would have been appropriate. The 405 PowerPC that was a predecessor to the 440 did not have a FPU as the 405/ 440 devices were initially aimed at embedded work (which is not generally floating-point intensive). The 405 and 440 have trace ports, however, it was impractical to trace several thousand processors, particularly when densely packaged in racks of 1024 compute nodes of 2048 processors (and air cooled).

2.3.6 Selective Profiling using breakpoints

A patent, Heisch (1998) used breakpoints to start and stop profiling within a program. The abstract follows: "A microprocessor performance monitor and instruction address break point facility are interconnected to provide finer granularity and performance monitoring. The microprocessor is initialized to collect processor statistics preselected prior to performance monitoring. Application start and stop instruction breakpoint addresses are preselected from a software program bounding instructions for which such statistics are desired. An exception handler is installed for instruction address breakpoints (IAB), enabling and disabling the performance monitor and stop addresses, respectively. The IAB register is then initialized to the start address, and the statistics counters are cleared. Upon starting the application, when the application start address instruction is executed, the breakpoint handler obtains control and enables the performance monitor counters, which count the desired statistics after returning from the breakpoint handler. Before returning, the handler sets the IAB register to the stop address. When the application stop address is encountered, the breakpoint handler disables the performance monitor counters, and rearms the start address in the IAB register. The performance monitor counters are then read to determine the desired statistics for the specific sequence of code within the boundaries of the start and stop addresses in the application."

Xbox 360 Profiling

The triple 64-bit PowerPCs in the game console have extensive debug features. See Brown (2005) for some of the profile and trace features. For profiling, the Xbox 360 CPU provided 16 32-bit counters which could monitor hundreds of events across all the functional units in the processor chip. There were programmable start/stop and synchronization conditions.

2.4 Performance Counter Tool Chains

There are several tool chains that allow reading performance counters, however, in an effort to make the interface "universal" there are several layers. Some cycle times are given in Table 2.2. As an example of reading the 64-bit cycle counter on a Pentium, the following assembler routine takes very little time:

```
rdtsc
mov hi,edx
mov lo,eax
```

Reading the 64-bit cycle timer on a PowerPC is not much different (Motorola Inc., 1997, pg 2-16):

```
loop:
  mftbu  rx    # Load Time base upper
  mftb   ry    # Load Time base lower
  mftbu  rz    # Load upper again
  cmpw   rz,rx # See if 'old = new'
  bne    loop  # Loop if carry occurred
```

The Pentium version should also check if the lower 32-bits overflowed into the upper 32-bit value of the 64-bit timer register, but it is much less than almost a 1000 cycles when called from user programs and being thread-safe or whatever else is performed in the portable event counter libraries. For real-time work, my preferences would be to use the minimum number of cycles to reduce the "probe effect".

The number of timers and possible events to monitor are given in Table 2.1.

Table 2.1: Performance Counters and Events Monitored. Data was taken from Sprunt (2002*a*).

Processor	Counters	Events
P5 (Pentium & MMX)	2	$>$ 200 on Pentium III
AMD Athlon	4	$\approx$ 25
IBM 750	4	$>$ 40
Motorola 7450	6	$>$ 200
Itanium	4	$>$ 90
UltraSparc I & II	2	$>$ 20
Alpha 21264	2	Not given, see §2.3.1 & §2.3.1

2.4.1 PAPI

The PAPI (Performance Application Programming Interface) project from the University of Tennessee provided a library of calls that read hardware performance counters. The figures given for PAPI 3 are listed in Table 2.2.

The PAPI API had the following goals:

- To provide a solid foundation for cross platform performance tools,
- To present a set of standard definitions for performance metrics on all platforms,
- To provide a standard API among users, vendors and academics,
- To be easy to use, well documented, and freely available.

PAPI was written in `C`, but could also be called from Fortran programs. The PAPI library named approximately 100 preset events which were defined in *papiStdEventDefs.h*. Obviously not all targets supported all events, but where possible, a software interface was provided. A high-level API provided eight functions, while the low-level API had approximately fifty functions. See University of Tennessee (2004) for thread and MPI (Message

Table 2.2: PAPI Performance figures taken from (University of Tennessee, 2004, pg 6)

Platform	PAPI_read()
Itanium 2 – Madison	1357 Cycles/Call
IBM Power4	4034 Cycles/Call
Itanium 2 (libpfm 2.0)	1606 Cycles/Call
Pentium 3 (perfctr 2.4.5)	324 Cycles/Call
Pentium 4 (perfctr 2.4.5)	401 Cycles/Call
SGI R12k	3681 Cycles/Call
UltraSparc II	2150 Cycles/Call

Passing Interface) use.

Statistical profiling relies on a periodic timer interrupt. The PAPI library aimed to generalize this functionality so that a histogram could be generated using any countable hardware event as the basis for the interrupt signal.

2.4.2 PCL

The PCL (Performance Counter Library) project, Berrendorf and Mohr (2003), allows the hardware performance counters to be read for the following processors; Intel Pentiums, AMD Athlon/Duron, PowerPC (604, 604e, Power3, Power3-II), Alpha (21164, 21264), MIPS R10k and R12k, and UltraSPARC (I/II/III). The library can be called from `C`, `C++`, Fortran and Java.

2.4.3 perfmon2

`perfmon2`, from HP Labs provided an interface "to access the performance counters present in all modern processors". The aim was to provide a standard for the Linux kernel, focusing on the Intel Itanium. The prior tools referenced were PAPI and PCL (briefly covered in the above two subsections), and Intel's VTune, OProfile, perfctr, pin and several others.

The Itanium had about 200 measurable events, while the Itanium 2 had almost 500 events that the PMU (Performance Monitoring Unit) could monitor. The width of the counters also moved from 32 $\rightarrow$ 47 bits between the two families. The documentation on the PMUs was normally kept secret (i.e. not well documented for user access) (Eranian, 2005, pg 9).

2.5 Instrumenting for Profiling

Code instrumentation is generally for tracing, however, in Ball and Larus (1992), the edges of the CFG (control-flow graph) were instrumented to obtain an exact count of basic blocks. By instrumenting for counters rather than a sequential trace of "witness markers", counts of locations of interest were generated. Ball and Larus also optimized the location of counters and reduced the number of counters required compared to other programs like `pixie` or CodeTEST.

The CFG for the program must be generated for the work in Ball and Larus (1992). By careful placement of counters and solving for *vertex frequency* or *edge frequency*, Ball and Larus claim that in many cases the placement is optimal. To reduce the costs of profiling (i.e. the number of counters), these counters were placed in areas of low execution frequency. The tracing portion of this work is covered in §4.10.2.

2.5.1 CodeTEST

CodeTEST was developed by Applied Microsystems Corporation, and described in WindRiver Systems (1999) as part of bundled options for WindRiver Systems' Tornado development framework.

The CodeTEST components included:

Instrumenter – a set of software utilities that prepared the target for testing.

Host Application – tools that ran on a workstation to measure code coverage and memory allocation.

Target Utilities Library – a set of routines that were linked with the target software to transmit *tags* to the host and respond to requests from the target server.

The *instrumenter* inserted test point instructions, or *tags* into the source code that was instrumented. The unique *tags* allowed the host to identify and track various program activities. The *instrumenter* also maintained the instrumentation database for program symbols, as CodeTEST did not use the symbols generated by the normal compiler and linker. *Amctag* (part of the *instrumenter*), inserted tags into the source code, created and maintained the instrumentation database, and optionally performed `C` or `C++` preprocessing. The remaining tools in the *instrumenter* acted as compiler drivers for the target (much like *gcc* is the driver for the GNU `C` compiler).

On the host, two tools — *Memory* and *Coverage*, could operate separately or simultaneously to generate several types of measurements for a single target test run.

Although CodeTEST was bundled with WindRiver Systems' real-time framework, it will obviously impact the target execution time and real-time behavior. The instrumentation was not intended to be left in the deployed target, but addressed three phases of software development — *unit* testing, *integration* testing, and *system* testing. The most likely tests would be *unit* testing, but the manual recommends using the tools at every stage of development and testing.

The front-end for the instrumentation consisted of various dialogs on the Windows host (for update times, functions to instrument, the level etc.). Levels of instrumentation were procedure entry and exit, or for all branch tests (including entry and exit).

The collected data from the target *tags* was used by the host tools for code coverage, profiling, high-lighting code that was not executed (in a window dialog box at the source code level), or exporting the data in formats compatible with spread-sheet type tools for users to generate their own reports.

The memory allocation functions were based on CodeTEST "wrapper functions". *Malloc*, *free* and others were substituted by *amc_malloc* etc.

Although CodeTEST could be classified as a profiler, in (WindRiver Systems, 1999, pg 7-6), the tags only recorded whether certain blocks were executed for code coverage, not their sequence or the number of times they were executed.

2.6 Performance Counter Limitations

There are limitations to performance counters, particularly as processors become superscalar, out-of-order, SMT or multicore. Some limitations from Sprunt (2002*a*) are;

Too few counters Only a few counters can run concurrently, so several runs of a program with different events monitored are required to obtain performance data.

Speculative counts If event counters include instructions that did not complete, the problem of speculative versus non-speculative count arises.

Sampling delay The pipeline contains several instructions and the spread of instructions identified as the event can span as many as 25 instructions.

Lack of data-address profiling Cache and TLB misses for data are difficult to identify.

In Dean *et al.* (1997), the problem of identifying the correct instruction was also mentioned;

> event counters do not accurately attribute events to instructions: the instruction that caused the event resulting in an event-counter overflow is usually earlier, by an unpredictable amount, than the instruction whose PC is delivered to the interrupt handler. Out-of-order and speculative execution amplify this problem, but it is present even on in-order machines.

In the same article, an example of the Pentium Pro's hardware events being smeared over 25 instruction after an event make it almost impossible to attribute the event to the instruction that caused it. The authors also noted that similar behavior was observed on the R10000 hardware event counters.

2.7 Hardware-Based Profiles

2.7.1 Profiling Soft-cores

The RTNI (Real Time Non-Intrusive) instrumentation for the Xilinx MicroBlaze™ soft-core was described in Fryer (2005). On-chip counters provided methods of totaling statistics for measurements. An instrumenting compiler made the counter and flag logical assignments. Besides profile data or events accumulated in counters, a limited trace mode was mentioned with FIFO and SERDES (Serializer/Deserializer) resources to send the data to an external host. See sections 6.5 and 6.5.1 for a description of the trace options and the instrumentation memory.

The `SnoopP` (*Snoop*ing Profiler) was described in Shannon and Chow (2004). There were several segments which had a lower– and upper address comparator attached to a counter. The counters incremented clock pulses rather than instructions and were enabled when the program counter was between the address comparator ranges. The Xilinx MicroBlaze™ soft-core in a Virtex-II 2000 FPGA was tested with 64-bit counters. The counters consumed more resources than the MicroBlaze™; 16 of the 64-bit counters took up 1129 flip-flops and 1719 LUT (LookUp Table)s. To implement 32 32-bit comparators used 1024 LUTs. The authors pointed out that a function called outside the profile comparators was excluded from the count, and that users would need to add address ranges around the called functions to obtain accurate accounting. To change the address ranges, the design had to be resynthesized.

As FPGAs improve in speed at lower prices, more soft cores and testing of existing cores in FPGAs will become common. Several cores are available for free, and have also been implemented in semi-custom chips (structured ASIC (Application Specific Integrated Circuit)). As an example of structured ASICs available from eASIC, visit www.easic.com — ARM, OpenCores' OpenRISC 1200, Tensilica and LEON3 as of April, 2008. Instrumenting these cores will be fairly easy if you have access to VHDL (VHSIC High-level Definition Language) or Verilog source. Besides eASIC's ez-IP program, there is also ReadyIP from Synplicity — ARM, CAST, Gaisler Research and Tensilica were part of the program by April, 2008.

2.7.2 VAX Histogram Hardware Monitor

The VAX 8700 had a special hardware monitor that was able to keep a count of each microinstruction executed, and was used in Bhandarkar and Clark (1991) where the authors compared a VAX 8700 to a MIPS M/2000 system from a CISC/ RISC architectural viewpoint. They profiled the SPEC (Standard Performance Evaluation Corporation) benchmarks to examine CPI (Cycles Per Instruction) metrics. The MIPS machine used `pixie` and `pixstats`.

Hardware profiles on the VAX are a special case which will be interesting for profiling soft-

cores in a FPGA. Profiling soft-cores can be performed fairly easily without changing the application source code, however, expect to add in additional VHDL or Verilog code. Several methods of hardware based profiling are given in the *Tools* section of some other research, (which we will post later) as well as in several patents.

2.8 Summary

Several profiling systems were examined. Many of the cache– and branch prediction metrics would not have been available from trace data — even a hardware trace, as there is no visibility into performance counters from the chip pins. `gprof` and `prof` statistics could be produced fairly easily from a trace file, however, gathering the trace is a difficult problem.

Many examples of profiling in this chapter relied on simulation, however, the authors did not say how they handled the problem of jumping to debug code and polluting the cache in the simulated processor, or whether the host took care of the debug or profiling code. For a hardware based system, jumping to debug code when hitting a breakpoint in cached mode will modify the cache. See a patent for a cache safe debugger on user settable breakpoints Maemura (1995).

Chapter 3

Tracing via Simulation

> Execution-driven simulation is the preeminent method computer architects use to evaluate trade-offs in future computer system designs. Mauer *et al.* (2002)

3.1 Introduction to Tracing via Simulation

Probably the most popular method (Uhlig, 1995, pg 5) of examining cache performance is *trace driven memory simulation*, which is an important tool for computer systems performance analysis and prediction. Tracing has been used to study disk scheduling Jin *et al.* (2001), cache behavior, parallel processing, database tuning Anderson *et al.* (1997), and operating systems Flanagan *et al.* (1992); Nagle *et al.* (1992). Many of the references at the end of this survey used trace driven simulation to study cache behavior and gather CPI metrics[1].

Software tracing has a long history in time-sharing mainframe centers, with one of the earlier papers describing the data collection facility for an IBM System/360 in Pinkerton (1969). Another early study described a CDC (Control Data Corporation) 6600 multiprocessor trace system Sherman *et al.* (1972). This survey examines tracing literature after the mid 1980s when RISC systems became available outside academia and research projects. For overviews of tracing, the reader is referred to Mazières and Smith (1994); Pierce *et al.* (1995). Over 50 trace driven simulation tools were examined in Uhlig (1995). For limitations of trace driven simulation, and trace capturing, the

[1] It should be noted that it is not possible to use hardware to trace cache unless there is a physical connection between the CPU core and external cache memory, and hence researchers who are not part of a semiconductor laboratory with instrumented testbeds have to resort to software models of cache, and simulation.

ATUM (Address Tracing Using Microcode) work provides many useful hints for anyone considering building any trace capture hardware Agarwal *et al.* (1986).

There are times when one cannot simply insert breakpoints or significantly slow down a running system Stewart and Gentleman (1997); where safety issues are involved (e.g., it is unsafe to breakpoint the control program of a welding robot operating a welding torch); when performing *in situ* stress measurements on a program; or debugging in the field where the program must continue to provide client services (e.g., telephone switching software). Even a simple application to scan several multiplexed 7-segment displays is useless for single-stepping or using breakpoints. An ARM brochure highlighted their debugger for embedded hard drive controllers, which would not be possible under any simulation environment (or single stepping/ breakpoints).

Gathering traces with minimal perturbation is difficult and usually requires hardware assistance. Due to the difficulty in collecting hardware traces, other methods need to be examined and possibly modified for an embedded target. Hardware tracing was important enough to establish a Trace Distribution Center at BYU (Brigham Young University), tracing at Michigan University, Harvard, and later at NMSU (New Mexico State University).

The problem of collecting trace data has not gone away and was aptly put in Larus (1993).

> In the past, collecting detailed program traces was extremely costly, and the few existing large trace files were carefully preserved and passed like ancient manuscripts among researchers.

The tracing chapters were divided up as per (Uhlig, 1995, pg 24). Unlike profiling, tracing obtains a complete listing of instruction– or data-referenced addresses. Examining hard real-time schedules requires a history of the program execution path together with time stamping of several events. Most studies used simulation to generate traces, and hybrid systems to emulate a target architecture. Simulation is particularly important for cache studies where code dilation severely impacts cache behavior. When traces are generated on basic blocks, the intermediate instructions might have to be emulated to regenerate the program execution for tracing studies.

Another possible area where simulation would help is in the estimation of WCET (Worst Case Execution Time) for small sections of code. For hard real-time systems the cache will likely be disabled, which simplifies the model. However, the pipeline stages need to be modeled for timing information. A simulator is useful during early stages of the project, particularly to generate traces for the host software infrastructure. This will be easier to change than modifying VHDL and PCB (printed circuit board) layouts, and once a specification can be drawn up, the actual hardware trace unit can be designed. The trace format will not change much at this stage.

There are several simulators; those listed here are mainly for the Alpha–, MIPS–, PowerPC–, x86– or SPARC devices. The simulation sections have been divided up by processor architectures. Earlier simulation papers favored large systems, cache- and performance studies. Deeply embedded devices did not attract academic interest until more recently (for power-aware scheduling, single chip multiprocessors, etc.). Perhaps large RISC systems lost ground to the financial resources of the x86 suppliers while embedded devices have improved in speed.

A note on simulation was taken from Hunt and Sawada (1999)

> In a nutshell, a single mathematical proof can provide the confidence provided by an exhaustive set of functional tests, often at a cost far lower than exhaustive simulation. For something as large as a microprocessor, complete functional testing is not an option because of the number of different states is so

large that no full testing regimen can be completed.

Several interesting architectures were simulated in academia before producing physical hardware. Many of these ideas found their way into mainstream chips. The architectures of these academic projects are more likely to be available than commercial simulators from leading shipping devices, as the sheer cost of developing a new chip does not warrant giving away large infrastructure intellectual property to potential competitors.

The following classification of simulators was taken from Mauer *et al.* (2002):

Full-system simulators have high functional fidelity and can simulate OS code because they precisely model devices (e.g., Ethernet, disks).

Static full-system simulators playback a recorded trace of system operation, and are functional-first simulators that use traces that include OS code and device events.

Dynamic full-system simulators are execution-driven and allow system behavior to be affected by timing (e.g., thread interleaving).

Functional-first simulators use a functional component to produce a (logical) stream of committed instructions that are fed to a timing component.

Trace-driven simulators are well-known examples of function-first simulation.

Timing-first simulation is a decoupled organization in which a timing simulator runs ahead of a functional simulator. The timing simulator executes instructions using the mostly correct functional implementation of the instruction set. This execution is compared to, and corrected by the functional simulator.

The layout is not intentionally in date order, however, MIPS did have an early start together with the Berkeley RISC research. DEC published many technical reports — VAX to MIPS to Alpha. SPARC became popular with Sun Microsystem's widely successful workstations. PowerPC papers are more popular at the high-end, particularly as IBM started to focus on the Power architecture for both mainframes and embedded markets (games consoles, communications). ARM is the biggest installed base of 32-bit RISC devices having taken the largest cellphone– and low power SoC market share. (Note that NEC and Hitachi might be the largest μController vendors, but we assume all the ARM vendors have a larger total). ARM simulation is mainly directed at hardware modeling for chip fabrication. Section 3.10 on *iWatcher* describes hardware, however, the original paper simulated hardware which is why is has been added to this chapter.

A large simulation effort that has not been covered was the IBM System z9, as the i390 architecture is not readily available to researchers and certainly not to embedded developers. See Theurich *et al.* (2007) where simulation was used to verify firmware. The IBM *Journal of Research and Development* dedicated a double issue to the z9 (Vol 51, N° 1/2, 2007). Embedded developers have not used Intel Itanium processors for high volume products, however, as processors become more complex, the simulation efforts for Inferno would be useful as a reference as well as partitioning any simulator framework — this was briefly covered in § 3.9.2.

3.2 Patents Affecting Simulation

There are several patents that could affect commercialization of a full system cycle-accurate simulator. There are many more on maintaining machine state for breakpointing, cache flushing during debug and mixed simulation where a host drives pins on a target testbed. I decided to address patents where they affect the simulation framework I am developing, as the field is too large to cover in a short survey.

However, some are far reaching and have been included where appropriate. One such patent is US Patent N° 7 149 676 B2, awarded 12^{th} December, 2006, to Krishnan of Renesas Krishnan (2006).

Descriptions of moving from a functional model to being able to checkpoint the CPU state and execute RTL (Register Transfer Logic) prior to the above patent was Carver *et al.* (1999). Whole machine simulation with variable levels of detail was done by more than one research group, but `SimOS` (§3.3.1), was one of the better known efforts. Other work at Stanford did simulate at the RTL level, but I am unsure of the performance (cycle accurate) and functional simulation being combined in any of the academic projects. `SimOS` was able to "warm the cache" and then zoom into areas of interest by changing the simulation level, but the change from machine emulation using binary translation to switch to simulation was not at the RTL level. Brochures from the major EDA (Electronic Design Automation) vendors certainly supported hardware/ software co-simulation with models for memory and being able to model sections of code on a functional simulator and then zoom in on a bus model or cycle accurate model. A combination of hardware trace capture and machine simulation was described in Sandon *et al.* (1997) and briefly covered in § 6.8.4. Other IBM simulation efforts are able to zoom into areas of interest and any big semiconductor vendor can presumably zoom into a section of a chip to run cycle accurate simulation.

3.3 MIPS Architecture Simulation

Much of the pioneering MIPS work was done at Stanford in the early 1980s. The architecture was targeted at the minicomputer and workstation vendors with rather high prices for a supposedly simpler and easier to manufacture device. Performance was better than the aging VAX from DEC, which was also high priced. The early MIPS devices were manufactured by Toshiba and others for customers like SGI, Nixdorf, Olivetti, and even DEC for a while before the Alpha shipped. The architecture was new resulting in many research projects. On the embedded front, IDT (Integrated Device Technology, Inc.) made an effort to provide evaluation boards, a tool chain and low-cost devices. Some developers left MIPS and formed QED (Quantum Effect Devices), who designed some chips for IDT before launching their own highly successful brand of 64-bit devices. Due to the small startups with no spare hands, the first articles appeared from academia, who were also more inclined to share ideas and had to publish.

OpenCores.org have several 32-bit cores and some MIPS-like work, however, the memory of Lexra is still there. Altium's TSK3000 soft-core looks very much like a MIPS core (from the instructions and registers), but there is no mention of the MIPS heritage anywhere in the documentation. Anyone seriously considering a MIPS compatible core should read Ristelhueber (2001), as the legal implications could sink a "start-up".

3.3.1 SimOS Full System Simulation

`SimOS` was developed as part of the Stanford `FLASH` project which started in 1992 Rosenblum *et al.* (1997). `SimOS` was an environment for studying the hardware and software of computer systems. Complete machine simulation was possible for several MIPS and SGI systems (including multiprocessors). The CPU models were:- R4000 (MIPS ISA (Instruction Set Architecture) IV) and a R10000 superscalar "out-of-order execution" MIPS CPU. There were two modes — fast which was basically binary-to-binary translation and producing a slow down of less than 10×, — and a more detailed mode yielding a 1000× slow down which simulated the CPU, cache, hard drive, Ethernet and memory.

The fast (emulation) mode was used to boot an operating system, warm the file cache and position the workload for detailed study. `SimOS`

could switch dynamically between emulation and simulation modes. The high-speed emulation included `Embra`, which used dynamic binary translation pioneered in `Shade` (§ 3.4.2). `Embra` extended the techniques of `Shade` to support complete machine simulation, including modeling the MMU (Memory Management Unit), privileged instructions and the trap architecture of the machine. `Embra` also extended `Shade` techniques to handle multiprocessors. There were two processor pipeline models in `SimOS`, the first called `Mipsy`, was a simple pipeline with blocking caches as used in the MIPS R4000. The second, called `MXS`, was a superscalar, dynamically scheduled pipeline with non-blocking caches as used in the MIPS R10000. `MXS` was an order of magnitude slower than `Mipsy` due to the significantly more complex R10000 pipeline.

To address the challenges of selecting the recording detail level and characterizing events, a `Tcl` (Tool command language) scripting language was embedded into `SimOS`. A scripting language allowed users to customize the data collection without having to modify the simulator. `Tcl` script annotations were used to allow mapping low-level events to higher-level concepts, which ran whenever an event occurred that had an annotation attached to it. Annotations had access to the entire machine, including registers, TLB, devices, caches and memories. Examples of simulated hardware events on which annotations could be set include:

- Execution reaching a particular program counter address,
- Referencing a particular data address,
- Occurrence of an exception or interrupt,
- Execution of a particular opcode (eg. *rfe* or *eret* to determine whether the application is in kernel or user mode),
- Reaching a particular cycle count.

Context switches could also trigger annotations, which was of interest when studying an application, as the user could determine when the process was active without having to directly instrument the operating system.

Workloads studied were: booting the IRIX operating system, database workloads, SPEC95fp benchmarks and `SUIF` parallel compiled programs. "Future work" mentioned better visualization in Rosenblum *et al.* (1997).

`SimOS` was also described in Rosenblum *et al.* (1995), where a bus monitor validated the results on the Stanford DASH multiprocessor (§ 6.8.11 for a brief description of the hardware).

The Alpha support was added by Redstone *et al.* Redstone *et al.* (2000) in their work at Washington University for SMT performance measurements. The `SimOS` software is available from Rosenblum *et al.*. Validating the performance of `Embra` for uniprocessor and multiprocessors was described in Witchel and Rosenblum (1996).

3.3.2 Stanford FLASH Multiprocessor

The Stanford FLASH Multiprocessor was a large MIPS R10000 based system designed to scale up to 4096 nodes and support both shared memory and message passing applications. The link to each node was a `MAGIC` protocol chip which was based on the MIPS ISA with some modifications for bit operations. The `MAGIC` chip was a simple two-way issue 64-bit RISC processor with no hardware interlocks, no floating-point capability, no TLB or virtual memory and no interrupts or restartable instructions.

To develop the protocol processor, extensive simulation was done at the gate–, RTL–, HLL behavioral– and complete machine level. The simulation environment was described in Heinrich *et al.* (1997). As the design progressed, more detailed simulation was performed, however, without initial design specifications that could be simulated, much of the evaluation was done with pencil and paper. The `MAGIC` chip was 250,000 gates with 220,000 bits of memory.

The initial simulation (after the ISA was fixed) was on a simple simulator that assigned

fixed latencies to all instructions. Next, a pipeline simulator added in low-level details of the microarchitecture. Later a complete machine simulation was done with an execution-driven simulation that coupled the cache and pipeline model. The simulator for the entire FLASH machine was called `FlashLite`. To verify the hardware, a simulation hookup was built that allowed the pins of the `MAGIC` chip to be controlled by a high-level simulator. Switch-level simulation was done with `IRSim` and transistor-level simulation with `Spice`. These simulations were done on very small parts of the design, as they were 100 and 10 000× slower than the RTL gate-level simulation respectively. Some of the speeds at the different levels of hierarchy were (in cycles per second): `FlashLite` C handlers — 90 000, `FlashLite PPsim` — 80 000, HLL Behavioral RTL — 13, HHL Behavioral RTL plus coverage analysis — 6, and HHL Gate-level — 3.

The development took three years when the paper was published representing a significant simulation effort. The authors recommended concurrent development of the hardware/ software, then testing small sections at a time, rather than the traditional serialized design approach.

In a later article Martonosi *et al.* (1996*b*), `FlashPoint` was used to gather performance statistics from the `MAGIC` protocol processor. When monitoring, `FlashPoint` was intrusive, as it augmented the default cache protocol with performance monitoring information.

Anyone wishing to build a simulator should read Gibson *et al.* (2000) beforehand. Some excerpts are: Simulation requires a significant effort to build the tools — the FLASH design effort produced a number of architectural simulators over a period of six years. There were still significant errors in these simulators, even though they were used to develop hardware. Mispredicted hardware performance was out by 61% (under-predicted) for the 300 MHz Solo-Mipsy. On a 16-processor simulation, the Radix-Sort was out by 31%. The authors warn that using simulators to predict performance speedups can be off by 30% — a simulator error that is often larger than the performance gains from architectural enhancements reported in the research literature. In the *Conclusion* section, the authors offer the following sage advice:

> This conclusion demonstrates the importance of building real hardware and should cause some concern not only for computer architects, but the broader research community for whom simulation is the only method of performance evaluation.

3.3.3 SPIM

`SPIM` (MIPS backwards) was a MIPS R2000/R3000 simulator that accepted assembly language programs providing a simple debugger. It did not simulate cache or memory latency, nor did it provide accurate timing for floating-point operations or multiply/ divide delays. `SPIM` ran the same endianess as the underlying machine — on a `x86` host `SPIM` was little-endian, whereas on SPARC or PowerPC based Apple Mac, `SPIM` ran big-endian.

The software ran on UNIX, Linux and Windows and may be used for non-commercial purposes. See Larus (1991) for a 25 page PostScript manual on `SPIM`. The execution slow down was given as 25×, however, the decoding was not given (assembly when loaded).

3.3.4 MIPSI - MIPS Simulator

`MIPSI` Sirer (1997?) was a senior undergraduate thesis. Although HTML advertised download areas, the links were not functional. `MIPSI` was an instruction-level simulator for a MIPS CPU. It ran both big– and little-endian code, with a slow down of 65× for most SPEC benchmarks. The original goal for writing `MIPSI` was to examine fine-grain instruction level parallelism in `C` and `ML` programs. `MIPSI` could handle 3010 FPU instructions, emulate a 3000 cache, had hooks for TLB measurement, could

generate instruction traces, allowed fast symbolic debugging and handled many system calls and signals.

`MIPSI` was influenced (and originally based on) `SPIM`, but with added functionality. `MIPSI` was able to simulate parallel machines, VLIW (Very Long Instruction Word)/ superscalar and superpipelined processors. The original reference was Sirer (1993). There was a fair amount of activity based on `MIPSI` — Tullsen *et al.* of Washington University modified `MIPSI` to simulate the Alpha processor for their work on multi-threaded processor architectures.

3.3.5 TORCH Simulator

There were three simulators for TORCH. One was `Tsim`, a very fast instruction level simulator based on `Mable`, which was developed by Peter Davies at MIPS. `Ttrace` was a `pixie`-based simulator that used basic-block counts to estimate total execution time on a simulated machine. The other simulators for TORCH were `xsim`, a complete Verilog model of TORCH, and several `C` routines used by the Verilog model PLI (Programming Language Interface).

TORCH was an experimental superscalar processor that had 40-bit instructions which were a superset of the MIPS R2000 instructions. (Unmodified R2000 programs could run in compatibility mode). TORCH performed instruction scheduling in the compiler, and hardware in the TORCH machine allowed the compiler to schedule any instruction before preceding branches, a term the TORCHers called *boosting*. In the TORCH Architectural Specification, a detailed description was given of how the 40-bit instructions were grouped into instruction *packets*. The packet grouped the extra byte of each modified 32-bit instruction in the lower order address of the packet (eight bytes), followed by eight 32-bit portions of the instructions. When the 40-bit instructions were read into the data cache, a small amount of extra hardware shifted each of the first eight bytes onto the following 32-bits of the instruction. This did have implications for addressing, as explained in the specification The TORCHers (1994). The home page for the TORCH simulators was at Stanford University, but although the HTML had download links, they were broken. The site was www-flash.stanford.edu/torch/simulators/ in case it is available later.

3.3.6 Tango Lite and PROTEUS

Tango Lite work was done at Stanford and PROTEUS at MIT (Massachusetts Institute of Technology). The two systems were described in Cmelik and Keppel (1993). Both these systems were multiprocessor simulators designed to run programs on nearly identical hosts and targets. The greater the host/ target discrepancy, the larger the "coloring" of the simulation. The tracing levels could be changed, with Tango Lite slow downs ranging from 10→750, and PROTEUS slow downs from 2→500.

A more detailed description from one of the two PROTEUS developers is Dellarocas (1991). The system was designed to simulate multiprocessors on a MIPS R3000 based DECstation with X-Windows graphics. One of the screen snapshots (Dellarocas, 1991, pg 45) might help in prior art for multiprocessor visualization — see IBM's patent Advani *et al.* (2000).

A 21-page manual for Tango-Lite was available from www-flash.stanford.edu/~herrod/docs/tango-lite.ps. Tango-Lite was a multiprocessor simulator designed to run on MIPS R3000-based uniprocessors. Instrumentation was provided by a tool called *Aug*, which augments one of more assembler files with simulation code for clock management, context switches, plus other events of interest. Tango-Lite was designed to give users a "clean slate" when designing memory simulators. These ranged in complexity from a simple system up to a complete simulation of the DASH multiprocessor.

3.3.7 MINT — MIPS Interpreter

MINT was a software package designed to ease the process of constructing event driven memory hierarchy simulators for multiprocessors. It provided a set of simulated processors that ran standard Unix executable files compiled for a MIPS R3000 based multiprocessor. The package was described in a tutorial Veenstra and Fowler (1994*b*) and a paper Veenstra and Fowler (1994*a*). The MINT software, which was available from Rochester University ran on SGI, DEC (MIPS based) and SPARC (simulating MIPS code).

MINT was a fast program-driven simulator for multiprocessor systems, designed to ease the process of constructing event-driven memory hierarchy simulators for MIPS R3000 based multiprocessors. The multiple streams of memory reference events drive a user-provided memory system simulator. MINT used a novel hybrid technique that exploited the best aspects of native execution and software interpretation to minimize the overhead of processor simulation. Combined with related techniques to improve performance, this approach made simulation on uniprocessor hosts extremely efficient.

3.3.8 MINT+

MINT+ was described in a paper on the Sandcraft Montage MIPS μP Choquette *et al.* (1999). It had a Tk/X11 graphic front-end for cycle-by-cycle information of instructions. One of the authors, Veenstra was also involved in MINT. Sandcraft developed their system modeling tools to share with customers to tune their applications for the Montage architecture or to jointly define a processor specification. The tools focused on three areas; simulation, system modeling and visualization. For the simulation, they developed an interpreter-based, execution driven simulator for the MIPS IV ISA. The ISA simulation modeled the user-visible registers, while the second part of the simulator modeled the microarchitecture in more detail. The ISA functions of the simulator were collected in a library called MINT+; there were two interfaces, a simple fast interface which also modeled the cache in enough detail so that workloads could be positioned for the second, more detailed interface which simulated the out-of-order speculative execution as well as the pipeline stages. The memory interface was modeled accurately for SDRAM (Synchronous Dynamic RAM) using a split-transaction bus, as well as catering for DMA (Direct Memory Access) transfers. The visualization tools provided an interface to the simulation output, ranging from simple histograms to complex "waterfall" diagrams of each instruction's progress in the pipeline. Views of the two integer ALU (Arithmetic Logic Unit)s interleaved with floating-point calculations could show the bus utilization at different ratios of processor pipeline- to bus frequencies.

As the target market was embedded and graphics applications, the simulator was able to run sections of a benchmark without having to load all the linked libraries or boot a full operating system.

3.3.9 Functional Verification at SGI

Commercial processor manufacturers make extensive use of simulation for design exploration, however, verification is becoming the bottleneck as complexity increases in shorter market windows. In Hosseini *et al.* (1996), simulation of a functional model is run in parallel to a RTL model to compare traces for errors in the RTL. Several tools were described to generate code for diagnostics — hand written directed diagnostics, pseudo-random generators, tools to generate instruction sequences which stress the processor, and finally "real world" programs. Single and multiprocessor configurations were tested.

The *SBVer* tool verified external interfaces. These become complex with multiple outstanding loads and stores, maintaining multi-level caches, and performing cache coherency in multiprocessor configurations. Knowledge

of the primary and secondary caches, the memory layout, TLB and memory spaces was built into the tool. *SBVer* was used to find flaws in four generations of processors. Branch prediction is critical in modern processors. *BRVer* was a branch verifier that could test branch prediction logic in a systematic way. A note about many pseudo-random code generators was that they avoid complex branching sequences, especially backwards jumps to prevent infinite loops. (Remember that the MIPS processor has a branch delay slot as well). Multiprocessor verification was provided by a tool called *MPVer*. Besides *MPVer*, users could test parallel applications by writing `C` programs, which would first be tested on a workstation or a multiprocessor system, then compiled into code for the functional simulator and RTL model. Several other tools were written to examine the diagnostics outputs, store the results into a database, query the database, profiler, methods of traversing the trace files in both the forward and backwards (in time) directions, and a code generator called *Theo*.

3.4 SPARC Simulators

The SPARC architecture was extremely popular as it was used in graphics workstations, servers and in telecommunications. The chip design had its origins in the Berkeley RISC work in the early 1980s, about the same time as the MIPS work at Stanford. There were several SPARC licensees but the processor targeted the high-end rather than deeply embedded work. In the embedded arena, the sliding register window scheme was not particularly attractive as there were so many registers to save on a context switch, however, the public domain appearance of the LEON core from the European Space Agency and the lack of any law suites made experimentation attractive. In 2005, Sun Microsystems offered the T1 core design and architecture under an open source license to anyone wishing to produce SPARC compatible devices. See Boulton (2005) for the initial announcement and the www.opensparc.net website. One such design was the S1 Core from Simply RISC, who announced their 64-bit Wishbone-compliant processor based on the OpenSPARC T1 microprocessor Simply RISC (2006). Large FPGAs have made soft cores viable for small design teams who can invest resources into an architecture without future law suites. This will influence choice in the future. This may be attractive to new Indian and Chinese developers, who would not be able to establish a new design against the current offerings without obvious significant benefits.

3.4.1 SpixTools

The historical perspective of `SpixTools` were described in Cmelik and Keppel (1993). `SpixTools` appeared circa summer 1988 for instruction level profiling for SPARC and inspired by similar tools from MIPS. `spix` (like `pixie` from MIPS in 1986) created an instrumented version of a program. The instrumented program, running an average 1.6× slower than the original, counted the execution of basic blocks as the program ran, and wrote out these counts when the program terminated. For these counts, `spixstats` (like `pixstats` from MIPS) was used to print dynamic profiling information.

Unlike `pixie`, `spix` could not generate address traces. `Span` was developed in the fall of 1988 to fill this need, but was abandoned in favor of Peter Hsu's `Shadow` which appeared in summer of 1989. `Shadow` used simple dynamic compilation, and applications without analysis ran on average 64× slower than native. `Shade` was the improved version of `Shadow`.

3.4.2 Shade

`Shade` Cmelik and Keppel (1993, 1994), was an instruction-set simulator and custom trace generator that used dynamic compilation and caching techniques to build fast cross-architecture simulators. Tracing was flexible, and could be modified while the program ran. It was possible to trace over a range of ad-

dresses (certain functions or libraries), and to change the trace selection criteria (address, data, branches, cache etc.). When the execution of one translation was followed dynamically after another, the two translations (predecessor and successor) were directly connected or *chained* to save a pass through the main simulation loop.

It mainly supported V8 SPARC architecture, but was also ported to support 32-bit big-endian MIPS and later to 64-bit V9 SPARC. On average, Shade-MIPS.V8 executed 10 SPARC instructions per MIPS instruction. The trace analyzer was user supplied. `Shade` was used to trace SPEC benchmarks. When run on a SPARC and simulating a SPARC, it ran 2.3× slower for floating-point programs and 6.2× slower for integer programs. `Shade` did not trace operating system code or multiprocessor systems. `Shade` simulated many machine details including dynamic linking, asynchronous signals and synchronous exceptions. Due to its speed, traces could be regenerated on demand instead of storing large files. `Shade` worked on binaries, so users were freed from preprocessing steps, requiring source code, or complicated build procedures. The authors were from the University of Washington and Sun Microsystems, Inc.

For readers interested in writing platform independent simulators that use translation, read (§4.1) of Cmelik and Keppel (1994). The SPARC was big-endian only, whereas MIPS could run either little– or big-endian. The authors chose to only run big-endian code to simplify access on the SPARC host. Most workstations these days are `x86` based, which are little-endian. Immediate fields for MIPS were 16-bits, while SPARC used 13-bits, and therefore the immediate MIPS fields could not fit into similar SPARC instructions. In (§4.2) of the same paper, the Shade-V9.V8 simulated a 64-bit SPARC on a 32-bit host, which obviously has wider registers and addressing. The authors chose not to simulate a 64-bit address space. For maximum use in embedded systems, the host would need to be portable (laptop). There are 64-bit laptops based on `x86` compatible devices (AMD-64 processors). The target and hosts differ in the number of registers for a CISC `x86` host and RISC target. The SPARC and MIPS differ mainly in the sliding register windows.

3.4.3 LEON

The European Space Agency initially developed the LEON soft core, which ran the SPARC V8 instruction set. Since the initial launch, the main author Jiri Gaisler, left to commercialize the technology. The new company is Gaisler Research, based in Sweden.

The LEON core was ported to newer FPGAs, continually improving the clock frequency. In 2008, the LEON3 core was ported to an ASIC from eASIC Business Wire (2008*a*), running at double the fastest prior clock frequency in a FPGA.

3.4.4 Thunder SPARC

Metaflow Technologies described their selection of tools for simulation and verification of a SPARC V8 compatible core, which took nine months from Verilog model to tapeout, followed by eight months to post-silicon validation Popescu and McNamara (1996). The design flow and innovative "verification backplane" certainly allowed a more orderly verification compared to some of tales that were read in the DEC and Intel processor efforts. At six million transistors and 80 MHz, the processor was certainly smaller than either the Alpha or the Pentium, however, it was able to compete successfully against Sun, DEC and Intel with an out-of-order processor able to issue three integer-, two floating point- and a branch instruction per clock cycle. The processor shipped in the first quarter of 1996. Besides the excellent article, the advice of not changing tools or upgrading tools during a project is something many an embedded engineer wishes they had heard (and heeded) before.

3.4.5 TFsim

TFsim was described in Mauer *et al.* (2002). In the paper abstract, a post graduate student developed a full-system multiprocessor performance simulator that modeled a pipelined, out-of-order microarchitecture in detail in less than one person-year! The authors included Mark Hill (PhD in cache memory and instruction buffer performance) Hill (1987). The work was done at the University of Wisconsin–Madison; where they previously developed the Wisconsin Wind Tunnel for prototyping parallel computers Reinhardt *et al.* (1993). The computer science department included James Larus whose trace and profile work are mentioned in this survey and many trace patents. TFsim used previous work, as well as a commercial simulator — Virtutech `Simics`. Anyone trying to develop something this complex without the backing of such experts is going to have to dedicate much longer than a year to the project, by which time the work is obsolete by new processor announcements. A commercial license for `Simics` at non-academic prices must be added to the budget for industrial users.

Another full-system simulation effort was `SimOS`, which is covered in § 3.3.1, and took more than one person year in a setting where they had access to detailed architectural descriptions of their target device (Hennessy was a cofounder of MIPS) and no shortage of processor experts — Stanford University. Comparing the two would be difficult, as `SimOS` could execute many of the MIPS instructions native on MIPS hosts, whereas TFsim could simulate a SPARC V9 under `x86` or SPARC. The performance figures were given for SPARC V9 simulated on `x86` running Linux to boot Sun's Solaris.

The size of the simulation was 200 million instructions on all runs, except the OS boot which was simulated for 1,2 billion instructions. For a four processor system, this was approximately 50 million instructions.

3.4.6 OpenSPARC-64 Verification

Verification obviously involves simulation. Sun Microsystems have made the OpenSPARC T1 and T2 RTL "open source" at www.opensparc.net, with links to simulators, verification scripts, papers and scripts for loading the design into a Xilinx ML410 development board. In spite of good tools and prior successful generations of SPARC processors, software will be required for scripting and manipulating data. An excellent description of the effort involved to compliment Synopsys' RVM (Reference Verification Methodology) was Cavanagh *et al.* (2008). Although an eight-core 64-bit single chip multiprocessor might be an embedded overkill, for designers facing multi-core embedded products, debugging or verification is going to be challenging; taking a few hints from the OpenSPARC effort will be worthwhile. Each core is able to run a different operating system, and therefore, a different kernel. This might enable a Spring type of RTOS on a single chip with the benefit of being able to place it into a FPGA for instrumentation or running at speeds far faster than software simulation. (Spring had a system processor handling scheduling for several application processors — see (Stankovic *et al.*, 1999, pg 230) for a detailed description). There are several universities who have joined the OpenSPARC centers of excellence, companies have made derivative work available using open source tools (including the Verilog compiler), for example, Simply RISC, so embedded work using one of these core might not be that far fetched!

3.5 PowerPC Simulators

IBM published many tracing articles and own several trace patents. Unfortunately, the older IBM trace patents are unavailable for free. (Note in 2011: The Research journals are no longer free either).

3.5.1 MET

MET Moudgill *et al.* (1999) was a PowerPC/ AIX-based environment to explore design trade offs at IBM. MET's features resembled those of other simulation tools, such as the `SimpleScalar` toolset, ATOM (Analysis Tools with OM) and the `Shade` profiler. MET differed from these environments in key areas — it supported PowerPC architecture, achieved higher simulation speed and used different means to execute mispredicted instructions.

The tool was based on trace-driven modeling. The tools comprising MET were `Turandot` — a trace driven paramatised processor model; `Aria` — an execution-driven trace generator that could process user code and shared libraries (but not operating system code) and could also trace mispredicted instructions; and trace readers for various formats. `Aria`'s slow down was about 40 instructions for each instruction in the program being traced.

Some of the limitations mentioned were: inability to collect traces from operating system calls, kernel code or system I/O. Applications that include multiple processors could not be traced either. The simulated speed of 100,000 instructions per second was the original goal. Even with the throughput achieved, the authors reported that traversing through a trace of several billion instructions takes days, not hours, of simulation.

3.5.2 PowerPC VLIW Simulation

IBM Research described a simulation environment to evaluate a VLIW implementation of the PowerPC architecture Moreno *et al.* (1997). Two of the authors were also involved in MET (§ 3.5.1). Workloads examined were SPECint benchmarks and a set of AIX utilities — the slow down was 7 $\rightarrow$ 10$\times$ optimized PowerPC code, which was fast enough to run realistic experiments on a regular basis.

3.5.3 CMU Microarchitecture Workbench

The CMU (Carnegie Mellon University) framework for performance modeling was able to model several microprocessors; the DEC Alpha 21064 and 21164, and the PowerPC 601, 604 and 620. To calibrate the model, a system with a PowerPC 604 chip was used (examined the performance counters). The first version was the *infant model*. This model failed badly on most of the test suites. Figures given were only 51% of latency correctly modeled, only 30.6% instruction pipelining correctly modeled, and 100% failure rate for the random sequences. The next model was the *child model*, which sorted out many of the *infant model* bugs, however, instruction latency accuracy increased from 51 $\rightarrow$ 95.9%, pipeline modeling improved from 30.6 $\rightarrow$ 75.4%, 100% of the alpha tests passed and 84.7% of the beta tests.

The above system was described in Black and Shen (1998). The authors also noted that inspection (single step) cannot find performance problems, and that the many bugs to the model can only be weeded out using testbenches. These were automatically generated by the workbench, hand written and also used random cases. The models should be validated against the RTL models, however, the authors did not have access to these models. With such unstable models, designers who make changes may introduce new bugs, mask old bugs or the contribution of a bug to performance might be masked causing the results to be skewed and make wrong design choices.

During the validation of the results, traces were extracted from code running on real hardware, and then ran them through both the *infant* and *child* performance models. The primary source of error was attributed to trace gathering. The authors were also not sure how far into the libraries the trace extended. Trace driven performance evaluation is widely used by many computer architects and researchers—how good are *their* results? Mostly, they never validate the results as there

is no real hardware or the trace nightmare is side-stepped. In this case, the traces were generated by an automatic test pattern generator (ATPG) built into the microarchitecture workbench that fed a trace generator. Perhaps the trace generator was not developed at CMU.

The 604 PowerPC is relatively old and also not that complicated compared to devices shipping in 2008 (almost ten years after the original article). The article highlights the effort involved in developing simulators. Most researchers do not have access to the RTL of modern processors, and will definitely not have access to any detailed architectural design before the processors have been described at a major microprocessor forum or patented. The one author, Bryan Black, actually spent four years at Motorola and was a member of the PowerPC 604 design team.

3.5.4 BGLsim

The Blue Gene "pseudo cycle-accurate" simulator was designed for simulating parallel machines, including all key features of the architecture — processors, FPU, cache, memory and interconnects. BGLsim ran 100 $\rightarrow$ 1000$\times$ faster than a cycle-accurate simulator and was briefly covered in Martorell *et al.* (2005). Although a detailed project involving many people, the discrepancy for a small section of code between the simulator and the actual hardware was 7.5% (Martorell *et al.*, 2005, pg 414). This exposes the problems of accurate simulators, as the grand challenge problems need to be tuned on loops that can only rely on simulation output, as the hardware monitors do not expose pipeline stalls or cache problems. The Blue Gene project cost several million dollars (not sure what the budget was, but see the *IBM Research and Development Journal* issue dedicated to the Blue Gene (Volume 49) to see the scale of the project — possibly in the hundreds of millions of dollars).

3.5.5 AlphaWorks CELL

The AlphaWorks team are part of IBM and developed several simulators for the CELL processor. One of the simulators was for the 64-bit PowerPC 970 which is similar to the PowerPC core in the CELL chip. The binaries are available for downloading. The Mambo project was a full system simulator able to boot Linux. Output of the Mambo infrastructure was a debugging interface, trace output and visualization facilities.

The host platforms supported were Linux, AIX, OS/X on PowerPC, `x86` and `x86-64`. The Mambo project was described in Peterson *et al.* (2006).

3.5.6 Microsoft Xbox 360

The Microsoft Xbox 360 had a single chip incorporating three 64-bit PowerPC cores, cache, a very high speed PHY (Physical Layer Device) and debug/ test in 165 million transistors. From the picture of the Xbox 360 die, the test/ debug portion of the chip takes less than 10% of the die area. The following extract was taken from Brown (2005):

> One of the priorities in developing the Xbox 360 system was getting a reliable chip on the first pass. The first pass hardware ran with bus, CPU, and cache all at full speed, and a demo game was running on it one week from power-on. This accomplishment was the result of an exhaustive testing and verification process throughout the design of the system.
>
> Verification was both parallel and hierarchical. Each component was verified separately, but larger units were also verified as a group. As much as possible, tests were designed to catch defects where they could be corrected quickly. The process was focused on quality, with quality measure-

ment standards and an extensive review process. IBM verification tools were used along with industry tools. Formal verification was used where appropriate. Another critical component of testing was intelligent randomized test generation, to stress-test components in a variety of circumstances. Hardware acceleration was used to make simulations more practical, using hardware/software co-simulation.

Because of this, the kernel code to be run on the system was in testing and verification long before the first pass chip arrived; this allowed testing, not only of the kernel code, but of the chip on which it was being run (in simulation). The end result was a successful Pass 1 hardware, and the move from first silicon to volume production took only eight months.

From news feeds, Microsoft developers purchased several Apple G5 dual-core PowerPC systems before first silicon, as the G5 was the closest available system to the new Xbox 360.

3.5.7 PowerPC 405 Simulator

IBM and others with an interest in the PowerPC architecture formed the *Power.org* alliance. The intention was to promote the architecture in products ranging from deeply embedded (even in phones) to mainframes. IBM had already made inroads in the gaming console business but embedded competitors like MIPS Technologies and ARM (Advanced RISC Machines) survived almost exclusively on selling IP (Intellectual Property). IBM had an advantage of being a silicon foundry as well as using the PowerPC in its own products. There have been various announcements of tool suppliers for the 405 and later cores, as well as AMCC's purchase of the 4xx IP IBM Microelectronics (2004). I was rather surprised to see IBM later offer the HDL (Hardware Description Language) for the PowerPC 405 Brandow (2005). (AMCC is also a Power.org member). In January, 2008, AMCC announced a deal with IBM for closer ties with the 4xx family. *Google* searches were not very useful to try obtain the monetary value to AMCC, but some of the searching uncovered problems with the original transfer in 2005 due to IBM staff in France being transferred to AMCC, who subsequently lost their jobs.

While I was in academia, I looked at the offer, however, did not make any effort to download the code. Xilinx also included the 405 simulator with their EDK for the Virtex-II Pro family, however, as mentioned elsewhere, the trace option did not work.

There are plenty of PowerPC 405 evaluation boards available with trace ports to compare the software simulators to real hardware, but it is probably more cost effective to build a debug infrastructure around a Xilinx Virtex-V FX evaluation board with a hard-core PowerPC embedded in the FPGA and running sections of code. (There are several ICE (In-Circuit Emulation) patents on simulating one chip with another, and also check pointing for running sections of code on actual "hardware-in-the-loop".)

3.6 g88 — Motorola 88000 Simulator

This was mentioned for completeness. The *g88* simulator software was often referenced in benchmarking and several simulation papers. The processor is no longer supported by Motorola, who since partnered with IBM on the PowerPC.

3.7 Alpha

The DEC Alpha was the fastest microprocessor for most of its brief lifetime. The high clock rate and mismatch with available memory meant

that aggressive architectural designs were required to maintain leadership. There were several projects aimed at performance monitoring, and many aimed at design exploration. The DCPI profiling work and ATOM tools were covered in other chapters, but the simulation research would always be tricky for academic purposes. Academic publishing would trail DEC as they were designing the new devices, and judging by other research teams, the time to develop a decent simulator is several man-years of effort even when reusing prior work. The novelty factor would be too risky unless researchers recommended new features which they could show as improved on a reasonably accurate processor model.

Even Compaq (who acquired DEC) seemed to struggle maintaining processor models. The challenge was described in Emer *et al.* (2002). `Asim` was a design to reuse modules for design exploration that could be configured using a graphical interface. The dependencies would be tracked and automatically generate an executable model for design exploration. The tools would also track the test cases and results of the different runs and parameters used to generate the tools. The `Asim` abstract almost placed the article into one of the quality projects rather than mentioning that designing processors is extremely high pressure when everyone is trying to do design exploration on a new device that does not yet exist on tools that have yet to be written. (And only when design exploration is complete can the chip design actually start). Even the collection of tools and the configuration mentioned in `Asim` would need "slash and burn" type of farming methods when the pizza boxes pile up and the car parks are still full late into the evenings. The authors were at Intel without any mention of which Alpha design teams they were in, which might have been how the split of DEC happened with the semiconductor part going to Intel and the computer side going to Compaq.

An article written four years earlier than the above cleanup of the code was Reilly and Edmondson (1998). This was a more realistic picture of what other teams also described in developing the next generation chip within a market window. The graphics to display the pipeline effects were based on Veenstra and Fowler's MINT project Veenstra and Fowler (1994*a*), which previously modeled the MIPS R3000. The Alpha 21264 pipeline was obviously much more complicated than the Sandcraft SR1-GX 64-bit MIPS, which used `Mint+` to display the waterfall plot of the pipeline results. (See Choquette *et al.* (1999) for the Sandcraft pipeline plots). The `Mint+` article appeared almost a year after the 21264 article. The 21264 Alpha model was written in `C` over a period of more than a year of intense development involving about 20 people. The authors mentioned that the use of conditional compilation would have spelled disaster if they were planning a product for shipping to customers. "Our goals did not include shipping the performance model; it was an exploration tool. Time spent on fancy fasteners is wasted when duct tape will do the job."

In the *What We Learned* section of Reilly and Edmondson (1998), the graphical tools were useful in understanding program behavior, but not as universally useful in debugging modeling problems. To be comprehensible to humans, graphical representations must focus on fairly narrow windows in time — a few hundred cycles. However, they found that most of the interesting model bugs were sequences spanning hundreds or even thousands of cycles. The model's debug trace facility proved indispensable for finding these types of errors.

3.8 ARM Simulators

These are mainly commercial to support SoC designers and software teams before silicon availability.

The accuracy of an ARM11 cycle-accurate simulator was studied by Samsung, an ARM licensee, in Chung *et al.* (2006). Although several efforts at validating simulators were published, this was the first that the authors were aware of that considered a recent ARM

processor. Previous cycle-accurate work on `SimpleScalar-ARM` was shown to be cycle-accurate compared to ARM-7 silicon, however, there are several architectural differences between ARM-7 and ARM-11.

The evaluations compared `Sim-Outorder`, `Sim-ARM1136`, and simulating ARM1136 RTL. Major differences in features between the `ARM1136` and `Sim-Outorder` were 8-stage versus 5-stage pipeline, no L2 cache in `ARM1136`, `Sim-Outorder` only accepting ARM-7 binaries, plus several other differences summarized in Table 3.1. Note that `Sim-ARM1136` was a modified version of `Sim-Outorder`, which was done as part of the ARM1136 cycle-accurate study by the authors of Chung *et al.* (2006).

Table 3.1: ARM1136 RTL compared to simulation

Test	Sim-ARM1136	Sim-Outorder
Dhrystone 1-iteration without branch predictor	0	24% loss
Dhrystone 100-iteration	3%	36% lower IPC
EEMBC default iteration		average differences as much as 19%
LDM/STM IPC		IPC difference as much as 38%

The ARM processor is simple compared to PowerPC, and even to MIPS, however, when the main architecture vendor does not maintain widely used simulators, one needs to question their commitment. Samsung certainly has funding for advanced EDA tools and as a licensee of ARM, one would assume that they received a heap of software IP when signing any license. Judging by press releases, ARM has done an excellent job of testing cores in silicon when releasing new features, but their publicly available tools do not receive the same attention. There were notices on ARM's website about support for the GNU tool chain. The multitude of ARM licensees have made silicon and evaluation boards available. EDA support is also good for ARM and FPGA trials should be even easier with the royalty-free Cortex core in Actel and Altera FPGAs. (Cortex only runs Thumb-2 code, but there are several FPGA test vehicles for ARM).

3.9 Intel/AMD Simulators

Intel and AMD are the leading `x86` vendors. AMD was first to extend the `x86` to 64-bit, while Intel does provide 64-bit compatible extensions to the IA-32 (`x86` core). However, Intel's main 64-bit thrust was in the Itanium or IA-64 architecture. These `x86` variants have been grouped together, even though the processors differ widely and are not true RISC devices or suitable for deeply embedded designs. These 64-bit devices are more than likely to be used for host work, particularly in 64-bit multicore configurations due to low cost, widespread availability, excellent performance and available operating system support. Their support requirements and interfaces will provide a good reference for any embedded work.

Intel developed many simulators for internal use, particularly design exploration, and later for customers to port software prior to IA-64 silicon. The following subsections covers several simulators, particularly those that were part of a larger framework. The AMD efforts are also added to this section. The AMD K5

efforts would be interesting for any testbed development plus the more recent AMD simulators for 64-bit multicores.

3.9.1 SoftSDV

SoftSDV was described in Uhlig *et al.* (1999). The aim was to provide software developers with a presilicon IA-64 platform. SoftSDV used dynamic binary translation, dynamic resource analysis and I/O (input/ output)-device proxying. There were three levels of simulation detail with speeds ranging from 10s of MIPS (Millions of Instructions per Second) for the fast emulator, around 200 thousand instructions per second for the analyzer, and between fifteen- and 34,8 thousand instructions per second for the detailed microarchitectural simulation. The tradeoff was speed, accuracy and completeness. The fast emulator was able to boot an operating system, position the workload at an area of interest and then change to the more detailed (and slower) simulator for compiler tuning or performance analysis. The detailed simulator was used for microarchitecture exploration which changed frequently before the design of the architecture stabilized.

SoftSDV was able to model up to 32 processors in a symmetric shared-memory configuration. Only one processor was simulated at a time with the simulated processors being switched in a round-robin order. All processors shared memory, translations and all simulated platform devices. The IA-64 architecture was simulated on a 32-bit IA-32 device with an average of 25 IA-32 instructions per one IA-64 instruction. The simulated address was also 64-bit, and had to be mapped into the host's address space without crashing the host. 64 Mbytes of memory was reserved in a 256 Meg host, although in the results section, a 512 MByte Pentium III at 500 MHz was used.

Of particular interest to embedded developers is the method of handling I/O via the *IO-device proxying* and using a small riser board with some mapping logic for address translation, interrupt mapping and handling the virtual PCI (Peripheral Component Interconnect) base address registers, as the host would have initialized all the PCI devices before loading the SoftSDV as a user-level program, which would then try to re-initialize the I/O as part of booting an operating system for testing.

The development involved three different teams (Israel, Santa Clara and Oregan), and was used to provide presilicon software development platform. The proof of the usefulness was being able to boot Windows 2000 and Trillian Linux within ten days of receiving Itanium first silicon. During this time, the hardware was also being validated for temperature, speed and physical checks. Once the hardware issues were resolved, other operating systems were ported quickly — under three hours for IBM's Monterey-64.

SoftSDV was able to switch between the various levels of detail, execute a detailed mode for some time, and then return back to the fast mode. Special *markers* were compiled around regions of interest in a workload, which SoftSDV would recognize as the workload executed, and could dynamically switch between speed-accuracy modes. This work predates US Patent N° 7 149 676 B2 Krishnan (2006).

From the architectural description and simulator design, it should become apparent to the WCET community that casual users of the IA-64 was unlikely to obtain accurate timing of large sections of code for real-time work. So much depends on hardware speculation, rewinding on mispredicted state, and grouping of instructions in the VLIW architecture, that even measurements on actual hardware will seldom produce identical results. Stepping between revisions of silicon are also too frequent for people outside of Intel to track for any WCET tools to be useful in a setting that does not rely on publishing results for ten years or older silicon.

3.9.2 Inferno

Inferno, described in Wang *et al.* (2003), was a simulation framework developed at Intel starting in early 1998 for a research Itanium processor. The simulator was able to explore micro-architectural scenarios with cycle accurate timing as well as a fast mode for booting an operating system. The framework was written in `C++`. Different simulators were managed via APIs to facilitate reuse and modular software, as a single program would be unmaintainable, as was the case with prior Pentium simulators at Intel. The faster simulator used *SoftSDV* which was written by Intel, and used binary translation techniques to group several instructions rather than operate on individual instructions. The internals were described in enough detail to impress on any future simulator developers that a modular approach (and most likely `C++`) would be required, otherwise it is too difficult to validate any models. The Itanium and modern desktop processors have to resort to concurrent execution of several stages, speculative pre-fetch of instructions, rewinding of mispredicted paths, out-of-order execution, etc., making general purpose simulators unsuitable.

3.9.3 Giza

Presilicon validation using operating system kernels was described in Carver *et al.* (1999). Two simulators were used; Giza and SoftSDV, however, SoftSDV was described in a prior *Intel Technology Journal* and therefore only Giza was described. Giza was built around an instruction accurate software simulator for the Itanium ISA, which handled multiprocessor, the front-side-bus, TLB and caches, and connected to software models of standard Itanium devices and chipsets. By using functional simulators before testing on the RTL model, valuable time was saved in discovering errors before the slower RTL code was exercised. When the kernels were run in RTL, a RTL checker was run at the same time to compare architectural state after instructions were retired. Any mismatches flagged an error which would require further analysis to isolate the cause of the problem.

In the *Porting Challenges* section of Carver *et al.* (1999), a post-processor (AfterBurner), was required to modify compiler generated assembly code and fix various problems that prevented running the code in the RTL model. Several errors in tools and RTL were discovered during the porting that were not discovered using standard processor validation tools in use at Intel.

The authors described checkpointing the processor in the functional model so that the RTL model could jump to any part of the kernel without having to run the initialization cycles in slow RTL. It also allowed simulation in parallel by starting at different locations on several machines and then running RTL code. There was a large overlap to reduce cold cache effects.

The kernel code was modified to interface with the functional simulator and to generate trace information for context switches, interrupts, process creation and termination, and address space modifications.

The comprehensive description of moving from a functional simulation to a more detailed RTL model predates US Patent 7 149 676 B2 of 12^{th} December, 2006, Krishnan (2006).

A note on simulation accuracy for the presilicon Itanium framework and compiler compared to actual hardware was taken from Dulong *et al.* (2001); SPECint95 figures were optimistic by 13% (Vortex was out by almost 30%), and the ranges of negative and positive 40% for the SPECfp95 benchmarks. The lack of memory overlapped fetches and writes in the simulator was given the blame. Such a large discrepancy on a five year presilicon effort is not encouraging for the WCET community, who have less motivation than a well-funded manufacturer trying desperately to change architectures two years after promised silicon.

3.9.4 AMD K5 emulation

The K5 was an x86 compatible processor designed from scratch after the AMD–Intel split. Although the verification articles referred to simulation of the ISA, the aim of the K5 verification was ensure that binaries for x86 could run on the chip. To speed up emulation, a collection of almost 4000 FPGAs was configured to partition the design and run in hardware. The design was for a 4 million transistor CPU. When Ganapathy *et al.* (1996) was written, the fastest hardware accelerators could simulate a full gate level of the K5 in the low 100 Hz range, which made booting Windows or running applications like Excel or Word impractical. The hardware jointly developed with Quickturn allowed execution in the 100 kHz range — three to six orders of magnitude faster.

The K5 hardware emulation project had the following phases:

Methodology Phase Quickturn R&D team and AMD engineers ran small test cases of a few hundred clock cycles to verify the tools, environment and optimal partitioning of the design.

Validation Phase The aim was to verify the K5 in emulation with test software. In parallel, an emulator board was built to run in the target system at emulation frequencies.

Systems Verification Phase tested the ICE interface by executing diagnostics from ROM (Read-only memory) on the motherboard, followed by x86 applications that ran billions of cycles to check compatibility and functionality issues before tapeout.

Silicon Test Support Phase was used to locate and fix errors found in the silicon, as the test environment was more "user friendly" than probing silicon. Once the tests were verified under emulation, changes were committed to the design database.

Systems Regression Phase was used to verify each revision before tapeout.

The emulation environment helped to run several operating systems before first silicon, as well as uncover several 16- and 32-bit errors. The first silicon was functional and allowed full speed applications to run and quickly discover "difficult to find" errors.

The FPGA based design was connected by roughly 400 cables. Thousands of FPGAs were mentioned in Ganapathy *et al.* (1996) (over 4000 in Christie (1996)). A major difference between the K5 and FPGA design was the use of a non-overlapped two phase clock and level sensitive latch based design rather than FPGA edge triggered latches. See McMinn and Ganapathy (1998) for the patent solving this problem. Basically, the high speed FPGA clock is gated with the system clock to produce a high frequency enable clock, which is used to clock an edge triggered latch in the FPGA emulating a level sensitive clock in the final design.

A major deviation in the design of AMD's first x86 processor (without Intel's help), was the use of a RISC microarchitecture. In (Christie, 1996, pg 18), two years prior to starting the K5 project they had designed a superscalar version of the AM29000 RISC processor with four-instruction issue and full out-of-order speculative execution. By late 1990, they had taken this design in logic simulation to the point of booting Minix. The K5 took four years from start to first silicon, relying extensively on software modeling, and as seen from the 4000 plus FPGAs, also on hardware acceleration. Hardware modeling on FPGAs is now a large market, with some reasonably priced products from companies like Aldec, Harps and Synplicity.

3.9.5 AMD 64-bit Simulator

AMD make simulation tools available to vendors and researchers. The free tools have fewer features, however, they are able to model a dual-core AMD Opteron processor. Unlike Intel trying to simulate the IA-64 on a 32-bit IA-32 platform, SimNow requires a 64-bit AMD

host, as the simulator relies on dynamic instruction translation for speed. SimNow is not cycle-accurate, but uses a "tick" which is one instruction long. The user-interface is via a cross-platform GUI (Graphical User Interface) based on Qt.

SimNow allows simulation of a complete PC by incorporating devices that are common on a typical PC platform. Disk images are created by an AMD supplied tool. ROM can also be simulated for BIOS (Basic Input Output System) developers. The debug interface was similar to typical monitor program commands, however, GDB (GNU debugger) was also available. There were several performance tools, compilers, libraries and documentation at AMD Inc. (2008*a*), and a comprehensive 245 page manual at AMD Inc. (2008*b*).

3.10 iWatcher

The *Intelligent Watcher*, `iWatcher` Zhou *et al.* (2004), was hardware support for debugging software. The hardware or the architectural support was simulated and consisted of modifications to the cache, pipeline, thread level speculation to run microthreads for the monitoring functions, and user callable functions to switch the monitoring on or off. The *iWatcherOn* and *iWatcherOff* functions were passed parameters for the start of the area to monitor, its length, type of access (read, write), reaction mode (report, break or rollback), and the monitoring function. `iWatcher` was classified as a dynamic monitor which used *location-controlled monitoring*.

The authors were interested in being able to debug software for stack smashing attacks, memory allocation errors, values used before being initialized, access to a memory location whether by indirect pointer or value, and being able to selectively debug sections of a program. Slowdowns and overheads were fairly high for a hardware based system–ranging from 4 $\rightarrow$ 80% for the various tests. The system was compared against `Valgrind`, an open source program for `x86` that simulated each instruction being monitored for program errors (memory allocation, access and others). The hardware proposal's overhead was much smaller than `Valgrind`'s (25 $\rightarrow$ 169$\times$ less for detecting memory corruption, dynamic buffer overflow, memory leak bugs and a combination of these bugs).

Limitations cited by the authors for other dynamic monitors was that they relied on compilers or preprocessing tools to insert instrumentation. This meant that more instrumentation was inserted than might have been necessary. For hardware breakpoints, most hardware has a limited number of code- or data breakpoints, whereas `iWatcher` was able to monitor any location or several ranges of locations. They also claimed that `iWatcher` was language independent, presumably because the traps were hardware assisted.

The closest hardware to `iWatcher` in embedded processors is a breakpoint on code or data addresses, however, the monitoring functions to call are not programmable for each address and the user generally does not have access to threads that can run the debug code, or know what is in L1 or L2 cache to tag cache lines in both caches with WatchFlags.

Deeply embedded cores like the ARM with Flash might only have two hardware breakpoints. For RAM (Random Access Memory) resident code, software breakpoints can be inserted anywhere, however, these will only trigger if code tries to execute the breakpoint instruction, not if a pointer accesses some variable on a breakpoint instruction.

In the *Conclusions and Future Work* section, the `iWatcherOn/Off` calls were inserted manually; the next version would examine other methods of inserting instrumentation (compiler or instrumentation tool).

What are the chances of seeing `iWatcher`-type hardware embedded into a processor? It would be important that some access to external trace hardware could correlate the hardware assisted software instrumentation. The idea of a logic analyzer trigger option with user selectable functions on any address match

is attractive, but there are several limitations which are covered in the thesis. Briefly, to be practical on any existing hardware, an external FPGA would be required for the instrumentation. The memory interface would have to be routed through the FPGA to be able to capture addresses, match on instructions and possible extended width for additional fields (trace on, profile, trap etc.), and the processor would need to have static RAM, unless you had the courage to debug a DDR (Double Data Rate) interface. JTAG probes are too narrow to allow tracing and with the new processor introductions in 2008, multicore debugging is not going to become any easier and manufacturers (other than AMCC in the 4xx PowerPC, Atmel AVR32 and the ARM ETM) are not giving end users trace ports. Dedicating large areas of a core to debugging should be common if the manufacturers want programmers to use the high-end devices, otherwise we have to believe them that multi-threading and SMT are the answer when the obvious is more contention to the memory which is already the bottleneck. Without measurements or having control over which threads are active (from a high-level language), there is not much end-users can do if there is an operating system between your code and the processor.

Finally, on `iWatcher`, there was a global *MonitorFlag* that could enable or disable monitoring of all watched locations. This will certainly be necessary for any FPGA or hardware instrumented testbed that also relies on software instrumentation, so that code can run at full speed without having to recompile the software or modify the FPGA VHDL, followed by downloading everything all over again. The *MonitorFlag* could disable tracing, or whatever after any selected event and continue running, whereas any modification to source code or FPGA configuration will require another download, and therefore implies starting from the bootup vector. The *MonitorFlag* could also be in series with a user switch or external trigger into the FPGA to start the instrumentation by enabling one flag.

3.11 Processor Independent

Although the section title suggests that any processor can be simulated, there are packages that can simulate more than one processor, and generally have to be recompiled or modified for different architectures or even processor derivatives within an architecture. General purpose simulators have been built from an "architectural description language", for example the LISA modeling language from RWTH, Aachen. Configurable processors from Tensilica require configurable tool chains, which are generated automatically by their development platforms. These systems are not considered in this survey, not because they are uninteresting, but they require an investment in time that embedded developers on short-term projects do not have.

3.11.1 SimpleScalar

The `SimpleScalar` tools were written in 1992 as part of the Multiscalar project at the University of Wisconsin. The internal ISA was called PISA (Portable Instruction Set Architecture), a MIPS-like instruction set with 64-bit long opcodes. Various authors modified the `SimpleScalar` tools for other processors — PowerPC, DEC Alpha. There are several articles on the `SimpleScalar` simulator. A technical report described how to install the tools, the internals and download details Burger *et al.* (1996). It was originally written by Todd Austin and modified by Doug Burger for his PhD (Burger, 1998, pg 24).

Two articles described the `sim-alpha` version of the `SimpleScalar` tool kit. In Desikan *et al.* (2001*b*), the authors chose the Alpha 21264 processor because the specifications of its microarchitecture were available in more detail than chips from other vendors; a point that vendors should note as chips become more complex and simulation will play a greater role in optimization. `Sim-alpha` was validated against a Compaq DS-10L workstation based on a 21264 Alpha processor. The

motivation was to reduce the error between simulation and real hardware. As the authors pointed out, errors in various simulators not validated against real hardware were often more than the gains their authors reported for new microarchitecture features. The mean error for various microbenchmarks on the validated `sim-alpha` was 2% compared to a mean of 19.5% for their earlier `sim-outorder` work (part of `SimpleScalar 3.0b`). In spite of the authors' efforts to obtain documentation, there were a large number of unknowns in the memory architecture, which resulted in an 18% error in macrobenchmarks executing the SPEC2000 suite. The timing section of `sim-alpha` was written from scratch with a large effort to model the cache, external memory, experiment with page mapping effects and other architectural features (store-wait table, speculation, branch prediction, register renaming etc.). A longer version of the above article appeared as a technical report Desikan *et al.* (2001*a*).

The February, 2002 edition of IEEE Computer journal had several articles on simulators; `SimpleScalar` was include from a historical perspective in Austin *et al.* (2002), where the authors noted that in 2000, more than one third of all papers published in top computer architecture conferences used `SimpleScalar`—impressive indeed! The supported processors included ARM, PowerPC, `x86` and Alpha. There were plans to commercialize the toolkit to better support the large user base.

3.11.2 Brown Simulator v2

The Brown Simulator Lango *et al.* (1998) was a high-level machine simulator intended for operating system prototyping. It provided support for context switching, programmed and DMA I/O, virtual memory and multiprocessor support (untested). The aim was to study operating systems and as such the simulator did not simulate actual machine code, but was merely a library of functions callable from `C/C++` that provided functionality of actual hardware. There were references to functions for multiprocessor, setting interrupt priority levels, traps and bus errors. The software was available under a GNU Public License. A rather useful feature was the "terminal device" which allowed reading and writing characters to a display, which is an idea similar to the JTAG debug agents in many IDE (Integrated Development Environment)/ tool chains.

3.11.3 Simics

`Simics` was originally developed at the Swedish Institute of Computer Science and spun out as a commercial venture in 1998. According to their website, the team behind Virtutech were the first in academia to run commercial server operating systems in a simulation framework (I assume this was before `SimOS`). An article Magnusson *et al.* (2002), described the `Simics` history; development was based on `g88` in 1991 and extended to include multiprocessor support. In 1994, the `gsim` simulator was rewritten as a multiprocessor SPARC V8 model, which was the first version of `Simics`. At the time of the article (Feb 2002), the development had taken more than 50-person years with a code base of close to a million lines.

Also in the same article, a VxWorks splash screen of a PowerPC booting was shown. I am unsure if that implies the product is suitable for simulating a real-time system.

More recently (2007), Virtutech as one of the Power.org members was reported to supplying IBM with several hundred `Simics` licences for the Power6 platform. The report also claimed that IBM will be using `Simics` as its single infrastructure across several PowerPC platforms as well as supporting other Power.org organizations. Confirming the claims was Freescale's July (2007) announcement of `Simics` models for the announced MPC8572E multi-core for early adopters before actual silicon Freescale Semiconductor (2007). The Freescale announcement was for a partnership that would accelerate migration from single- to multi-core devices and also offer a fast and accu-

rate hybrid environment that could dynamically switch between a functional-accurate model to a cycle-accurate model for real performance prediction. Freescale was offering free access to the 8572 simulator for a limited period. In 2008, the brochure announcing the eight-core QorIQ P4080 PowerPC processor from Freescale mentioned the Virtutech partnership.

It will be interesting to see how the holder of US Patent N^o 7 149 676 B2 responds to being able to dynamically switch between levels of simulation.

Although accurate models are necessary, consider a quotation regarding realistic workloads.

> In most cases, we do not know how to implement an accurate model with performance sufficient to run realistic workloads. So, in practice, models that attempt to be highly accurate end up running very small "toy" workloads. The result is accurate answers to irrelevant questions. Magnusson *et al.* (2002)

3.12 Summary

Simulating long traces can take days, however, by placing trace data into separate directories and interval files, one can jump to a particular location in a trace. An elegant solution was given in *Jumpshot* which allows jumping to any location without traversing the whole trace Wu *et al.* (2000). Simulators in compiler tool chains are directed at software debugging. The Xilinx PowerPC 405 ISS (Instruction Set Simulator) was meant to emit trace data, however, that part of the simulator was not functional. GHS (Green Hills Software) included ISS as part of their compiler tool chains — when down loading a program the user can choose between the ISS or whatever JTAG server is loaded. These simulators cannot be extended due to the proprietary nature of commercial software.

MIPS appeared to have an early lead in simulation work. The PISA ISA in the `SimpleScalar` work, Hennessy and Patterson's DLX, the relatively simple R3000 and pioneering work gave MIPS a high profile. MIPS's squabbles with Lexra have not helped their cause in the deeply embedded area, which firmly belongs to ARM.

SPARC tools were also popular, possibly due to Sun's business model of being "open" together with widespread use of their workstations.

IBM launched the PowerPC architecture after SPARC, MIPS and ARM. Until the recent launch of Power.org (2005), they never gave away any VHDL or models. The PowerPC was also more complex which would discourage short term projects. Any long term work is soon eclipsed by new processor introductions and IBM's own staff to compete with for publications (they have inside knowledge of what has yet to appear).

When OpenCores were going to develop an ARM compatible device, there were legal threats. Anyway, soft cores are here to stay for low volume (even moderate volume) and experimentation, so simulation might not be as important to smaller teams who will simply churn through the "edit-compile-download-measure" cycle, and test live hardware at rates that software solutions will have difficulty matching. There is also no long waiting time for "first silicon" so the effort that would be put into tools to improve chances of correct silicon merely take away resources from working on a FPGA. The expense of masks or NRE (Non-recoverable engineering costs) are not an issue on a soft core which allows more experimentation and less reliance on a simulator.

Chapter 4

Tracing via Instruction Modification

"A tool generating insufficient information is of no use to the user." (Srivastava and Eustace, 1994, pg 1)

4.1 Introduction to Tracing via Instruction Modification

This method of tracing requires either modification of the original source code, a modified compiler, or instrumenting the binary images. For small programs, simple manual methods can be used (like *printf()*'s in the source code), however, for large programs, automated methods become necessary. Most of the systems described in this chapter were research projects or intended for high-end processors. In ATOM, fully instrumented slowdowns were given as 12×, however, in the RAMP (Research Accelerator for Multiple Processors) letter for NSF (National Science Foundation) funding, the authors noted that the SPEC2005 benchmark programs would have to run at least 600 seconds on a machine that had a rating of ≈ 700 for integer codes and ≈ 900 for floating point codes on the SPEC2000 benchmarks. Trying to simulate these on currently available machines would be an exercise in futility. The authors also pointed out that the average simulation time on a 2005 class workstation simulating a SPEC2000 benchmark on a cycle accurate simulator took on average five days per benchmark (Arvind *et al.*, 2005, pg 18).

Embedded systems are improving in speed, but trying to simulate any non-trivial program may not be practical, particularly real-time work where the environment also requires simulation.

In Srivastava and Eustace (1994), three generations of program analysis tools were classified as follows:

- First generation tools for Basic block counting (`pixie`, `epoxie` and `QPT` (Quick Profiler/Tracer)).
- Second generation tools for Address tracing (`pixie` and `QPT` could also generate traces). These included WRL (Western Research Laboratory) Titan tracing and analysis, MPTRACE (Multiprocessor Trace), `ATUM`, `Tango Lite`, `PROTEUS`, `g88` and `Shade`.
- ATOM was considered a third generation tool.

The information is this chapter was taken from the various articles on the tools, and used the most cited references or the easiest available references. None of the tools were tested, and where available, the source was downloaded to take a glimpse. The tools were generally used for studying cache or compiler output, and would therefore be highly architecture (and often processor model) specific. None of the tools were aimed at embedded or real-time targets, however, instrumentation, infrastructure requirements and analysis for embedded targets would have a lot in common with the 1990s workstations as these RISC devices moved into the embedded space. Unfortunately, the processor with the best tool support, the Alpha, has silently disappeared.

4.2 ATOM

According to (Eustace and Chen, 1995, pg 15), Amitabh Srivastava wrote the original version of ATOM, and Greg Lueck modified that version to support shared libraries and kernel instrumentation. ATOM was one of the many tools developed by DEC for the Alpha processor. ATOM was described in a technical report, Srivastava and Eustace (1994), as a tool building system that allowed selective instrumentation, with data passed via procedure calls to functions linked into the final program (ran in the same address space). The tool was independent of compiler and language system, as

it operated on object modules. ATOM allowed a procedure call to be placed before or after any program, procedure, basic block or instruction; giving different views of the final executable. The analysis routines were user supplied. According to the technical report, (Srivastava and Eustace, 1994, pg 16), ATOM's fast communication between application and analysis means that there is no need to record traces as all the data is immediately processed, and the final results are computed in one execution of the instrumented program. The times taken to instrument the 20 SPEC92 benchmark programs ranged from 97 $\rightarrow$ 257 seconds. The execution time for the cache analysis tool was $11,84\times$ slower compared to the uninstrumented version, however, the cache tool instrumented each memory reference.

In (Lo *et al.*, 1998, pg 3), an early version of ATOM only generated user-level traces.

Later, Alan Eustice (DEC) and J. Bradley Chen (Harvard University) Eustace and Chen (1995), described methods of using ATOM tools to instrument kernel code, however, the standard ATOM tools were designed for user-level code. Their user guides explained how to instrument and trace both system and user programs using the ATOM tools Chen and Eustace (1995); Eustace and Chen (1995). The ATOM tools consist of four programs; *ksample*, a multiprocessor debugging tool; *kgprof*, a `gprof`-like tool for sampling the kernel; *k3rd*, a tool for detecting kernel bugs; and *BT*, a branch prediction tool. In (Mogul and Ramakrishnan, 1997, pg 237), program slicing was described and inserting software 'spikes' into the code using ATOM.

ATOM only occurred a 1% slow down on Alpha programs when instrumenting system calls, otherwise nearly $12\times$ slower for fully instrumented versions. The authors recommended leaving the instrumentation permanently installed for the instrumented calls, as the penalty was only 1%.

ATOM together with `PatchWrx` was used to instrument the Alpha version of Microsoft's NT, and was described in Casmira *et al.* (1998). Of interest was the mention that the Alpha "PALcode was developed to provide VAX compatibility and efficient system utilities, while also providing a vehicle for efficient trace capture". As an indication of the amount of operating system activity in Oracle, when capturing application-only traces, the 45 MB trace buffer took a few minutes to fill, whereas when the operating system was traced as well, the trace buffer filled in a few seconds.

In (Srivastava and Eustace, 1994, pg 4), `OM` was initially developed on DECStations running under Ultrix, and was therefore able to work with different architectures. As ATOM was built using `OM`, it should be able to run on different architectures, however, the DEC technical reports only mention ATOM running on Alpha processors. There was certainly no point in spending effort on obsolete workstations once the Alpha started shipping. ATOM did not "steal" any registers from the application program, but allocated space on the stack before the call, saved the registers that might be modified during the call, restored the registers after the call, and deallocated the stack space.

4.3 HALT

`HALT` Young and Smith (1996), the *Harvard Atom-Like Tool*, allowed researchers to instrument their conditional branch instructions using `SUIF`. To research the branch problem, it sufficed to condense a program run into a *branch stream*. In `HALT`, a similar approach to ATOM was used to modify the software program so that the branch stream could be monitored by user-supplied analysis code. No additional hardware was required, however, the instrumentation did dilate the code (not mentioned by how much).

`HALT` worked as a `SUIF` pass, adding calls to analysis code either to original `SUIF` files or to `SUIF` machine library files. Since the output of `HALT` was also `SUIF` code, further `SUIF` passes could be run after inserting analysis code. Typically, minimal analysis code wrote a branch trace to a file which was post-processed

with various analysis programs. HALT was not as flexible as ATOM; it only supported instrumentation of initialization, branches, function entry and exit, and termination routines. HALT inserted calls to analysis routines as directed by branch_unique_num annotations, which were given unique non-negative numbers by another labeling pass program.

The trace could be piped to another program for consumption and analyzed as the instrumented program ran, or the trace could be saved to disk to ensure experimental repeatability. The authors previously implemented a system called SCBP (Static Correlated Branch Prediction) for their branch prediction research. SCBP had four major stages: profiling, local analysis, global analysis and code layout. SCBP used a data structure called a *history tree* to describe each branch of a program. HALT provided the profiles, while other SUIF passes performed some parts of layout. Once the history tree was collected, SCBP did local minimization and a recursive pruning algorithm ensured that only beneficial correlation paths were considered by SCBP code transformation. After local minimization, global analysis determined which basic blocks to copy and then generate a new SUIF output file for downstream SUIF passes to ensure efficient code layout.

Initial support was only for MIPS, with intentions to support HPPA and x86. See ftp://eecs.harvard.edu/pub/hube/halt.tar.gz for source code. In the machsuif download, there were descriptions for Alpha and x86. The machsuif software was written in C++ and used noweb documentation.

4.4 Mtool

Mtool was used to instrument programs for building an initial profile and then identify important regions to instrument with performance probes. Mtool used the internal CPU clocks for profiling which could be read within 10→100 cycles. The aim was to keep overhead of performance probes under 10%. Data collected included: actual time spent in selected loops, procedures and synchronization calls, and basic block counts. "Mtool generates an accurate picture of program performance in just over the time for two executions of the uninstrumented parallel code" Goldberg and Hennessy (1993). This represented an improvement over previous systems that used simulation and tracing to gather performance data, since simulators slowed down program execution by a factor of 100 → 1000 and were therefore impractical to run over full programs. Mtool ran on SGI workstations and MIPS-based processors. There were intentions of extending support to SPARC, IBM RS6000, Intel i860 and to integrate it into an environment that more fully characterized synchronization overhead.

In Horowitz *et al.* (Horowitz *et al.*, 1998, pg 189) several performance monitoring tools were discussed — their interest was "Informing Memory operations", however, for their study Mtool had the following problem:

> Mtool gathers memory statistics for loop nests by comparing basic-block execution times from program runs with estimates of execution times based on ideal memory behavior. In this way, it is able to use techniques based on program-counter sampling to gather program memory statistics. The main drawback to approaches like Mtool is that statistics at a loop or basic-block granularity are often too coarse-grained to be useful in understanding program memory bottlenecks.

4.5 pixie

A detailed description of pixie was given in Smith (1991). The original author was Earl Killian at MIPS Computer Systems, Inc. pixie could trace, profile, or generate dynamic statistics for any program that ran on a MIPS pro-

cessor (MIPS I and MIPS II at time of article). `pixie` annotated executable code to collect dynamic information at runtime. It was fast, worked by partitioning a program into basic blocks, and then ran directly on the machine without emulation or additional system calls. `pixie` options allowed generating branch profile information, instruction traces, data traces, or instruction and data traces. `pixie` only traced basic block entry addresses and load/store data addresses. Note that `pixie` generated user-level address traces for a single task.

`pixstats` analyzed program information obtained with `pixie`. A processor simulator, `xsim`, was written by Michael Smith to work with `pixie` traces Smith (1991).

Several projects at DEC Western Research Laboratory used `pixie` as they adopted MIPS processors for a brief period between the VAX and Alpha. The `pixie` related work at DEC was mainly for comparing "Link-Time code modification" and instrumenting binaries. Some details on `pixie` are available in the comparisons to the DEC work (which is covered in §4.16.1) and Wall (1989, 1992).

The `pixie` trace format is given in section 8.2.

4.6 TATL

Mercury Computer Systems' commercial tool, TATL (Trace Analysis Tool and Library), instrumented multiprocessor real-time code running on Mercury Computer Systems' hardware and MC/OS™ operating system. The tool chain consisted of an event-recording library, global time synchronization server, the event buffer collector, a viewer, and a histogram viewer. There were several brochures and HTML pages on Mercury Computer Systems' website, however, the CGI references are probably not much help. Mercury Computer Systems' home page is www.mc.com. The TATL tool was described in Mercury Computer Systems (2001).

Using TATL required four steps:

- Insert functions from the event-recording library into the application.
- Compile and run the application to create event logs in the local memory of each processor.
- Use a tool to collect the data from the local memory of each processor, merging it into a single file.
- Graphically view and manipulate the merged event log.

To reduce the impact of the library during event trace collection, all initialization and control processing was separated into functions that could be performed at the setup phase of the application.

The global time synchronization server was a small system process that ran on a designated processor in a `RACE` system. The server used the low latency crossbar communication facilities of the `RACE` architecture to synchronize hundreds of processors. This was achieved because the `RACE` architecture kept the communication path open during this remote read request, the single global time stamp was sent back with a latency of only a single clock cycle per crossbar switch in the path through the network. For a 66,66 MHz system, this was only 15 ns per crossbar switch, so even for large networks, the variation in time stamps was usually under a microsecond.

The data formats were published allowing third party interfaces. There were input filters for MATLAB to enable analysis of data that was accurately collected by the TATL tool.

4.7 KernInst

`KernInst` was able to dynamically instrument kernel code and was tested on an unmodified Solaris UltraSPARC kernel. The tool used fine-grained code splicing. Instrumentation points could be almost any kernel machine code instruction. Patching was described with differences for other architectures described — particularly single-instruction splicing on MIPS with a branch delay slot . Be-

cause KernInst instrumented entirely at runtime, gathering kernel information was an interactive process. For the article, refer to Tamches and Miller (1999).

4.8 MPTRACE

MPTRACE Eggers *et al.* (1990), was described as a software tool for shared-memory multiprocessors that was designed to:

- Produce a portable software trace tool for multiprocessors that required no hardware monitoring or modification.
- Reduce runtime dilation well below that of other software techniques and even below that of microcode schemes.
- Produce accurate traces that were as close as possible to what would be obtained with a non-intrusive tracing technique.

MPTRACE was based on inline tracing, a software approach that entails no hardware modification. The technique automatically modified the assembly language version of the compiled target application, inserting code to collect traces. Thus, at execution time, the program traced itself. The number of program points that required instrumenting were reduced by compiler flow analysis. A postprocessor was incorporated into MPTRACE that reconstructed the full trace from the original source and the dynamic trace information.

There were three phases to MPTRACE: A preprocessor that read the assembly version of a source program, inserted tracing code and created data structures for identifying the types of memory references (roadmap). The modified program was linked with the MPTRACE runtime routines and executed. The execution produced an encoded trace file that included only the information that could not be reconstructed at post-generation time, such as data references or control transfers based on runtime data. The postprocessor examined the encoded trace file, along with the roadmap from the preprocessor, and produces the final trace data.

MPTRACE was developed on an i386 based multiprocessor Sequent Symmetry machine. Source code was available.

MPTRACE was compared in Cmelik and Keppel (1993) to pixie, but annotates assembly code instead of object code. It ran on a multiprocessor and collected more trace information. Multiprocessor tracing is harder than uniprocessor tracing because timing changes can change the address trace. Multiprocessor programs typically ran a factor of 10× slower and uniprocessor programs 2× slower.

4.9 TRAPEDS

TRAPEDS (Trace-Producing Execution-Driven Simulation), was described in Stunkel and Fuchs (1989, 1992) and Stunkel's PhD thesis (which was not available for free downloading from the University of Illinois). TRAPEDS was written to study cache performance on a multiprocessor (i386 based).

TRAPEDS traced addresses on multiprocessors by modifying the executable at the assembler level, producing a new executable program that dynamically produced correct address traces of user code. By analyzing the trace as it was generated, the problem of saving huge trace files was eliminated. Large trace files (and a large number of trace files), result in any multiprocessor, as the runs are determined not only by the number of processors in a cluster, but also by the input data sets. A disadvantage of not saving the trace is that it is difficult to repeat experiments or share the data with others.

Similar to MPTRACE, but the analyzer was called incrementally at run time. TRAPEDS ran a factor of 10 $\rightarrow$ 30× slower on an Intel iPSC/2 Hypercube, depending on the number of processors (and excluding analysis time). The Hypercube ranged from 1 $\rightarrow$ 16 i386 processors, and the TRAPEDS study was to examine cache performance for different hyper-

cube sizes. The trace-drive simulation was for a non-shared memory multicomputer.

In the *Conclusions* section of Stunkel and Fuchs (1989), two of the drawbacks of TRAPEDS were; the addresses of user code were virtual addresses, and the operating system could not be traced. In a later article Stunkel and Fuchs (1992), both user and *full* system traces were generated. Although the operating was not traced initially, in the typical multiprocessors at the time, their operating system support was minimal in comparison to workstations, and users were more interested in improving programs that they had source code for, while the operating system source is not freely available (or casually tweaked).

4.10 University of Wisconsin–Madison

There were several prominent researchers from the University of Wisconsin–Madison, so I decided to group them into one large section with separate subsections. The profile, trace and simulation of Ball, Larus, Woods and Hill covered many publications, and were mentioned in several trace patents.

4.10.1 AE — Abstract Execution

`AE`, was described in Larus (1993). Control flow was captured to record only transitions between blocks where a program chooses among alternative paths. `AE` wrote out an identifier in each block that was the target of a conditional branch. After tracing, a trace regeneration routine used this sequence of identifiers in conjunction with a program's control-flow graph, to reproduce the program's entire control-flow trace. `AE` used a byte or halfword to record a block's identifier. `AE` has been incorporated into the GNU `C` compiler. The slow down was $1 \rightarrow 12\times$ the unmodified program's execution.

4.10.2 QPT

`QPT`, by Thomas Ball and James Larus was described in several papers Ball and Larus (1992); Larus (1993); Larus and Ball (1994).

The order of some of the dates is due to how papers are published and referenced. The journal reference would take precedence over a conference proceeding, which would also be referenced before technical reports. For patent applications, the earliest publicly available reference is important. As an example, Larus and Ball (1994) was based on a technical report Larus and Ball (1992). The Larus and Ball (1994) publication was received by the publishers in July 1992, revised in July 1993 and published in February 1994.

In Ball and Larus (1992), the authors wanted a system that could obtain an exact basic block profile or trace. Two algorithms were detailed;

- an algorithm to instrument a program for profiling that chose a placement of counters that was optimised–and frequently optimal–with respect to the expected or measured execution frequency of each basic block and branch in the program;
- an algorithm to instrument a program to obtain a subsequence of the basic block trace–whose length was optimised with respect to program's execution–from which an entire trace could be efficiently regenerated.

Each algorithm was split in two parts. The first chose points in the program where to insert profiling or trace code. The second used the results from the program's execution to derive a complete profile or trace. These algorithms were based on the maximum spanning tree algorithm, and applied to the program's CFG according to the authors. The paper described both profiling and tracing; the profiling was mentioned in section 2.5.

For profiling, the sequence through the basic blocks is not important, whereas for tracing it certainly is. A simple method was to write a unique mark (witness) to a trace file whenever

it executed. The trace file can simply be read to regenerate the execution. A more efficient method was to write a witness only at basic blocks that were targets of predicates, which was done in AE. The authors suggested writing witnesses when edges were traversed and not when vertices executed, and choosing the edges that write witnesses carefully. Single-procedure and multi-procedure tracing were considered. When the AE trace system was upgraded to place witnesses rather than instrument every branch, the number of witnesses was reduced by a factor of up to three times for programs with more complex control-flow graphs. The two tools mentioned were QP and AE.

QPT was a follow-on to AE which also used abstract execution. The targets were MIPS R2000 and SPARC CPUs. QPT runtimes ranged from $1,4 \rightarrow 12,3\times$ slower than non-traced programs, depending on the statistics gathered Pierce *et al.* (1995). The difference between QPT and AE was that QPT instrumented the executable, while AE was part of a C compiler.

The trace regeneration process of QPT was a program object file that was linked into the compiled consumer program, which then read the compacted trace directly from disk Pierce *et al.* (1995).

See Mazières and Smith (1994) for a description of extending QPT to handle multi-tasking for a SPARC processor (as they could use reserved registers for tracing). Future work gave hints at using the extended QPT for kernel tracing.

The QP and QPT work was described in detail in Larus and Ball (1994), which was cited above. Both the MIPS and SPARC versions were discussed as well as problems with compiler conventions, branch nulling, SPARC condition code registers, MIPS delayed branch and lack of compiler optimization level information in the linked binary.

Although the Larus (1993) paper was cited in several patents on prior art, it was an overview of prior work by Larus and related work. It did not detail the use of instrumenting edges rather than branches, or reducing the number of counters for profiling or tracing, and neither did the trace patents, as it would not be possible for some hardware based system to instrument something it had not seen before.

4.10.3 EEL and PP

EEL (Executable Editing Library) was used to build a tool called PP (Path Profiler) that instrumented program executables to record flow sensitive and context sensitive profiles Ammons *et al.* (1997). PP recorded not only instruction frequency, but also fine-grain timing and event count information by accessing the hardware counters in the UltraSPARC processors.

Measurements were taken for SPEC95 benchmarks to isolate hot paths that were found to be concentrated in a small number of routines (1→24), which incurred most cache misses (44→99%) and executed roughly $10\times$ as many paths as cold routines.

PP improved on previous work by the authors on QPT, in that QPT did not label procedure timing by context, causing information to be lost that could not be accurately recovered. Also, the previous tools made naive assumptions when propagating timing information in the presence of recursion, which resulted in further inaccuracy.

Overheads were given as: recording hardware metrics along intraprocedural paths incurred an average overhead of 80%, along the call graph context incurred an average overhead of 60%, and recording path frequency along with call graph context incurred an average overhead of 70%. The overhead of intraprocedural path profiling was given as 32% for the SPEC95 benchmarks.

4.11 IDtrace

IDtrace Pierce *et al.* (1995), an instrumentation tool for Intel architecture Unix platforms was classified as a "late code modification" tool, as it inserted tracing code into an executable. IDtrace could produce a variety of trace types

including profile, memory reference and full execution traces. `IDtrace` could instrument stripped binaries (symbol table not required). For full traces, code size was about 5× larger and ran 10 → 12× slower than the original. Executables for tracing were statically linked and kernel code references were not included in the trace.

The tool was compiler dependent, as it needed to recognize jump table code for disassembly purposes. The first step was to locate and then disassemble the code sections of the executable. During disassembly, the code was split into basic blocks and a relocation table created that stored block locations. Since instructions were inserted into the code, almost all instructions were shifted in memory, so their branch and jump instruction targets were translated to reflect this. For data objects, however, address translation without the symbol table was impossible. It was important that all data locations remain unchanged during instrumentation. In some cases, data in the code segment caused problems for disassembly. Another problem that `IDtrace` encountered was due to variable length instructions in the Intel `x86` ISA.

One of the problems in the Intel `x86` ISA is that there are both short and long target branch instructions, and with instrumentation, some short branches move out of range. There are also approximately 180 instructions that can modify memory in the i486, and many of these can perform both a load and a store. Some `x86` instructions even access strings. The MIPS R3000 has only 14 instructions that modify memory Pierce *et al.* (1995), which can only perform a single write or read, with no instruction able to access more than one memory address.

4.12 Pin

Pin was a software system that performed runtime binary instrumentation of Linux applications. Pin and the Pintools were described in Luk *et al.* (2005). Pin was written for *ease-of-use*, *portability*, *transparency*, *efficiency* and *robustness*. There were several architecture independent Pintools including profilers, cache simulators, trace analyzers, and memory bug checkers. The four architectures that the tools ran on included the Intel `x86`, EMT64, Itanium and Xscale (ARM). Instrumentation was performed by a "just-in-time" compiler. Pin, the application to instrument and Pintool all run in the same address space with each one having its own copy of *glibc* as some of the `C` library functions were not re-entrant. Pin sat above the operating system and could only capture user-level code. *Ptrace* was used as the mechanism for loading Pin into the address space of an application and attaching it to an already running application in the same way as a debugger. It was also possible to detach from the application, leaving it to continue executing the original, uninstrumented code.

Pin's reason for not using "trampoline" binary translation rather than JIT (Just-in-time) compilation was that the `x86` architecture has variable length instructions, so replacing instructions with calls could overwrite the following instructions. Static and dynamic linked programs could be tested, whereas previous tools used static binary instrumentation (ATOM, EEL and others).

Opcodemix and PinPoints were part of the Pintools. Opcodemix could determine the opcode mix per basic block, per routine or per image, and was architecture specific as opcodes are architecture specific. Much of Opcodemix was generic across the four supported architectures.

PinPoints was to automate finding the region to instrument and simulate, as the applications the authors were testing (commercial database with 60 MByte execution image size), would have taken years to simulate the whole application.

In (Luk *et al.*, 2005, table 4), the dynamic instruction count for some large applications were given; the ranges were from 521 billion→25406 billion. The SPEC2000, some computational fluid dynamics, photo-

realistic rendering, and Oracle database applications were tested using Pin and the Pintools. Overheads were given and compared to existing systems. The code was released under GNU licensing conditions.

4.13 Goblin

`Goblin` Pierce *et al.* (1995); Uhlig (1995) was a trace tool for PowerPC to instrument IBM RS/6000 applications by annotating code at the basic block level from object modules with detailed symbol table information. The instrumented objects were reassembled and linked into the new executable by the system's assembler and linker. `Goblin`'s first step was to disassemble the executable into assembly code objects. It then annotated the assembly code, recorded static data about the blocks in the objects, then updated the symbol table to reflect the instrumentation changes in each object. The profile routines were introduced as a profile library to be included in the image at the link stage. Users could select different kinds of traces by linking in different trace libraries, which included complete basic block, full memory reference trace, and an on-the-fly basic block statistics calculator to avoid the problem of storing large traces.

4.14 MemSpy

`MemSpy` was described in Martonosi *et al.* (1993). The authors compared `MemSpy` against other trace sampling tools and generated results for 16K-, 128K- and 1Meg caches. The tool sampled sections of the trace. The speedup when sampling one tenth of the code was between $4 \rightarrow 6\times$ the fully sampled version, causing slow downs in the range of $10 \rightarrow 40\times$ the uninstrumented code.

`MemSpy` was a performance debugging tool designed to locate memory bottlenecks and cache problems. The high-level information presented breakdowns of how much time was spent within each procedure for memory stalls due to cache misses and in computation. The code was instrumented at the original assembly code level with a procedure call to a memory simulator. At runtime, `MemSpy` performed the following actions:

- Save the application registers so that the memory simulator did not destroy them,
- Used a cache simulator to determine whether the reference was a cache hit or a miss,
- Updated `MemSpy` statistics for the relevant code and data objects,
- Restore application registers to their original values, and
- Return control to the application.

The following cycle counts were given: 20 cycles if the reference was a cache hit, roughly 200 cycles for a cache miss, and 60 cycles to save and restore registers. These overheads contributed to the non-linear speedup for reduced sampling.

Errors were reasonable and enabled programmers to locate problems quickly. A useful table was given of when to use `MemSpy`, which applications were amenable to sampling, and where to use short–, medium– or long samples.

4.15 Tapeworm II

`Tapeworm II`, a *trap driven simulator*, was described in detail in Uhlig's PhD Uhlig (1995) and Uhlig *et al.* (1994). `Tapeworm` was a software based simulation tool that resided in the operating system kernel to trap TLB and other kernel traps to evaluate the cache and TLB performance of multiple task and operating system intensive workloads. The work was done on a MIPS-based DEC workstations (R2000, R3000 CPUs in DEC 5000/200 and DEC 5000/240), a Gateway `i486` PC, and in the *Future Work* section, was being ported to SPARC and DEC Alpha based workstations. The work on the R2000 based processor was compared to results obtained from *Monster*,

which is a logic analyzer based instrumentation system (see § 6.8.5). Other comparisons used `pixie`, which is an address driven trace simulator (see § 4.5).

Trap driven simulation differs from the more common trace driven simulation in that traps are set in all memory locations in a workload's address space. When the workload executes, memory locations not currently in the cache cause traps that represent simulated cache misses, which are directed to the `Tapeworm` simulator. The traps are simply counted and cleared; there was no need to search a data structure representing a simulated cache. A major advantage of this method over trace driven simulation is that the workload does not require instrumentation or rewriting because traps are dynamically set and cleared while the application runs. Another advantage is that `Tapework`'s kernel privileges allow traps to be set on user-level and kernel-level code.

`Tapeworm` required modifications to the operating system virtual memory code, however, any study at this level would require source code anyway. Two attributes, *simulate* and *inherit*, allowed selective simulation of a single program (and the operating system interaction), or every program started by a shell (after a task fork).

Repeated runs of trace driven simulation generate similar traces on a workstation, whereas precise trap driven simulation sequences are impossible to reproduce from run to run due to dynamic system effects. Trap driven simulation is restricted to simulation of memory hierarchies and their components, while trace driven simulation is easily and efficiently extended to simulate other architectural features such as instruction pipelines.

`Tapeworm II`'s performance figures in (Uhlig *et al.*, 1994, Figure 2, pg 137) are significantly better than trace driven simulation, with `Tapeworm` slowdowns decreasing to nearly zero for larger caches. On the smallest cache size (1K), the Cache2000 slowdown was 30.2× while `Tapeworm`'s was 6.27×. The Cache2000 slowdowns were never below 20×.

Although the `Tapeworm` measurements were done on workstations, embedded targets could use similar concepts for measuring code coverage without having to instrument the workload. The TLB would need to be setup, but a simple mapping with all memory available in physical memory would simplify the additional software instrumentation. Many embedded processors have simple TLBs and variable page sizes, so for course measurements, a 4K mapping could be used.

4.16 Other Code Modification Trace tools

See Pierce *et al.* (1995) for a good overview of trace and profile tools, and for statistics of slowdowns and code dilation. The code modification tools in this section were summarized from the DEC technical reports. Although they can be used for profiling and tracing, their intended use was architectural exploration (mainly cache behavior), and checking compiler optimizations. `Mahler`, `epoxie` and `nixie` worked on individual programs and not the kernel. Their inclusion in this survey was to highlight the effort involved in binary instrumentation, the problems of delayed branches and the compiler knowledge required to modify a binary image. Faster machines for cross development (since 1989), lack of large commercial workloads like databases for embedded targets, and shorter market windows for embedded products does not justify the binary instrumentation investment for teams using more than one embedded target architecture. The effort will more than likely be better spent instrumenting source code for a hybrid trace monitoring or profiling system over limited sections of code.

4.16.1 Mahler

`Mahler`, a back-end code generator and linker for the experimental Titan workstation build at DEC Western Research Laboratory, was compared to `pixie` in Wall (1992). Possible in-

strumentation was source-level instrumentation compatible with `gprof`, instruction-level instrumentation like basic block counting, and address tracing. The linker performed global register allocation and other optimizations. At the link stage, there is relocation information, so instrumentation was easier than the case for `pixie` which operated on binaries that might not even have a symbol table. There were 64 registers in the Titan, so allocating some for instrumentation was less critical than the three registers that `pixie` must "steal" from the 32 available in the MIPS architecture. (There are not really 32 available, as R0 is always wired to 0, R31 is used for subroutine returns, etc.). `Mahler` instrumentation was complicated by the delayed branch in the Titan architecture.

Other uses for `Mahler` were described in Wall (1989). Pipeline instruction scheduling, interprocedural register allocation, instruction-level instrumentation for performance analysis of the code modifications rather than performance enhancement. `Mahler` was part of the code modification tools developed at DEC — some were developed for a specific purpose and then discarded, others were used in production systems (it appears at the research laboratories rather than customers), and also shipped to customers for the Alpha DCPI tools.

4.16.2 nixie

`nixie` was described in Wall (1992) as an attempt to reduce the overhead of `pixie` for MIPS based DEC workstations. Various assumptions were made about the MIPS compiler output, which reduced the executable code size and runtime of the executable compared to `pixie`. `nixie` was described as a prototype and possibly due to the short lifespan of DEC's MIPS workstation product line, never really gained widespread use. There appeared to be about a 30% advantage from the tables given in Wall (1992), however, most instrumentation work on early MIPS machines (pre 64-bit) used `pixie`.

4.16.3 epoxie

`epoxie` was briefly described in Wall (1992) as an experimental prototype for late code modification on MIPS workstations. The authors wanted a tool similar to `pixie`, but to include the linker information similar to `Mahler` without modifying the linker. The figures in the tables were nearly all identical to `nixie` in the comparison between `Mahler`, `nixie`, `epoxie` and `pixie`.

4.17 Embedded Instrumentation Using Instruction Modification

Many of the tools in this chapter modified the binary or assembler for instrumentation. Embedded work changes between architectures too often to develop or purchase tools that can instrument binaries selectively. Sometimes there is a single source tree but multiple build directories — Linux is typical of one source tree for multiple architectures, with the compiler taking care of the processor derivatives passed in the top-level *Makefile*. There are several ideas we would like to test on different devices, and even multi-core devices, so source code instrumentation would be most suitable.

4.18 Replay

Many trace studies were conducted to develop program debuggers that worked by "replaying" the program from its trace. In Netzer (1993), a shared-memory multiprocessor was traced for replay, as repeated runs on non-deterministic hardware produces different results. The paper title on optimal tracing was of interest to any embedded work, as tracing infrastructure is more of an issue on small memory constrained embedded targets than on a workstation. The amount of shared-memory references that were traced were in the region of 0.01 $\rightarrow$ 2%, which the author claimed was

$2 \rightarrow 4$ orders of magnitude reduction over previous techniques that traced every access.

Netzer was interested in detecting races by checking each shared access at run-time and if a race was found, then it was traced. For an embedded system, the repeated runs would also give different results, but the run-time overhead is too high. When the paper was written, the following remark was made Netzer (1993); "Adaptive tracing strategies are not only becoming practical for the first time but are becoming necessary. In the past, the cost of writing to disk was insignificant compared to the cost of even a small amount of computation. Processors are now so fast that a significant amount of work can be performed in the time it takes to write only a single trace record." The situation has become worse, with a significant number of cycles being spent just to access L2 cache and more for main memory.

4.19 Summary

Software modification generates trace data without altering the hardware. Several hardware related trace patents referenced James Larus' work Larus (1993).

The articles referenced in this chapter generally described gathering traces, not how they were processed. The authors often referred to "user supplied" trace analyzers.

The software tracing (other than CodeTEST) was run on workstations where the programs could be repeated and only relied on memory contents. Embedded systems rely on sensor readings in many comparison tests, making it difficult to repeat the paths through a program without simulating the external environment as well. The effort involved for simulating the external environment for industrial sized problems is probably more than using hardware tools (logic analyzers etc.). This effort and the lack of incentives for industrial publishing contribute to few tools available for software tracing embedded systems.

Aspect programming determines what to compile into the running application and is typically used to instrument workstation applications. `C` programmers are well acquainted with the *#ifdef — #endif* blocks throughout programs. These might be used for different embedded architectural targets in an effort to use a single source code tree, or enabling different levels of debugging (or setting bits in a 32-bit debug variable). When developing new code that will be using hardware instrumentation for tracing, it will be worth the effort of placing the conditional compilation blocks into the code and using it for unit testing. Instead of having to track an instrumentation database of unique *tags* as per CodeTEST, a "leaf function" can be inserted at each point of interest that is an assembler routine that writes out the return address (link register or equivalent) to a fixed address in an attached FPGA. The FPGA can automatically add a time stamp and transmit the data to a host that compares the address (program counter of the return address of the leaf function and not the call address of the leaf function), to the symbol table from the statically linked ELF (Executable and Linkable Format) program.

The hardware assistance will reduce the amount of perturbation compared to a purely software based trace, however, the overhead is significantly higher than tracing out of a trace port. Hardware based tracing and assistance is covered in more detail in Chapter 6.

In ATOM, TRAPEDS and other trace systems that link in user supplied analysis functions, the traces are not stored. These systems tried to overcome the problem of saving huge trace files. The level of tracing for branch prediction, cache studies, and possibly multi-threading are far higher than trying to view or identify a RTOS. Embedded targets do not generally have large disks for storing huge trace files, however, increasing numbers of embedded targets have high speed Ethernet and commodity hosts with hundreds of gigabytes of storage per disk are available at almost a tenth the price of the machines that were used in the original trace studies (DEC Alpha, SGI MIPS workstations with a few gigabytes of

disk space).

The authors opt for simulation to fill in the gaps between basic blocks, but in Chapter 3 where we cover simulation, new processors have made this impractical — even in embedded space. To study the cache, the only alternative to a highly instrumented testbed is simulation (or a trace port on newer devices that sits between the CPU and the cache). The studies in this chapter were not for real-time systems, but many of the instrumentation techniques can be modified for high-end embedded targets.

The slow downs with simulation can easily be several thousand times. In the letter to the NSF for the RAMP project, the average time to simulate the SPEC2000 benchmarks on a 2005 workstation was 5 days, with the longest taking 24 days (Arvind *et al.*, 2005, pg 19).

For embedded work, instrumenting changes addresses as code is added or removed. Typical embedded targets like PowerPC and MIPS have 32 registers, so instrumentation where some registers are borrowed is not as serious as for ARM or the AVR32 which have half the number of available registers. Delayed branches or nulled branches make binary instrumentation tricky, and the effort in developing tools for constrained targets (which supposedly have small code bases to execute), might be greater than instrumenting the source with automated tools. Initially RISC processors had fixed instruction sizes, however, both MIPS and ARM have 16-bit variants of the 32-bit instructions for code density. The AVR32 was launched as a variable length instruction set device, and more recently, Freescale offered variable length instructions for PowerPC derivatives aimed at the automotive market. Note that libraries will also need to be instrumented for global tracing or profiling, as well as the kernel. Both these code bases are not easily instrumented with automatic tools (either binary or source instrumentation).

Chapter 5

Tracing via Microcode Modification

5.1 Introduction to Tracing via Microcode Modification

Systems like ATUM can only run on processors that have access to microcode which can be modified. Modern RISC and VLIW processors do not support user modified microcode (instructions are hard-wired). Even the 68000 CPU with microcode did not allow user modification. The closest tracing to modifying microcode was the T-bit method, where a software trap was generated after every instruction when a trace bit was set in one of the privileged registers (usually status register), however, according to (Sites and Agarwal, 1988, pg 187), the T-bit method slow down was much larger than the modified microcode method.

5.2 ATUM (Address Tracing Using Microcode)

ATUM was described in Agarwal *et al.* (1986). ATUM generated a full address trace of all instructions executed by modifying VAX 8200 microcode. By enabling or disabling tracing, modified microcode wrote trace data into a reserved memory buffer. When the buffer was full, tracing was suspended, the buffer written to secondary storage and tracing resumed. To compensate for the slow down, the clock interrupt rate was lowered. ATUM was used to provide detailed cache analysis data. Tracing the operating system was no more difficult than tracing user processes, and could trace any operating system that ran on the machine. ATUM was also able to trace dynamically linked code and interrupt handlers.

Shortcomings of ATUM were only microcode controlled activities could be traced, so I/O and hardware generated traffic could not be observed. The trace length was limited by the reserved main memory, and obviously by the amount of secondary storage. Note that all trace systems will be limited by secondary storage or high speed trace buffers between dumping the trace. Two megabytes of reserved trace main memory stored about 400 000 references. No attempt was made to "freeze" the machine when the trace buffer filled, but a simple Pascal program was written that could start the trace, dump it to disk, and re-enable the trace (the trace was automatically disabled once the trace buffer was full).

Workloads were studied for context switching, TLB– and cache performance, plus tuning for Ultrix and VMS over a wide range of workloads and multi-programming levels. Each trace was roughly half a second for a machine rated as a VAX-11/780. Four to eight traces were taken per application.

In reviewing their results, the effects of systems references degraded the cache performance considerably for all their benchmarks and cache organizations. Often the miss rate increased by a factor of two when considering operating system behavior. Other areas where ATUM was used was to debug some operating system code, examine unusually slow programs where no other tools could give any insight, and examine long interrupt latency problems. The authors noted that using ATUM for ordinary users was an overkill. The slow down given by the original authors was 10×.

In Cmelik and Keppel (1993), ATUM microcode simulated a cache to perform trace compression. It could run 20× slower excluding I/O. In Pierce *et al.* (1995) the slow down was reported as 10×.

A multiprocessor version of ATUM, called ATUM-2 gathered traces for a quad-processor VAX 8350 Sites and Agarwal (1988). Similar to the initial ATUM, while traces were dumped to disk, the references issued by the processors were lost. The slow down was reported to be 20×. Three operating systems were traced: VMS, Ultrix and Mach.

ATUM-2 traces differ from the basic block address traces in that every instruction was traced. Each trace record was five bytes long: a type byte followed by a four byte entry. The type byte included the processor number, the type of the following entry, such as an instruction address, data read or write with the length of the data field accessed, a process identifier, and several others. Each trace consisted of a number of interleaved reference streams for up to four processors. The TLB miss was also recorded, which allowed post processors to obtain the physical address from the virtual addresses that were generated in the trace. Cache sizes from 4KB to 1024KB were simulated as well as studies of semaphores. Several issues between single processor and multiprocessor tracing were discussed in the appendix. The reason why the microcode could be modified was that the VAX 8350 was designed to be field upgraded without shipping new microcode ROM. The trace microcode modifications were installed at bootup on each processor in the limited microcode patch RAM space.

5.3 Summary

Commercial processors do not allow easy access to microcode. Generally, for speed, modern RISC processor instructions are hardwired. With the soft cores available in source form, it may be possible to use some of the ideas in this chapter to instrument the ALU for counting various instructions. FPGAs continue to improve in density and speed while lowering prices. In-depth studies may well justify custom FPGA development for counting individual opcode execution rates; not necessarily including the processor, but intercepting the instruction stream.

For embedded processors, microcode tracing is impossible, however, the limitations and problems in tracing systems in general are well documented in the ATUM work in Agarwal *et al.* (1986). For real-time work, context switches and interrupt execution traces are more important than cache or TLB performance work, and ATUM was possibly the only non-hardware modification system able to trace at this level.

End users are unlikely to want to modify microcode as this assumes such intimate knowledge of a machine and would invalidate any warranty.

Chapter 6

Hardware-Based Tracing

6.1 Introduction to Hardware-Based Tracing

Hardware monitoring has the lowest system perturbation, however, it needs to be designed into a system rather than retrofitted afterward. Instrumentation with logic analyzers is becoming less practical as device speeds increase, packages change from sockets to BGA (Ball grid array) or soldered down components, and loading due to probes is critical for a 100 MHz+ bus.

Hardware based tracing was typically logic analyzer based when processors were socketed, or if modifications were made to the processor circuit board for modern μP packages (TQFP (Thin Quad Flat Pack), BGA). Modifying a workstation's motherboard is not undertaken by casual end-users, as this requires the original circuit diagrams and invalidates any warranty. End-users are more likely to modify embedded targets, however, these are not standardized and repeating any other research group's experiments is difficult.

Built-in instrumentation is becoming necessary with reduced visibility as SoC designs absorb processors, peripherals, cache and memory. The multi-core and SMT devices have even higher requirements for hardware assisted debug. Even low-cost ARM devices include the ETM.

Instrument– and semiconductor vendors addressed these requirements by integrating logic analyzers, trace generators and SERDES for high speed access via a small number of pins. To extract high speed data over serial interfaces requires higher speed than the previous parallel interfaces to maintain the bandwidth requirements — rather obvious but patented. There are several patents that appeared after devices were produced which seem questionable. Besides, there are plenty of companies using serial interfaces to on-board debug agents, so the patent process seems to be ignored. See chapter 7 for examples of trace patents.

FPGA capabilities and prices are at levels enabling permanent instrumentation. Several soft-cores are available, increasingly with trace ports that use the built-in BlockRAM. Logic analyzer cores are also fairly common, so perhaps by using this IP one can avoid patent issues. Sticking to standards like the Nexus 5001 Forum should also help.

Articles related to hardware tracing and debug are mostly written by silicon vendors, EDA vendors and IP third parties. The novelty for academia has faded as the sizes of teams to produce chips get larger and costs escalate sharply. Patents and also a problem with very

broad titles covering areas that are prior art and common knowledge. The trends are towards standards to minimize the patent impact on suppliers and tool writers. If the EDA market becomes more fragmented — there are already so many interest groups with a small number of members — then end-user acceptance becomes more difficult. Nexus also had several members abandoning their positions in the association.

An excellent book on hardware instrumentation is Tsai *et al.* (1996). The book was published in 1996 and would be excellent defense against more recent patents. Although the work was prior to 1996, many current ideas are simply borrowing older ideas and taking advantage of better levels of silicon integration or price structures. Another trend that was not evident in 1996 was that commodity PCs would become so powerful (graphics and raw power) that having a cluster for parallel simulation or large format screens for visualization would affect how hardware tracing would progress. It is likely that simulation will become more popular along with visualization, particularly as gathering hardware traces becomes more difficult.

Patents for taking data off a microprocessor development system Murphy *et al.* (2003), and then advocating open standards seems contradictory, but a patent on an ICE combined with the above, after years of systems from Hewlett-Packard and Intel adds to the puzzle. Intel were early producers of development systems and ICE, besides who was it that produced the "bond-out" processors for ICE manufacturers? Two Intel ICE development patents are Poret and Mckinley (1987); Groves (1996).

The layout is slightly different to prior chapters. Where possible, sections have been grouped by processor, however, test equipment vendors usually support multiple processor architectures. I also placed the logic analyzer sections at the beginning, followed by trace analyzer vendors, and then the more general articles. Patents have been included in this chapter where appropriate.

6.2 Logic Analyzers

Capabilities vary widely; the high-end is usually associated with Agilent (previously Hewlett-Packard's Test and Measurement equipment division), Tektronix and Yokogawa. Other vendors have lower cost offerings, often as part of a microprocessor development system or ICE. A sampling of these vendors are American Arium, Ashling, Noral, iSystems, dli, Corelis, Lauterbach, Embedded Performance Inc., and CARDtools.

A little history may seem out of place in a survey, however, what was standard practice for many years and instruments that were launched prior to patents that were issued to others is a warning to end-users who wish to use any non-standard trace instrumentation. Broadly defined tracing patents are covered in Chapter 7, while a few relating to logic analyzers are hinted at in there. The *single-chip logic analyzer* or *on-chip logic analyzer* patentees are in for a tough time!

As integration progressed from SSI to LSI to VLSI, it was obvious that systems would move to board-level, board-level products would move to single-chip, and the single-chip would keep integrating more devices at lower cost (SSI (small scale integration) $\rightarrow$ MSI (medium scale integration) $\rightarrow$ LSI (large scale integration) $\rightarrow$ VLSI (very large scale integration)). It also made sense to move logic analyzers into the SoC designs when there was no other alternative — there were no longer any external pins to probe that were attached to the core, due to on-chip memory.

The move to on-chip logic analyzers must also consider processor history. The workstation manufacturers developed their own heavily instrumented testbeds with the best logic analyzers. The embedded devices could use logic analyzers and for a while, "bond-out" devices were specially made for ICE manufacturers. Some form of probing or built-in logic analyzer was part of any decent ICE. Motorola's 680x0 and Intel's x86 family were still available in pin-grid-array packages with adapters for

logic analyzers. As late as 1994, HP (Hewlett Packard) was still advertising "Software Analyzers" that interfaced to a logic analyzer to correlate the bus trace to source code Hewlett Packard (1994). After the failure of the Apple Newton PDA (Personal Digital Assistant), ARM and their semiconductor partner, VLSI Technology Inc., found themselves in the IP business. The small ARM core found its way into ASIC devices and soon visibility disappeared. It was some time before catalog ARM devices were available but the debugging became a big enough issue to prompt HP and ARM to announce joint trace projects Steffora (1999).

In 1999 and 2000, there were several announcements and products available for HP logic analyzers for tracing embedded cores. The adverts mentioned low-cost trace that every engineer could afford to use; support for different vendors' tool chains and obviously methods of extracting the data from the core. This work followed IBM's work on trace units IBM (1991); IBM Microelectronics (1998, 1999) and Motorola's CPU32 BDM (Background Debug Mode) port. Around the same time as the ARM joint development, the IEEE-5001 Nexus standard work was being drafted with several semiconductor and tools vendors (1999). It is indeed a bit of a surprise that several patents were accepted as minor modifications of this work. For instance if there are two cores in one chip, then there will be two trace ports. Someone might decide to share the trace port to only debug one of the cores, as in patent Byrne and Holm (2006). The extraction of the data from the cores is over a set of multiplexed pins at fairly high data rates — it is obvious that a FIFO is required between two separately clocked units (the target and host systems), however, this appears in several patents. To reduce the number of pins and speed up the transfer, one can use differential signaling and increase the data rate over narrow channels. This is what SERDES devices have been doing for years and the whole communications industry is based on transporting data over serial cables — often using differential signaling. However, a patent officer sees fit to pass differential signalling on as one of the claims in a patent Murphy *et al.* (2003). In section 6.2.1, similar ideas were applied in the ATC and ATC2 embedded logic analyzers in Xilinx FPGAs.

Disadvantages of standard logic analyzers are briefly

- If trace information is compressed, output on both edges of a clock, or encoded to indicate data or instructions, then a package is required on the logic analyzer.
- Limited memory size which seldom captures more than a second of processor activity.
- Any memory or hard disk storage on a logic analyzer will be much more than the equivalent on a commodity PC.
- Trace formats are not often published, and it is not always possible to export the trace in some neutral format to read into other user-generated tools.
- Stitching files together or merging multiple time stamped records is not typical of a logic analyzer, as there is usually only one logic analyzer per complete system versus one per processor in a distributed system[1].
- Costs are disproportionately high compared to the equipment being monitored, and are difficult to justify for short-term projects.
- Connections to boards are not standard — even touch probes from Agilent are different to those from Tektronix.
- Long cables are difficult to drive from low power devices which already have low fanout capability, and equipment generally is on a shelf of a bench some way from the target. High speed requires drivers at the pod close to the signals being driven, and the pod connection to the target is delicate while supporting the

[1] See work from IBM on embedding dedicated capture logic (logic analyzer) per card on a distributed system and merging time stamps in Chen *et al.* (1997)

weight of active drivers, variable threshold comparators, and their own power supply. Boards being tested need to lie flat to prevent the pod "hanging" unsupported off a connector.

Equipment manufacturers do not seem to be attracted to low-cost testers for slow embedded targets. Very high speeds are unnecessary for tracing processor addresses, which need only be as fast as the processor's access. If the address is latched using deskew facilities in new FPGAs, the speed can be the same or slower than the speed of the RAM connected to the processor, and perhaps widened to twice the line width being monitored to effectively halve the required bandwidth. At some stage, any trace system will overflow no matter how much memory is available. The data may be taken off the trace port at very high speeds but ultimately, when stored to a non-volatile medium, the mismatch in speed will cause a slow down.

6.2.1 Agilent

Agilent has a long history of making logic analyzers and digital logic debuggers. See Agilent Technologies (1973) for what was probably Hewlett-Packard's first logic analyzer, the 5000 A Logic Analyzer. Several application notes in the *167 series* dated January, 1975 described the 1600A and 1607A logic analyzers to verify Intel's 4004, 4040, 8008 and 8080, Motorola's 6800, and Fairchild's F8. There used to be a *Test and Measurement* website hosted by Hewlett-Packard, but the articles are now difficult to find even using their search facility. Almost ten years ago, several tutorials and presentations on SoC and OCD (On-Chip Debug) started to highlight the problems faced by leading consumer vendors. In Marshall (1999), the point was made that tools cannot operate as single point solutions, but need to be integrated so that measurements across various buses and clock domains can be correlated. In spite of OCD over JTAG, complex processors like the MPC 8260 PowerQUICC II can have several buses active simultaneously (local/PCI bus and the PowerPC 603 bus). The many integrated communications devices would also need to be correlated to the software trace.

Several companies partnered with Agilent to embed cores into ASICs or FPGAs to leverage existing infrastructure. An excellent tutorial on the ARM debug in an ASIC with embedded ICE and logic analyzer was given in Marshall (2000).

Although I could not find specific patents relating to the joint development or the ARM ETM, there were several products announced from the mid 1990s relating to testing processors disappearing into ASICs, as well as tutorials of SoC development using ARM as an example. A patent that could possibly cover the joint work (and certainly wider as well) is Yano and Miyamori (1999). There are plenty of patents on logic analyzers, trace ports from ARM, MIPS and IBM and trace methods going back to Couleur *et al.* (1969). The Agilent (Hewlett-Packard) logic analyzer patents cover methods of displaying waveforms, manipulating signals and groups, trigger methods using graphic or text inputs, and how to trigger on serial protocols. For interested users, the easiest search is www.google.com/patents and enter *Hewlett-Packard* into the *Assignee* field and *Logic Analyzer* to match any words. These patents generally show earlier ones they reference, as well as which later patents referenced the one that is currently being viewed. Some US Patents awarded to HP/Agilent for logic analyzers are 4 040 025, 4 455 624, 6 707 474 and 6 624 830.

In Rosqvist (2001), the Aptix System Explorer MP4 was described. Aptix, Agilent and engineers from IBM in Austin worked together to develop methods of probing individual nodes in multi-FPGA emulators. Being able to easily label waveforms, select a node without recompiling a design, displaying waveforms correctly labeled after demultiplexing, and configuration from an attached workstation were early tools for debugging SoC designs. The Agilent Tool Developer Kit was used by Aptix to run an application in the 16465B or

16702B logic analyzers that decoded the data in real-time and presented the results on the logic analyzer's waveform window. The Aptix system was a field programmable circuit board that accepted FPGA modules, standard components, and user connectors. The programmable interconnect was based on SRAM (static RAM) technology supporting up to 936 bidirectional ports. The paper was promoting emulation versus event-based simulators or cycle-accurate simulators, which have slow downs in comparison to hardware of up to 10 million times. The emulation speeds obtained were usually 4 MHz and sometimes over 10 MHz.

Agilent's website had discontinuation notices for the E5904B Option 500 FPGA trace port analyzer in December, 2006 (when accessed). The B4655A FPGA dynamic probe launched in 2004 which was compatible with the 16900–, 1680– and 1690 Series logic analyzers. The Xilinx FPGAs would require the ACT2 core. Xilinx sells the core and license as the ChipScope Pro.

An early article on an embedded logic analyzer, Davis (2000), described joint work by Xilinx and Agilent, which was compatible with the 16700 series logic analyzer writing waveforms in VCD (Value Change Dump) or FSDB (Fast Signal Database) formats. For a general article describing combined oscilloscope and logic analyzer, see Frieden (2005), who also described the FPGA Dynamic Probe. Another view of the combination of ChipScope Pro and the FPGA Dynamic Probe was described by Xilinx in Hansen and Przybus (2005). There were several FPGA Dynamic Probe and Xilinx ChipScope Pro articles in 2004 when Agilent introduced the Dynamic Probe. For references, see Hansen (2004); Olsen (2004); Woodward (2004).

As processors become faster, instrumentation in a probe or on-board will become the norm. See Corcoran and Poulton (2007) for an example of a 40 Gsamples/s 8-bit analog-to-digital converter with 50 million transistors on a single chip as a front-end for an oscilloscope with a million sample buffer. In Business Wire (2008*b*), an embedded oscilloscope was announced! It seems logical that a FPGA able to clock inputs at over 500 MHz with large on-board memory and high speed SERDES ports will end up on-board or in probes as the front-end to a logic analyzer.

6.2.2 Tektronix and FS^2

Tektronix is another logic analyzer–, oscilloscope– and test equipment manufacturer that most embedded developers have heard of. Tektronix lagged behind Agilent in FPGA support for Xilinx, however, in 2006 both FS^2 and Tektronix announced joint support for Xilinx–, Altera– and Actel FPGAs. Altera's NIOS processor has trace support from FS^2 and Tektronix. In Tektronics, Inc. (2006), FS^2 and Tektronix announced support for tracing Xilinx's MicroBlaze™ device.

Tektronix has several application notes on their website for PowerPC–, serial channel– and FPGA debug. They also have trace options for various processors using the logic analyzers rather than trace port analyzers. Hints are given when to use external analyzers or embedded logic analyzer cores and the trade-offs involved Tektronics, Inc. (2002).

From FS^2 and Tektronix (2006), the FPGAView software package was installed onto a Tektronix logic analyzer. FPGAView included a utility to automatically generate a block of logic that was configured in the FPGA — the *Logic Analyzer Core*, which handled the signal definitions and external connects mapped to the logic analyzer pins. The software allowed for a large set of internal signals to be mapped to a small number of output pins via a user-configurable multiplexer. With the Tektronix logic analyzer connected to the FPGA pins and a standard JTAG port, FPGAView software controlled which internal FPGA signals were mapped to the output pins, enabling real-time display and debug. The software also eliminated the need to manually enter and map signal names into the logic analyzer.

6.3 Embedded Logic Analyzer Vendors

The need for embedded logic analyzer cores was identified some time back. More capable FPGAs have enabled internal signal state capture with relatively little effort. The larger EDA vendors also provide logic analyzer cores, e.g. Synplicity's *Identify RTL Debugger*. The FPGA vendors each developed their own versions as well.

6.3.1 First Silicon Systems (FS^2)

Founded in 1998, FS^2 provide VHDL or Verilog for SoC designs and to silicon vendors to instrument their cores. The amount of data depends on the functionality required and is scalable. A white paper from FS^2 Leatherman (2000) gave a brief description of their OCI™ (On-Chip Instrumentation). They were chosen by MIPS for the trace capability in the 4KE cores First Silicon Solutions (2002). MIPS announced two trace related documents MIPS Technologies Inc. (2002*a*,*b*). MIPS acquired FS^2 in 2005.

FS^2 worked with QuickLogic to instrument the QuickMIPS SoC which included the processor, peripherals and a FPGA. The hardware was described in Heape and Stollon (2004). Other details were available in Leatherman (2000), which predates some on-chip instrumentation patents.

Actel and FS^2 jointly developed the Logic Navigator™ and JTAG based instrumentation for Actel FPGAs. A PC screen displayed the waveforms as per typical logic analyzer, without requiring a logic analyzer. The capture depth was obviously limited by the amount of onboard memory that could be dedicated to instrumentation. External memory in the instrumentation pods could be used together with programmable multiplexors — a feature that allowed interactive probing without resynthesizing and reprogramming the FPGA.

In an email from FS^2 Swartley (2003), the trace option takes 30 → 50 % of a MIPS die. That was the reason that most of their licensees for the OCI™ used the trace core in FPGA versions of the 4KE MIPS devices (the MIPS contact said most of their licensees did not want to commit that much silicon to production devices). The only catalog MIPS devices that were found with trace ports were from Toshiba — TMS390x and TMS4955. Although the PDtrace was defined several years ago, it does not seem popular in catalog MIPS devices. With the introduction of low-cost μControllers from Microchip with a trace port, hopefully affordable trace port analyzers appear for MIPS.

On the MIPS website, 15th April, 2008, a hot spot analyzer for fast Linux kernel profiling was announced. Almost a year earlier, Design & Reuse (2007), basic profiling was announced — "System code can be quickly profiled using Zero Overhead PC Sampling. System Navigator tools are able to sample the contents of the PC register without affecting the real-time performance of the target system, allowing software engineers to quickly locate code hotspots. Profiling results can be viewed at the module, function or source line level." There are several patents on sampling "on-the-fly", however, MIPS was granted Thekkath *et al.* (2007) to cover the above features.

6.3.2 ClearBlue

ClearBlue was "design for debug" IP for post-silicon debug. Dafca Inc., launched in 2003, received $15 million venture capital and in 2004, was awarded a $1,9 million Advanced Technology Program (ATP) funding from the NIST (National Institute of Standards and Technology).

ClearBlue used reconfigurable logic to insert instrumentation into IC (Integrated Circuit)s for post-silicon debug. The process of inserting the instrumentation was automated, the tool employed a "patented" distributed configurable infrastructure fabric.

SoC designers could incorporate ClearBlue's

reconfigurable fabrics during RTL development, enabling powerful at-speed observation, diagnosis and debug that accelerated validation and system-level bring-up with early silicon engineering samples — according to the press release on Embedded.com in 2006.

ClearBlue debuggers included signal trace, on-chip logic analyzers and performance monitoring that all fed into graphical debug formats like FSDB and VCD. Communications to the ReDI enabled SoC was via JTAG.

6.4 Hardware Based Trace Ports

There were several patents awarded to IBM for tracing and built-in trace ports. These following subsections briefly cover individual processor trace ports, and have not been grouped by processor or vendor.

6.4.1 Xbox 360 Tracing

The triple 64-bit PowerPCs in the game console have extensive debug features. Brown (2005) described the profile and trace features:

> The Xbox 360 CPU has an array of testing and debug features. Console games are held to rigid quality standards and harsh deadlines, making good support for debugging a significant benefit. The Xbox 360 CPU allows full-speed operation while tracing execution and running tests; this helps to maximize defect coverage, including marginally slow circuits. The analog PHY is likewise covered by a built-in self test (BIST). Over a thousand internal signals can be traced, running at full speed. Robust local and global triggers and pattern-matching capabilities are available. Additionally, the CPU had an external debug bus for extended traces; which ran at 1/4 full speed for the CPU, but let the FSB (Front-side bus) run at full speed.
>
> The trace function was controlled through the JTAG interface and allowed logic signals to be sampled and stored within the on-board trace arrays. Each of the processor cores and the L2 cache had two personal trace arrays and controller. The FSB had one trace array and controller. The rest of the chip units shared a single trace array and controller. Each controller could be set up to trace 64 logic signals and to act as a triggered logic analyzer allowing great flexibility in the timing and coverage of events. The on-chip trace buffers allowed 1024 samples. If the trace required more space than the on-chip arrays supported, the external debug bus could be set up to capture the data externally. Because the CPU ran so fast, an individual sample is limited in number of logic signals captured. Tracing could be turned on and off at any time. However, the trace buffers were initialized at the start of each trace event so only the most recent trace event data was available.

Unfortunately there are no publically available development kits for the Xbox 360. The trace details were not released either, but judging by the licence fees games developers must pay Microsoft, purchasing trace units will not be a problem to meet tight deadlines.

6.4.2 Real-time Instruction Trace in PowerPC 40x

IBM Microelectronics provided trace hardware inside the PowerPC 4xx family of embedded CPUs and means for customer access IBM Microelectronics (1998, 1999). The additional trace hardware provided full speed program trace and allowed program flow reconstruction

even with cache enabled.

Eight pins were used to determine the following states:

- Instruction did/did not execute,
- Branch taken/not taken,
- Address broadcast bit (three address bits then take ten cycles to broadcast 30-bit address — 32-bit instructions so two least significant address bits are zero),
- Other machine states.

On-chip buffering for address valued handles cases when there were less than ten cycles to broadcast an address (for exceptions or branch instructions).

AMCC bought the 4xx PowerPC intellectual property from IBM IBM Microelectronics (2004), and introduced several new 405 and 440 derivatives — all with trace ports. The newer families have improved options for triggering and filtering the trace data. In 2007, AMCC announced the 460 family and a dual-core 2 GHz PowerPC (Titan); these all have trace ports.

6.4.3 ColdFire Trace Port

The Freescale Semiconductor ColdFirey.fd y.fd architecture includes a trace port, although this is not taken out in all ColdFire derivatives. The addition of tracing pushes up the CPU cost as the trace function takes up 25% of the die (Hohl *et al.*, date unknown (probably 1995, pg 1). However, the MCF5208 in the 196 pin BGA package with the trace port costs less than the 160 pin TQFP device without a trace port. Additional information can be found in Hohl *et al.* (date unknown (probably 1995); Motorola (1997). From the JTAG details in the user manuals, it appears that the trace can be triggered by a user generated exception, however, this still needs to be confirmed as trace ports are usually activated from a JTAG interface.

6.4.4 Atmel AVR32 Trace Port

According to O'Keeffe (2006), Ashling in Ireland were involved in the design of the OCD features of the Nexus compliant trace port and trace analyzer for the Atmel AVR32. Ashling claims to be the biggest shipper of Nexus based trace analyzers on their website. These include the Nexus trace ports for the Freescale 55x and 555x PowerPC families that target powertrain– and engine management applications. The Ashling trace port analyzers cost several thousand US$, so in 2008 when Atmel launched the US$ 599 *AVR ONE!*, tracing need no longer be a luxury. By the way, the debugger and tool chain are available for free off www.atmel.com under the *Microcontrollers* section, and then under the *Tools and software* selection for the AVR32.

The AVR32 answers many an embedded developer's prayers. In the launch white paper, Atmel Corp. (2006), they mentioned many patents, however, the free patent search sites did not pick up any US-based patents using "AVR32" and "Atmel" as search criteria. The trace features allowed an external trigger via JTAG, the Nexus port, or user instructions in normal programs. The AVR32 trace *Getting Started* manual, Atmel Corp. (2008), showed the Eclipse IDE interface and inserting a trace point. There is a comprehensive manual on the AP7000; see chapter 37 for the trace and debug features (Atmel Corp., 2007, pp 892–901).

6.4.5 ARM Embedded Trace Macro

ARM's ETM is one of the more popular trace ports, particularly as ARM μPs are increasingly found in deeply embedded devices that have Flash and RAM on-chip with no external core visibility. In 1999, a joint announcement with HP outlined the trace port and trace analyzer Steffora (1999); The Wave Report (1999). HP and ARM announced that they were developing an OCD system to provide a real-time execution trace for ARM core-based products.

The co-developed execution trace system con-

sisted of an embedded trace module on-chip, a trace port analyzer connected to the target system and trace-control extensions to the software debugger running on any PC. Combined, these elements provided real-time instruction and a data trace to engineers using the ARM CPU deeply embedded within their ASICs. This trace system provided similar visibility for monitoring the execution of internal CPUs to "bonded out" chips, without the high cost or technical difficulties.

Later, HP released the E5904B trace port analyzers — option 300 was for ARM's ETM and option 060 was for the 400 series PowerPC. The manuals were dated 2001, however, they were being discontinued according to Agilent's website in 2006. ARM's manual for the Agilent probe was dated September, 2001.

In ARM Limited (2004), ARM introduced new debug tools aimed at narrower trace "funnels" and the SWD (Serial Wire Debug) subset of the JTAG pins to dedicate fewer pins to debugging deeply embedded devices. Pin count is particularly critical in mass volume items, which also need to be programmed over these same interfaces as there are no other programmable options. To enable on-board tracing, the *CoreSight* ETM and single wire debug allowed trace data for various internal buses or code to be instrumented and removed over a *Single Wire Viewer*. Details were given for selective tracing via breakpoints and counters built into the trace capture logic to ignore tracing after programmable thresholds. In another white paper (Orme, probably 2006, pg 4), figures were given of the ETM generating 1 byte for every 7 cycles, which equated to 28 000 lines of code for a 4 KByte trace buffer.

For tools vendors and silicon implementation, there are several hundred pages of debug documentation relating to CoreSight debugging at www.arm.com.

6.4.6 Xilinx PPC Trace Port

The Virtex-II Pro and Virtex-4 FX have embedded 405 PowerPC devices, which include trace ports. Unfortunately, not many Virtex-II Pro evaluation boards take out the trace port. More recent Virtex-5 FX devices embed the 440 PowerPC and evaluation boards generally take out a trace port.

In Gould and Veneman (2006), the Wind River ICE probe was described, which incidentally also used Virtex-II Pro FPGAs. In deeply embedded systems, virtual memory is often not switched on. In previous articles, embedded Linux could not be debugged with the Wind River probe, as it was unable to trace software with the MMU enabled. Apparently the Abatron BDI2000 probe is able to trace virtual memory accesses. (The advertisement for the Abatron BDI2000 on the TimeSys website lists one of the features as "Linux kernel debugging for MMU processors".)

Trace port analyzers vary in price, and from quotes received from GHS, it is non-trivial to re-Flash the on-board FPGA to change architectures even on a $10 000 SuperTrace probe. Another article Njoroge (2004) (see section 2.3.5), mentioned the problems with incompatible firmware in trace port analyzers when new processors are tested. It probably pays to rather take the money that was going to be spent on the trace port analyzer and invest in a logic analyzer. An article describing how to trace a PowerPC 405 with a logic analyzer was given in Frieden (2006*b*). However, they did not mention how to trigger or filter the trace and whether user-generated exceptions or the JTAG was used to setup the tracing hardware inside the 405.

6.5 Tracing Soft-Cores

Most soft cores now support tracing. If the RTL is available, additional instrumentation can be inserted by the user. In the case of the core being supplied as a netlist, it might be possible to trace the PC as in the Altium TSK3000.

6.5.1 Xilinx MicroBlaze™ Trace Port

In Fryer (2005), a MicroBlaze™ "soft core" instrumentation proposal and "work-in-progress" described complex triggers to minimize trace data, dump processor state to FIFOs for possible *re-execution* and gather several statistics in counters to emit via SERDES ports available in modern FPGAs. Another interesting concept in the same article was *instrumentation memory* which uses the same processor address to fetch a second memory stream of short opcodes that define instrumentation functions. In the *Summary* section, the author suggested standard proposals for trace compression, using multi-gigabit serial ports to save pins versus the silicon costs with possible Nexus standardization.

The Agilent and Xilinx jointly developed MTC (MicroBlaze Trace Core) was described in Frieden (2006*a*). The MTC gathered trace data that an external Agilent logic analyzer could display as inverse assembly language listings. The FPGA Dynamic Probe connected to the embedded instrumentation core and automatically assigned pin names from the imported FPGA design. The 16800-series logic analyzers were able to correlate signals external to the FPGA to source code executing on the soft-core CPU. At 4 GHz, these instruments are indeed capable.

Nohau developed the DebugTraceBlaze IP for debugging the MicroBlaze™ in Xilinx FPGAs. The debug block monitored the OPB (On-chip Peripheral Bus) with the option of tracing into on-chip BlockRAM. The GUI and IP resource usage were given in Wilburn (2004).

6.5.2 Lattice Semiconductor Logic Analyzer

Lattice Semiconductor released details of their LatticeMico32 free core in September, 2006. In 2008, they also announced that their future tool supplier would be Synplicity (as bundled tools), but that Mentor and others would still be supported. A tutorial to setup the *ispTRACY Logic Analyzer* is given in Lattice Semiconductor Corp. (2007). Pictures of the waveforms were shown from PC screen shots. Unlike Xilinx and Agilent, the logic analyzer core from Lattice Semiconductor appears to be bundled with their tools.

6.5.3 OpenCores Trace Ports

OpenCores.org have several free 32-bit processors as well as some trace hardware. The JTAG and trace work was last updated in 2004 (accessed in October, 2006). Some of the OpenCores work has been commercialized by Beyond Semiconductor, and tracing becomes more important as designs shift for easily instrumented FPGA testbeds to ASICs. See the website for processor manuals and their debug options. (www.beyondsemi.com)

6.5.4 Altium LiveDesign

Altium have logic analyzer IP in their *Live Designer*$_6$ tool kit that runs on the Nano board hardware. They also provide their own "free" soft core, called the TSK3000 32-bit RISC. The compiler support from Altium is the Tasking Compiler suite which supports Nios, MicroBlaze™, their own TSK3000 32-bit soft core, PowerPC and ARM.

Altium supply a configurable logic analyzer, the *LAX* block, for instrumenting FPGA designs which was configurable from 8→64-bits, and described in Altium Inc. (2005). Memory could be internal FPGA block memory or external synchronous/ asynchronous RAM with up to 20 address lines. There were also inverse assembler options for the cores supported in the Altium Designer tool chain.

The Altium instrumentation used the Nexus5001 protocol, which should help to avoid the many patents with on-chip instrumentation, particularly if shipped within products to end-users for permanent instrumentation. The soft-cores do not appear to have trace ports, so the data bus would have to be monitored. The TSK3000 has a five stage pipeline

and cache, so the details will need to be sorted out if I get to test the *LiveDesign*$_6$ tools on a large FPGA board. In the tutorial examples, the PC was attached to the *LAX* block for tracing, which appeared to be for each access, not only discontinuities (branches etc.). MIPS is becoming less of an interesting target, so do not rely on this happening.

6.6 The Ideal Debuggable-Processor circa 1982

The WCET community and RTOS designers have been asking for "the ideal processor" for some time. Over ten years ago, the call from Dropsho (1995) yielded little response, however, the problems were well known by embedded developers, and the AVR32 from Atmel appears to be one of the answers to may of the patents filed since 1980 for debugging (tracing into user addressable memory from user initiated start/ stop commands, performance counters, etc.).

Over twenty five years ago, several suggestions for more visibility into high-level debugging were made. An excellent article McLear *et al.* (1982), described important factors to consider when designing large systems (involving hundreds of developers). The authors argued for the debugger to be placed within the processor itself, and listed many limitations of an ICE for debugging any OS (Operating System) that used mapping (virtual memory or paging). The slow-downs were also problematic for RTOS work — presumably due to resources being taken from the processor by debug monitors. Many of the features that were mentioned made their way into chips several years later, however, they also became patents. For prior art references, particularly for an instrumented testbed or soft-core, have a look at some of the recommendations. They are listed briefly:

Interface Issues A pluggable standard connector that allows the debug host to be removed from the system, and the software written in a manner that uses separate interrupt vectors, memory space and can be "detached" for field– rather than lab use.

Access to processor resources There should be a means to access any part of the processor, and any part of the system that is visible to the processor, i.e., memory or I/O.

Memory mapping Before entering the debugger, all the processor registers and memory mapping must be loaded into a dedicated portion of memory, so that it is possible to work with virtual memory or paged systems.

Serial ports Two UART (Universal Asynchronous Receiver Transmitter)s are necessary; one for program downloading and communicating with the support processor; and the other to provide an interface to users.

Timers For performance measurement mainly.

I/O Ports Useful to trigger logic analyzers or external instruments.

Interrupt Controller Separate from the debugger. Possibly linked with the trace and breakpoint event recognition hardware, to run at the highest level (no user processes to run at this level), so that the debugger can always retain control of the processor.

Breakpoints Both a breakpoint instruction available to the debugger, and preferably a separate one available to the operating system. For single-stepping in user space, another option is using a bit in the status register to "trace" single instructions.

Stopping individual processes This was recently promised by GHS and WindRiver, however, it was not part of the processor, but built into their operating systems.

Matchers These are basically comparators for breakpointing on a single location, a

range of addresses, a read, or a write. This requires hardware, as it is not practical to implement a range of breakpoints using software alone.

Tracers The "transfer" trace was described as a trace where every discontinuity in the program counter was recorded — branch, call, return, etc. Higher-level debugging would require procedure-level tracing. The authors also suggested recording the "from" and "to" addresses, the instruction, the program status word and all general purpose registers. Earlier in the article, a suggestion was that procedures could be called with parameters and return addresses made visible (and changeable) by the user; saving all registers at each procedure-level trace was not considered in other trace articles, but are a necessary condition for any replay through a failure.

6.7 Nexus 5001

The following information was taken from www.nexus5001.org. The Nexus 5001™ Forum, formerly known as the *Global Embedded Processor Debug Interface Standard Consortium*, was formed in April 1998 to define and develop a much-needed general-purpose interface for the software development and debug of embedded processors.

In September 1999, the group chose the IEEE Industry Standards and Technology Organization (IEEE-ISTO) for its unique forum and support services to facilitate its efforts to advance the development, marketing, validation, and implementation initiatives in support of IEEE-ISTO 5001™-1999.

On 28th August, 2007, OCP-IP and Nexus announced a collaboration agreement on debug standards for debugging at three levels — hardware centric-, software centric– and system-on-chip debugging. A white paper, Stollon *et al.* (2007) was written some months before, but the announcement is very important in the patent minefield when combining IP from different groups and design flows.

6.8 Published Journal Articles

These articles are not guaranteed to be patent-free, however, they are useful for citing prior art and describe their methods without resorting to repetitive *legalese*.

6.8.1 Low Perturbation Address Trace Collection

Daigle, Xia and Torrellas Daigle *et al.* (1996, Date unknown) used hardware monitors in conjunction with software instrumentation. A multiprocessor Silicon Graphics POWER Station 4D/340 with four MIPS R3000 (bus-based cache-coherent multiprocessor) and a 4-processor Alliant FX/8 bus-based multiprocessor were instrumented.

The SGI monitor which was connected to the bus recorded all activity without affecting the system. The functionality of the monitor was controlled by a software-programmable Xilinx FPGA. For each bus transaction, the monitor captured the 32-bits of the physical address, whether the transaction was a read or write, the originator of the transaction (either one of the four processors for a cache miss or the I/O system for a DMA transaction), and the number of cycles elapsed since the last bus transaction. All information was stored in a trace buffer of 2 million entries capable of ignoring certain transactions. Depending on the memory reference frequency and miss rate, the trace buffer filled up in 0.5→4 seconds.

The Alliant monitor had a trace buffer of one million events connected to each of the four processors. The events included a 32-bit physical address, 20-bits for a time stamp, a read/write bit and other miscellaneous bits. The trace buffer typically filled in around 700 ms. In both systems, when the trace buffer filled to a predetermined threshold, the processors were interrupted while the trace was writ-

ten. For the SGI, the flush was to an externally mounted disk, while for the Alliant to an externally connected workstation which emptied the trace buffers and stored them to disk.

The workload events instrumented were:

- To separate application from operating system references, each *entry* and *exit* of the operating system needed to be detected. While entry was easy to detect, exit was difficult. The exception handlers in the operating system were instrumented. Operating system *exits* did not always return to the caller, and in general, the number of *entry* and *exit* code executions was different. This required examining the status register before the return instruction to determine if the return was to the application.
- To generate the dynamic call graph of a program, the sequence of subroutine invocations was detected. This involved instrumenting the beginning of each subroutine and if the program that was being traced was the operating system itself, then right after the subroutine call instruction.
- To determine the exact sequence of instructions executed, it was sufficient to determine the sequence of basic blocks — which involved instrumenting the beginning of each basic block.
- Saving virtual addresses with software assistance, as the monitor only recorded physical addresses.
- The operating system scheduler was instrumented to send the PID (process identifier) of the new running process to the tracer on each context switch.
- Entries and exist of interrupt handlers, TLB faults, system calls and other faults were instrumented. Similarly, when the caches were invalidated by the operating system, these were recorded.

The trace buffer stored addresses only, so traces needed to be encoded by dummy references called *escape accesses*. In the SGI system, a byte read from an odd address with the upper four address bits set for uncached and unmapped access within the operating system address range could be picked up in the trace buffer. Operating system accesses for instructions on a MIPS CPU were on even boundaries. The *escape access read* was to register `r0` which is wired to zero, so no memory or register contents were altered. However, the address to access must be stored in some constant, which used a register.

The instrumentation described assembler, with *.set noreorder* and *.set reorder* around the *escape accesses* to prevent the compiler from rearranging the instructions to improve the pipeline. The interrupts also needed to be disabled for multiple *escape* sequences relating to instrumenting TLB entries.

The code dilation for different programs was given as 2.1% for scientific programs with fewer branches and up to 48% for one of the compiler passes. The kernel size increased by 39%. Execution time slow down ranged from $1.5 \rightarrow 2.8\times$ when every basic block in the operating system and application were instrumented. The work represented a significant effort over a two year period, as the custom hardware had to be built and tested, plus the operating system had almost 2500 routines to instrument.

6.8.2 SHRIMP Performance Monitor

The Princeton University SHRIMP (Scalable High-performance Really Inexpensive Multi-Processor) project was described in Martonosi *et al.* (1996*a*) for a multiprocessor system based on standard Pentium (60 MHz) PCs running Linux. The main focus of the SHRIMP project was the design of hardware and software for low-cost user-level communication mechanisms. Hardware and software was designed to monitor the inter-processor communication statistics of a LAN (Local Area Network) adapter, which was connected to an Intel Paragon mesh routing backplane.

The hardware consisted of a plug-in card on an EISA bus that could monitor events on

both the network interface and EISA bus. The hardware was flexible and FPGA based, with two modes — *histogram mode* and *trace mode*. In histogram mode, it incremented a count associated with a runtime-selectable set of packet characteristics; in trace mode it appended packet-specific information to a sequential trace in DRAM.

Each SHRIMP network interface board and each performance monitor maintained a 44-bit global clock register. These were synchronously controlled across the whole system by 10 MHz clock signals distributed by the Paragon backplane. The histogram memory contained one million 40-bit words, expandable to 16 million words. The SHRIMP monitor provided efficient support for selective notification. When running in histogram mode, the monitor had a threshold-interrupt feature that notified the CPU when a count value passed a user-specified threshold.

The monitor could also be configured to use *triggered event tracing*. That is, when the monitor detected a threshold event, it not only signaled an interrupt, but also could switch from histogram to trace mode automatically. This feature allowed long periods of histogram-based monitoring, followed by detailed tracing once a particular condition was detected. This allowed full speed execution for a long time to catch sporadic system bugs.

The SHRIMP monitor did not provide basic block or procedure level tracing, but was mainly used to measure network latency.

6.8.3 SurfBoard

Surfboard (SHRIMP Usage Reporting Facility) was a performance monitor for the Princeton SHRIMP multiprocessor. The work was described in a technical report Karlin *et al.* (1999) for SHRIMP-II, which followed on from the SHRIMP monitor described in Martonosi *et al.* (1996*a*) and section 6.8.2. As with SHRIMP, the primary interconnection network was the Intel Paragon mesh routing backplane. A commodity Ethernet LAN card was a secondary interconnection.

The SNI (Shrimp Network Interface) boards implemented virtual memory-mapped communication to support protected user-level messaging, fine-grained remote updates and synchronization for shared virtual memory systems. Packet data received by a SNI board was forwarded to the Surfboard for data measurement. The direct connection from the Xpress memory bus to the SNI board allowed the SNI to snoop memory references; it was this snooping that enabled the SNI board to perform an automatic update of the communication mechanism.

The SHRIMP system used the LAN to setup the mappings and the Paragon backplane for the data transfer. The idea was to separate the mapping setup from the data transfer, where the communication model provided direct data transfer between a sender's and receiver's virtual address space. According to the authors, this also meant that the receiver of the data did not know when the data arrived (Karlin *et al.*, 1999, pg 3). This made it difficult to instrument the automatic update — imagine periodically comparing the contents of receive buffers with the backup copies of data. This would be very intrusive and still not easily identify the packet sender.

The Surfboard was connected at each SHRIMP node and captured information at the arrival of incoming packets. Once monitoring began, the SNI board sent the Surfboard a copy of each raw packet as it was received, and the monitor parsed the raw packet data to extract the fields of interest.

The SurfBoard had three independent data collection modes: *word-count*, *histogram mode* and *trace mode*. These three modes were independently controlled and could operate simultaneously. There was an external interface which allowed several Surfboards to all trigger on the same event. This with the combined global time stamp allowed system wide traces to be interleaved and correlated. Other differences to the earlier work were: histogram overflow could trigger concurrent trace; histogram

memory width reduced from 40 to 32-bit to better match the bus width; increased the histogram FIFO size to 1365 entries (from some small size not specified).

Under the *Lessons Learned* section in Karlin *et al.* (1999), the Surfboard did not come "on line" until near the end of the useful life of the project. "By this point most of the measurements that the Surfboard can make directly were already made by instrumenting the software or by direct measurements using commercially available high-speed logic analyzers."

6.8.4 PowerPC NStrace

NStrace from IBM was described as a bus-driven hardware trace facility that interfaced to an architectural simulator Sandon *et al.* (1997). A logic analyzer recorded bus activity (2 million samples), yielding instruction traces with lengths in the order of one hundred million instructions. The work was done on a PowerPC 604 processor running AIX, Windows NT, Mac OS X (Apple Mac Operating System) and Java applications.

To capture the trace, an interface board between the processor and the motherboard identified significant transactions that affect subsequent processor simulation. The interface board also compressed the data which was captured by an external logic analyzer (Tektronix DAS 9200 with 100 MB fully configured), recording about one second for an equivalent trace of 100 million instructions. The collected data included kernel, library and user applications.

The trace data was converted to a standard format for the simulator, as several different analyzers were used in different ways. Trace initiation involved two steps: achieving a known processor state and signaling the bus capture hardware to begin recording.

For the 604 simulator, the state of the processor that must be controlled to predict its subsequent behavior comprised the contents of the two caches, the contents of the two TLBs, and the values of the architected registers. The procedure of reading out the current state was to first disable and invalidate the caches, then to trigger the recording by placing a distinctive address on the bus, then to store all of the register values, then to invalidate the TLBs. Storing the register values forced them out onto the bus where they could be recorded, since the caches were disabled. At the end of this procedure, the caches were enabled so normal processing resumed.

The authors based their bus-driven processor simulator on the PVS (PowerPC Visual Simulator), but had to add access to a bus interface unit as the PVS only had an "internal" model of memory, and also provide access to a bus transaction file. External exceptions (interrupts) and the internal *decrementer* presented challenges to the mixed simulator. Another challenge mentioned was that the simulator executed strictly in program order, while the 604 was a superscalar processor that did not always execute in program order.

Several tools were developed to work with NStrace. One was for trace driven simulation. With a 100 Mbyte bus recording, the corresponding trace file generated was around 2 Gbytes. Two additional tools were developed; one to get the "big picture" and one to see the details. `opstat` gathered basic metrics, and `Browser` was able to zoom in on details (disassembled instructions, bus transactions, etc.).

6.8.5 Monster

`Monster` Nagle *et al.* (1992) was developed at Michigan University as an nonintrusive monitoring system. A Tektronix DAS 9200 logic analyzer with 512K RAM was connected to the processor pins of a DECstation 3100 (16 MHz MIPS R2000 CPU), which had external cache and was therefore able to monitor all processor activity. `Monster` also had a software component to allow tracing specific regions of interest, due to the limited memory in the

logic analyzer. Special "markers" were used to mark kernel *entry*, *exit* and other regions of interest. These markers were placed in uncached memory with interrupts disabled — a technique that guaranteed that the markers were not flushed from the pipeline or appear as phantom opcodes. The DAS 9200 logic analyzer's state machine and pattern recognition features were programmed to overcome the problem of counting cache misses as duplicate opcodes. Another problem with this method of markers was while interrupts could be masked, exceptions could not, and running with interrupts disabled distorted the running of the operating system.

The combination of hardware and light software instrumentation resulted in minimal system distortion, as the markers required up to 12 instructions and took from 24 $\rightarrow$ 48 cycles to complete — less than 1% change in kernel size. `Monster` took 12 hours to collect 11 seconds of real-time system activity for fully automated hardware collection Uhlig (1995).

6.8.6 BACH

BACH (BYU Address Collection Hardware) was a hardware monitor that collected long (over 200 million contiguous references) on a variety of hardware and software platforms. The early BACH hardware was described in Flanagan *et al.* (1992) for the i486 trace which monitored 96 signals. The traces were collected using custom hardware that monitored each CPU pin from a signal conditioning card (CPU plugs into this card). It also monitored several additional signals to indicate when the operating system was entered or exited, and when the trace buffer was full. BACH interacted with a digital I/O board in the traced system that halted the processor when the full trace buffers were emptied. The traces were concatenated to obtain complete traces. Some unused memory and I/O locations were used to annotate the trace. The data bus was part of the monitored signals, making it easy to determine values read in from I/O devices.

In a later paper Thornock and Flanagan, a history of BACH was given: the first system collected 512K samples of 96 bits at 20 MHz, the second system increased the speed to 30 MHz and was used until 1993 when logic analyzers appeared with deep memories and processor bus speeds increased rapidly. The third BACH system was based on a Tektronix TLA 520 logic analyzer which collected 512 K samples of 200-bits at 100 MHz. The current version was described as based on a Tektronix TLA 700 logic analyzer with 16 M samples of 272-bits at 200 MHz acquisition speeds.

The *Performance Evaluation Laboratory* `http://pel.cs.byu.edu/tds/`, had links to papers, technical reports, theses and traces from Intel 486 and Pentium running various flavors of Unix, Linux and MACH. MC68000 HP-UX systems were also traced, as were SPARC systems Flanagan *et al.* (1992); Thornock and Flanagan.

One of the problems that the BACH researchers noted was that when the traces were saved to secondary storage and the traced CPU spins in a tight loop, I/O requests tend to become available as soon as tracing was enabled. This was shown as a large peak for interrupts and exceptions within 5% into the trace. For the BACH trace system, the interrupts were typically for the network and timer which resulted in 4,500 references per buffer of 400,000 references (1.125%). Cache was switched off to make all accesses visible which slowed execution by a factor of 2 for the i486.

From a page dated 2000, http://traces.byu.edu/new/Documentation, the new traces were gathered using off-the-shelf equipment — a number of Tektronix TLA720 logic analyzers.

6.8.7 CITCAT — Constructing Address Traces from Cache-filtered Address Traces

Generating accurate, meaningful traces is difficult, as is their storage and distribution. A technique of calculating the traces and taking

a snapshot of the processor at a known point in the trace allows regeneration of any length trace from a relatively short program. The system Rose and Flanagan was called CITCAT, which combined the best features of instruction inlining, hardware monitoring, and processor simulation to produce a hybrid technique that was capable of generating extremely long, statistically accurate instruction traces. The work was done in the *Performance Evaluation Laboratory* at BYU, and used the BACH facilities for measuring an i486 CPU's trace.

6.8.8 NMSU TraceBase

NMSU have a number of traces and software available to manipulate traces on their *TraceBase* page at http://tracebase.nmsu.edu/tracebase.html. Several trace compression techniques were discussed. There were MIPS R2000, R3000 and DLX traces in the uniprocessor trace section. The PDATS (Packed Differential Address and Time Stamp) and PDATS II formats are discussed in sections 8.4 and 8.5.

6.8.9 MAMon Probe

Shobaki *et al.* instrumented a hardware scheduler Shobaki (1998, 1999, 2002); Shobaki and Lindh (2001). Shobaki's emphasis was hardware/ software co-simulation and a real-time coprocessor. A FPGA for monitoring provided flexibility and also filtered the data to record for external visualization.

Shobaki was in favor of simulating limited parts of a RTOS on a cycle accurate ISS, particularly context switches, IRQ (Interrupt Request) response times and flow through a program Shobaki (1998). To monitor interrupts, semaphores, inter-task communication and synchronization and other system level events, the authors described their monitor, MAMon which was a hardware based probe integrated into a FPGA and attached to an external host via a parallel port Shobaki and Lindh (2001). The probe had 128 kBytes with 10 bytes per sample, giving 13,000 events before having to empty the buffer. The time stamp was a 32-bit counter with 1 μs resolution. The EPP (Enhanced Parallel Port) parallel port ran at 2 MB/s for 200,000 events per second. With their Linux driver, the speed was slower — 1,3 MB/s.

The probe was described further in Shobaki (2002). Their SARA (Scalable Architecture for Real-time Applications) was based on three CompactPCI boards with one RTU (Real-Time Unit). The processors were PowerPC 750 CPUs. The RTU and IPU (Integrated Probe Unit) were implemented in a Xilinx Virtex-1000 FPGA. In the SARA system, the time stamp was changed to 48-bits, increasing the event record to 12 bytes.

6.8.10 TimerMon

The authors were from Seoul National University and Hansung University, both in Seoul, Korea. They argued that cache will inevitably be used for real-time systems in spite of difficulties of measuring WCET. They previously studied WCET on a MIPS R3000 Hur *et al.* (1995).

TimerMon Lee *et al.* (1998) was a PC-based ISA bus card. The authors were interested in measuring WCET with caches enabled to validate complex models used to calculate execution times. TimerMon used 2 MBytes of memory to buffer 512 K trace samples, each of which was 32-bits and up to 1024 different types. The timing resolution was 50 ns. Two systems were traced; a 33 MHz 486DX and a 133 MHz Pentium.

6.8.11 Stanford DASH Multiprocessor monitor

The DASH prototype contained 64 MIPS R3000/R3010 processors. In the paper describing the performance monitor, a 48 CPU system was running Lenoski *et al.* (1993). The performance monitor took up 20% of the directory controller processor board and consisted of three major blocks: FPGA which selected and

preprocessed events, two banks of 16 K×32 SRAMS with increment logic to count event occurrences, and a 2 M×36 trace DRAM which captured 36 or 72 bits of information on each bus transaction. One of the uses of the count SRAM was a histogram array, in which certain events were used to start, stop and increment a counter inside the FPGA. The stop also triggered the counter to be used as a SRAM address to increment.

The 2 M×36 trace array had two modes based on programming the FPGA — up to 2 M memory addresses together with the issuing processor number and read/write status were captured, and a second mode where only 1 M addresses were captured but with the directory controller's PROM address and a bus idle count for each trace entry. With software assistance, the tracer could be used to capture much longer traces with minimal distortion. The trace information could be used to do detailed analysis of reference behavior on another memory simulator. The only restriction was that only bus references were captured, not references satisfied by the processor cache. If complete address traces were desired, uncached memory spaces had to be used.

6.8.12 Shared-memory multiprocessors

In Stewart and Gentleman (1997), several processors were dedicated to monitoring other processors on the same backplane. The boards were Motorola 68040 or PowerPC boards running the `Harmony` RTOS. Their approach was to "program for observability" where a *spy task* monitored the execution of one or more other tasks without stopping them, merely by examining the state of global shared-memory. *Spy tasks* were analogous to hardware monitors, line analyzers and snoopy caches.

In a multiprocessor where the data caches were not coherent, the safest way to examine memory was through a *debug agent* running on the monitored target. When a *debug agent* ran, it was intrusive, however, by allowing a remote processor to read "stale" data (not write memory), the *spy task* caused no adverse affects on the target cache and the *debug agent* was not required. The authors debugged their systems with the data and instruction caches enabled because of the timing impact when disabling the caches. They took appropriate precautions so as not to "leave dirty footprints in the data caches".

Circular buffers and filtering were performed by the monitoring processors, which were also responsible for uploading the data to a host. The authors advised against collecting too much data, (rather too little), and that the challenge was to identify what data to collect.

6.9 Summary

Several hardware methods were summarized in this chapter. There appears to be little in common as each one tried to address a "novel" area for publication purposes. The wide interest should also cover most embedded debugging tasks. Hopefully there was plenty food for thought in this chapter, but as always, before making hardware, have a look at the patents. A few comments that follow are in relation to my intended hardware platform, and not as limitations of the previous solutions.

Instrumenting the subroutine entry and exit points of a multi-tasking system is not very useful, as tasks are periodically swapped out; depending on coding style, tasks that are swapped back to the current running task do not re-enter at the start of the task. If the task initialization is called as a separate function, then the task might be callable as a function with the entry at the beginning. Often, task initialization is at the beginning of a function with the main task body between *while (1)* { ... } statements. The context switch needs to be instrumented anyway, so task IDs can be obtained and possibly call graphs.

A comment on Surfboard may appear unfair, but does illustrate the hurdles to capturing hardware trace data. It is rather surprising that in 1999 so much effort was spent on an 8,3 MHz

EISA bus system at a well funded university like Princeton. From comments made in some of the many figures given, the researchers were not able to determine whether some anomalies were related to Linux interrupts — surely very important in studying the performance of device drivers? They might also highlight problems of debugging I/O– rather than memory mapped devices. The Pentium also had virtual I/O as far as I recall when trying to debug a QNX driver over VME (Versabus Motorola European) bus with a Pentium processor (through a PCI bridge).

The NStrace work (section 6.8.4) mentioned the trace compression preprocessor inserted before capturing the data into a commercial Tektronix logic analyzer. The work was published in 1997 which possibly goes back to 1995. The rather broad patents mentioning trace compression did not reference this work, which might be useful for anyone considering compression or filtering in a commercial tool.

A disadvantage of the `Harmony` debugger was additional processors, as well as shared-memory across a slow VME backplane. Integration improvements have moved several boards onto a single slot, so shared memory access might have less contention with I/O. Any "slave" processor stealing memory cycles for debug will probably be inside an attached FPGA or on the other side of one to the processor with high-speed dual-ported RAM between the two.

Chapter 7

Trace Patents

7.1 Introduction to Trace Patents

This chapter is a sampling of various patents in the tracing field to make potential IP developers (in the debugging field) aware of widely defined patents — all in spite of the knowledge being well known prior art or considered obvious. In many cases, using application notes or datasheets could infringe a patent issued after the datasheet publication date. Unless the effort is for personal use, rather stick to non-patented standards. As part of motivating for built-in instrumentation, it was surprising to find how much of the obvious was patented, and in the final hardware, the prior work will be referenced to avoid litigation. In some of the patents, "legal speak" refers to memory as "on-chip or external", — as if there is any other possibility! Good luck with your own efforts.

Patents are there to protect the inventor, however, the patent system is not always fair. MIPS had some patents that were awarded for dubious alignment instructions when IBM preceded them by several years and the PC has been able to do unaligned transfers since it moved beyond a byte interface of the 8088. Lexra did not even infringe these patents, was sued by MIPS, challenged MIPS in court, and won the case Ristelhueber (2001). Basically, MIPS put Lexra out of business on a frivolous case even though Lexra was beating them in technology, customers and price.

The trace patents refer to prior patents and then add what embedded designers and tools vendors have been doing for years. Suddenly, prior art becomes a patent and the wording is

so vague in an effort to cover all future possibilities. Interestingly, the microprocessor was patented almost twenty years after companies were selling them, and the patent holder then proceeded to coerce semiconductor companies for patent royalties. The patent for the microprocessor was awarded to Gilbert Hyatt almost twenty years after he filed in 1970 Pollack (1990). Intel and Texas Instruments cross-licensed several patents in 1971 and 1976, with Intel paying TI royalties for the microprocessor patent. TI filed for the patent on the microprocessor, which was awarded to Gary Boone as U.S. Patent N°3 757 306 for the single-chip microprocessor architecture on September 4, 1973 Wikipedia (Accessed April 2008).

The trace patents, mostly obtained at www.freepatentsonline.com, were found by searching on Google, and then on *Free Patents Online*. Before 2007, several patents were accessed at *Patent Storm* www.patentstorm.us. In 2006, Google added a patent search site Gonsalves (2006), at www.google.com/patents which is similar to *Free Patents Online*, however, the PDF (Portable Document Format) files are saved as the title name, rather than the patent number (preceded by "US").

The more interesting patents are briefly described in the following sections. (Also only what was allowed in the available time).

7.2 Tailorable Embedded Event Trace — TDB1291.0092

Most of the trace patents refer to an IBM patent, IBM (1991), which was awarded in 1991, however, only the abstract could be downloaded. A single patent download costs $40 for IBM's Technical Disclosure Bulletins.

IBM provided tracing in their 403-, 405- and 440 PowerPC devices, which were covered in section 6.4.2. The trace was compressed, so how are patents awarded for trace compression to other assignees afterward?

7.3 Motorola CPU32

Motorola, now Freescale Semiconductor, developed the BDM debug port in an effort to integrate limited ICE functionality into embedded targets. The BDM port froze the core while inserting special instructions into the instruction stream to read or write registers and examine memory; it did not provide trace capability. The Motorola CPU32 processor and description were available prior to April, 1997, as referenced in several patents.

7.4 Single port trace buffer architecture — US Patent N° 6148381

Jotwani (2000), covers tracing via a single-port buffer to either internal or external trace memory with circuitry to minimize buffer overflow. The patent describes the use of logic analyzers and expensive trace port analyzers as examples of equipment that the patent addresses. The trace port data can be removed serially or in parallel.

The advantage of using a single port memory is that it takes almost half the size of a dual-ported memory, thus for the same size, the amount of trace data can be doubled compared to dual-port memory. The buffer overflow mechanism gave priority to reads when the buffer was above some threshold to make space for more trace data, and "writes over reads" otherwise, as if the write could not proceed, then trace data would be lost unless the processor was stalled.

7.5 Shared embedded trace macrocell — US Patent N° 7007201

In Byrne and Holm (2006), LSI Logic described using one ETM for multiple cores in an effort to save 30 000 to 70 000 gates per ETM. There

are several companies who promote multi-core and multi-threaded processors with debugging methods (like tracing). The idea of a multiplexed JTAG or "super JTAG" was covered by FS^2 in their OCI™ products for multiple cores or IP blocks. JTAG products for multiple chains and vendors were promoted by Corelis and others, and the idea of JTAG is to have a single scan chain on a board to enable one connection to test the whole board. As instrumentation piggy-backed onto JTAG, the long scan chains were bypassed and debug registers added to speed up the debugging. Bypassing a device added one clock delay to the chain, but multiple JTAG chains within chips were fairly common as SoC designs covered a RISC device coupled to a DSP (Digital Signal Processor) (as in cell phones).

The idea of using one logic analyzer to debug multiple processors was not novel — it was a cost issue. Similarly, to share the ETM is a cost issue but in terms of gates. This patent could affect FPGA based instrumentation where an embedded logic analyzer is shared between two or more soft cores or monitors more than one external processor. The LSI Logic patent Byrne and Holm (2006), is specific to the shared ARM ETM. (The ETM infrastructure was co-developed by ARM and HP. There are several ARM manuals relating to the ETM core as well as multi-core devices. There are also a lot of patents awarded to ARM for the ETM — search on *Google Patents* for "ARM ETM".

7.6 Trace Compression — US Patent N° 6918065

A trace compression/ decompression patent was filed in 1999 Rich and Edwards (2005). The wording is extremely verbose, and not very different to the Johnson (1999); Johnson and Ha (1994); Milenkovic and Milenkovic (2003) work. IBM's prior articles, IBM Microelectronics (1998, 1999); Sandon *et al.* (1997) certainly covered trace compression and how to interpret the data, and with IBM being the company to receive the highest number of patents for the past 14 years Kivett (2008), if this was much different to IBM's previous work, then it is difficult to interpret. The Rich and Edwards (2005) patent mentions FIFOs, but that is standard practice when data is emitted at high speed to another system in a different clock domain — trace collection hardware will certainly be in a different clock domain as the target processor will be external and connected to a debug host, for example a PC. Mentioning FIFOs is common in trace related patents, one that might be useful for referencing is Poret and Mckinley (1987), as it also describes early debug features typical of a "bonded-out" chip and even the ARM ETM.

7.7 Processor including a combined parallel debug and trace port and a serial port — US Patent N° 6175914

Daniel Mann of AMD was issued several US patents, the one described in this section is N° 6175914 Mann (2001). The trace compression is described further on, which is the part of the patent that is unique.

> ...the present invention provides trace information over a communication port that is operable both as a trace port and as a parallel debug port. The trace port provides trace information indicating instruction execution flow in the processor core. The parallel debug port provides for transmission of debug information between a debug host controller and the processor. The operation of the communication port as a trace port and as a parallel debug port is mutually exclusive. The parallel debug port and the trace port physically share a majority of input/output terminals of the communication port. ...A separate serial debug port is

> also provided which can be used to enable the parallel debug port.

I have edited the following paragraphs, as the diagrams and figures are not included in this document, plus the normal patent length is several pages.

Under *Operating System and Debugger Integration* the following interesting aspects were mentioned:

The operation of all debug supporting features, including the trace cache, can be controlled through the debug port or *via processor instructions*. These processor instructions may be from a monitor program, target hosted debugger, or conventional pod-wear, if supported. The debug port performs data moves which are initiated by serial data port commands rather than processor instructions. (*Note: The later AVR32 can also trace via user-visible processor instructions*.)

Operation of the processor from conventional pod-space is very similar to operating in DEBUG mode from a monitor program. All debug operations can be controlled via processor instructions. It makes no difference whether these instructions come from pod-space or regular memory. This enables an operating system to be extended to include additional debug capabilities.

Of course, via privileged system calls such a *ptrace()*, operating systems have long supported debuggers. However, the incorporation of an on-chip trace cache now enables an operating system to offer instruction trace capability. The ability to trace is often considered essential in *real-time* applications. In a debug environment according to the present invention, it is possible to enhance an operating system to support limited trace without the incorporation of an "external" logic analyzer or in-circuit emulator.

Extending an operating system to support on-chip trace has certain advantages within the communications industry. It enables the system I/O and communication activity to be maintained while a task is being traced. Traditionally, the use of an in-circuit emulator has necessitated that the processor be stopped before the processor's state and trace can be examined, unlike *ptrace()*. This disrupts continuous support of I/O data processing.

Additionally, the trace cache is very useful when used with equipment in the field. If an unexpected system crash occurs, the trace cache can be examined to observe the execution history leading up to the crash event. When used in portable systems or other environments in which power consumption is a concern, the trace cache can be disabled as necessary via power management circuitry.

An instruction trace record is 20-bits wide and consists of two fields, TCODE (Trace Code) and TDATA (Trace Data). A valid bit V may also be included. The TCODE field is a code that identifies the type of data in the TDATA field. The TDATA field contains software trace information used for debug purposes.

The trace cache is of limited storage capacity; thus a certain amount of "compression" in captured trace data is desirable. In capturing trace data, the following discussion assumes that an image of the program being traced is available to the host system. If an address can be obtained from a program image (Object Module), then it is not provided in the trace data. Preferably, only instructions which disrupt the instruction flow are reported; and further, only those where the target address is in some way data dependent. For example, such "disrupting" events include call instructions or unconditional branch instructions in which the target address is provided from a data register or other memory location such as a stack.

Other desired trace information includes: the target address of a trap or interrupt handler; the target address of a return instruction; a conditional branch instruction having a target address which is data register dependent (otherwise, all that is needed is a 1-bit trace indicating if the branch was taken or not); and, most frequently, addresses from procedure returns. Other information, such as task identifiers and trace capture stop/start information, can also

be placed in the trace cache.

The outcome of up to 15 branch events can be grouped into a single trace entry. The 16-bit TDATA field contains 1-bit branch outcome trace entries, and is labeled as a TCODE=0001 entry. The TDATA field is initially cleared except for the left most bit, which is set to 1. As each new conditional branch is encountered, a new one bit entry is added on the left and any other entries are shifted to the right by one bit.

Using a 128 entry trace cache allows 320 bytes of information to be stored. Assuming a branch frequency of one branch every six instructions, the trace cache therefore provides an effective trace record of 1,536 instructions. This estimate does not take into account the occurrence of call, jump and return instructions.

The trace control logic monitors instruction execution via processor interface logic. When a branch target address must be reported, information contained within a current conditional branch TDATA field is marked as complete by the trace control logic, even if 15 entries have not accumulated. The target address (in a processor-based device using 32-bit addressing) is then recorded in a trace entry pair, with the first entry (TCODE=0010) providing the high 16-bits of the target address and the second entry (TCODE = 0111) providing the low 16-bits of the target address. When a branch target address is provided for a conditional jump instruction, no 1-bit branch outcome trace entry appears for the reported branch.

It may be desirable to start and stop trace gathering during certain sections of program execution; for example, when a task context switch occurs. When trace capture is stopped, no trace entries are entered into the trace cache, nor do any appear on the bond-out pins of the trace port.

Several `x86` *specific features like segments, size of instructions etc., were described for several pages. Interested readers should consult the original patent.*

When executing typical software on a processor-based device, few trace entries contain address values. Most entries are of the TCODE=0001 format, in which a single bit indicates the result of a conditional operation. When examining a trace stream, however, data can only be studied in relation to a known program address. For example, starting with the oldest entry in the trace cache, all entries until an address entry are of little use. Algorithm synchronization typically begins from a trace entry providing a target address.

The processor can provide trace synchronization information to ensure that address information for reconstructing instruction execution flow is provided in trace records with sufficient frequency. If the trace cache contains no entries providing an address, then trace analysis cannot occur. This situation is rare, but possible. A trace record (or an indication in a trace record), is provided for instructions that change the program flow such as conditional branches. However, as previously discussed, target address information is not provided in the trace record for instructions such as conditional branches where the branch target address can be determined according to whether the branch was taken or not taken. In such cases, the trace record provides only an indication of whether the branch was taken. Target or other address information is provided, however, for those instructions in which the target address is in some way data dependent and for other TCODES.

The processor determines whether each trace record includes target address information. Each trace entry having target address information causes a counter to be loaded to a predetermined value which allows the counter to count the desired maximum number of trace records generated before current program address information is provided. Thus, depending on if the counter is configured as an up counter or down counter, the counter is either loaded with zero or the maximum count, respectively. The counter counts each trace record produced which does not include target address information. When the count of

such trace records reaches the predetermined number; trace logic provides the current program address as a trace entry, thereby providing trace synchronization information.

A counter is reloaded each time a target address is generated, and decremented by one for trace entries not having an address. If the counter reaches zero, an indication is asserted to trace control, which causes a trace entry to be inserted with a code indicating that it is a synchronization entry (TCODE=0110) and a current program address. The current program address can be the most recently retired instruction or, alternatively, a pending instruction. In addition, when a synchronizing entry is recorded in the trace cache, it can also be provided to trace pins to ensure sufficient availability of synchronizing trace data for full-function ICE equipment. Trace entry information can also be expanded to include data relating to code coverage. Even without these enhancements, it is desirable to enable the processor core to access the trace cache. In the case of a microcontroller device, this feature can be accomplished by mapping the trace cache within a portion of I/O or memory space. A more general approach involves including an instruction which supports moving trace cache data into system memory.

7.8 Commonly referenced trace patents

These were "cut-and-pasted" from the HTML (HyperText Markup Language) pages into a text editor. They were not put into the bibliography and may be of use to anyone else looking at similar applications. The references include prior patents and semiconductor vendors' data books.

US Patent References:
N° 5058114, *Program control apparatus incorporating a trace function* issued on 15^{th} October, 1991, inventor was Kuboki, *et al*.

N° 5321828, *High speed microcomputer in-circuit emulator* issued on 14^{th} June, 1994, inventor was Phillips, *et al*.

N° 5357626, *Processing system for providing an in circuit emulator with processor internal state* issued on 18^{th} October, 1994, inventor was Johnson, *et al*.

N° 5371689, *Method of measuring cumulative processing time for modules required in process to be traced* issued on 6^{th} December, 1994, inventor was Tatsuma.

N° 5394544, *Software system debugger with distinct interrupt vector maps for debugging and application programs* issued on 28^{th} February, 1995, inventor was Motoyama, *et al*.

N° 5488688, *Data processor with real-time diagnostic capability* issued on 30^{th} January, 1996, inventor was Gonzales, *et al*.

N° 5491793, *Debug support in a processor chip* issued on 13^{th} February, 1996, inventor was Somasundaram, *et al*.

N° 5530804, *Superscalar processor with plural pipelined execution units each unit selectively having both normal and debug modes* issued on 25^{th} June, 1996, inventor was Edgington, *et al*.

N° 5544311, *On-chip debug port* issued on 6^{th} August, 1996, inventor was Harenberg, *et al*.

N° 5615331, *System and method for debugging a computing system* issued on 25^{th} March, 1997, inventor was Toorians, *et al*.

N° 5630102, *In-circuit-emulation event management system* issued on 13^{th} May, 1997, inventor was Johnson, *et al*.

N° 5642479, *Trace analysis of data processing* issued on 24^{th} June, 1997, inventor was Flynn

N° 5724505, *Apparatus and method for real-time program monitoring via a serial interface* issued on 3^{rd} March, 1998, inventor was Argade, *et al*.

N° 5751942, *Trace event detection during trace enable transitions* issued on 12^{th} May, 1998, inventor was Christensen, *et al*.

N° 5752013, *Method and apparatus for providing precise fault tracing in a superscalar microprocessor* issued on 12^{th} May, 1998, inventor was Christensen, *et al*.

N° 5764885, *Apparatus and method for tracing data flows in high-speed computer systems* issued on 9th June, 1998, inventor was Sites, *et al*.

N° 5768152, *Performance monitoring through JTAG 1149.1 interface* issued on 16th June, 1998, inventor was Battaline, *et al*.

N° 5771240, *Test systems for obtaining a sample-on-the-fly event trace for an integrated circuit with an integrated debug trigger apparatus and an external pulse pin* issued on 23rd June, 1998, inventor was Tobin, *et al*.

N° 5774708, *Method to test the running of a program of instructions carried out by an ASIC and ASIC pertaining thereto* issued on 30th June, 1998, inventor was Klingler

N° 5802272, *Method and apparatus for tracing unpredictable execution flows in a trace buffer of a high-speed computer system* issued on 1st September, 1998, inventor was Sites, *et al*.

N° 5812760, *Programmable byte wise MPEG systems layer parser* issued on 22nd September, 1998, inventor was Mendenhall, *et al*.

N° 5828824, *Method for debugging an integrated circuit using extended operating modes* issued on 27th October, 1998, inventor was Swoboda.

N° 5848264, *Debug and video queue for multi-processor chip* issued on 8th December, 1998, inventor was Baird, *et al*.

N° 5867644, *System and method for on-chip debug support and performance monitoring in a microprocessor* issued on 2nd February, 1999, inventor was Ranson, *et al*.

N° 5889981, *Apparatus and method for decoding instructions marked with breakpoint codes to select breakpoint action from plurality of breakpoint actions* issued on 30th March, 1999, inventor was Betker, *et al*.

N° 5903718 *Remote program monitor method and system using a system-under-test microcontroller for self-debug* issued on 11th May, 1999, inventor was Marik.

N° 5943498, *Microprocessor, method for transmitting signals between the microprocessor and debugging tools, and method for tracing* issued on 24th August, 1999, inventor was Yano, *et al*.

N° 5978902, *Debug interface including operating system access of a serial/parallel debug port* issued on 2nd November, 1999, inventor was Mann.

N° 5978937, *Microprocessor and debug system* issued on 2nd November, 1999, inventor was Miyamori, *et al*.

N° 5996092, *System and method for tracing program execution within a processor before and after a triggering event* issued on 30th November, 1999, inventor was Augsburg, *et al*.

N° 6009270, *Trace synchronization in a processor* issued on 28th December, 1999, inventor was Mann.

N° 6041406, *Parallel and serial debug port on a processor* issued on 21st March, 2000, inventor was Mann.

Intel, "Pentium Processor User's Manual vol. 3: Architecture and Programming Manual", 1994, pp. 17-1 thru 17-9.

K5 HDT, e-mail describing K5 HDT, Jan. 11, 1997, pp. 1-6.

Motorola, "CPU32 Reference Manual", pp. 7-1 thru 7-13 (admitted prior to Apr. 8, 1997).

Motorola, "MEVB Quick Start Guide", pp. 3-5 thru 7-2 (admitted prior to Apr. 8, 1997).

Revill, Geoff, "Advanced On-chip Debug for ColdFire Developers", Embedded System Engineering, Apr./May 1997, pp. S2-S4.

Advanced Micro Devices, "Am29040 Microprocessor User's Manual–29K Family", Advanced Micro Devices, Inc. 1994, pp. 12-1 through 12-26.

O'Farrell, Ray, "Choosing a Cross-Debugging Methodology", Embedded Systems Programming, Aug. 1997, pp. 84-89.

Ganssle, Jack G., "Vanishing Visibility, Part 2", Embedded Systems Programming, Aug. 1997, pp. 113-115.

Ojennes, Dan, "Debugging With Real-Time Trace", Embedded Systems Programming, Aug. 1997, pp. 50-52, 54, 56, and 58.

Many patents refer to Larus (1993), which in turn discussed the following prior trace systems; Tracer, ATUM, TRAPEDS, a tracer for Titan, MPTRACE, AE and optimal control tracing—both developed by Larus and others, and QPT (also jointly developed by Larus). The compression described by Larus used the Unix *compress* utility. Other than proposing storing the outcome of a branch into a single bit rather than a byte, most of the work was published earlier, including references to their own prior work. The storing of a single bit was not described, or how one could identify several block identifiers with a single bit. Mann described using single bits in Mann (2001), also briefly discussed in §7.7.

7.9 Summary

Patents are much like the copyright notices in GNU code. People put in a copyright notice, and someone else comes along and promptly just adds their own copyright notice without saying what is their own contribution or what was corrected/ modified. Patents should reference all the prior art, even previous patents which also had comments like *those skilled in the art will recognize that any modifications are still covered by the original invention*, or very widely defined methods of storage *DVD, modulated carrier waves (Ethernet, Internet, intranet), computer storage, optical, magnetic blah...blah* .... The software patents are even more ludicrous. The trace compression and decompression schemes are rehashes of previous work. The method of using one bit for a branch-taken and then placing 15-bits in a single location to indicate if the last 15 branches were taken, (by Mann in §7.7) is really high compression.

The "snake oil" vendors who ply IP and try to close down open legitimate work because they spent the patent application funds and were fortunate enough that the patent search attorneys were not that thorough was certainly one of the reasons why open source will become more popular. There are also moves by IBM and others to have software patents listed for discussion before being granted, and a lot of software patents repealed (including some of IBM's own).

Is there any protection in referring to previous papers or journal articles when being sued for patent infringement, even when in some cases the patent is granted after your product ships? The debug interface merchants try their best to lock everyone out of future competition. It will be interesting to see how Dafca Inc., fares with their pioneering "design for debug" tool suite called ClearBlue 2006.1. (§6.3.2).

The US Patent Office was under pressure from the European Union to revamp their registration. One improvement was the change of the protection period; from 17 years since issue, to 20 years from filing. This was to avoid "submarine patents".

Chapter 8

Trace Formats

8.1 Introduction to Trace Formats

Trace records range from several kilobytes to gigabytes. The smaller records are likely to come from FPGA based instrumentation which might use the internal block RAM or a trace port analyzer/ logic analyzer. The longer records are typical of instrumented software. How the traces are collected is independent from the visualization software, however, to use existing software packages, the traces need to be stored in a format that is understood by the visualization software. There are a number of formats mentioned in the literature. Where the layout details are available, the formats are briefly described in this chapter.

Our interest is in distributed real-time systems, and therefore we require time stamped data; ideally from a single global clock, however, the chances of a global clock are slim if heterogeneous hardware from multiple vendors is used. There are several methods to minimize skew between a global– and local clock, but these are not discussed here.

Self defining traces are flexible in that a user can generate trace data for any event of interest, not only those supported by a trace analysis tool. The method of generating the traces will often dictate how the trace is stored. If *sprintf()* calls are used, then the out-

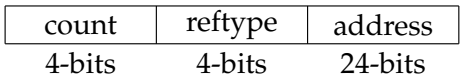
put is likely to be ASCII (American National Standard Code for Information Interchange), which takes up more disk space than binary data. For large traces, the time to convert from ASCII to binary can be significant. The benefit of ASCII is being able to recover from storage problems easier than a corrupt binary file, however, storage problems will be documented as the system is used.

The format of any trace data that we use later will be defined in header files before the data, much like ELF files. XML (eXtended Markup Language) is also popular for defining metadata, however, at this stage neither of these two formats have been tested with our trace collection hardware.

8.2 pixie Trace Format

Although described in section 4.5, the `pixie` trace format was documented in this chapter. (`pixie` was able to generate traces besides profile information for MIPS executables.)

The trace formats were taken from Smith (1991). For profiling, two binary files were generated starting with a "magic number" and a hash value created from the original file passed to `pixie`. These were followed by integer values. The one file contained offsets of the addresses of each basic block, and the other was indexed on the same basic blocks, but contained counts of the number of times each basic block executed. The addresses were word offsets from the beginning of the program, but a zero value indicated a basic block ending in a conditional branch instruction. For the trace, if only instruction trace was requested, then only `BASIC_BLOCK` and `ANNUL` records were stored. The fourteen types of records are given in Table 8.1. The trace format was a 32-bit value with three fields.



`pixie` tracing imposed a limit on the basic block size, as there were only four bits for the count field, however, Smith did show how to overcome the count limit of fifteen and provided source code for some examples.

Table 8.1: Reference types generated by `pixie`

WORD_READ	0	All load instructions
DOUBLE_READ	1	All load doubles
WORD_WRITE	2	Store word
DOUBLE_WRITE	3	Store double
BYTE_WRITE	4	Store byte
HALF_WRITE	5	Store half word
SWR	6	Store word right
SWL	7	Store word left
LWC1	8	Coprocessor 1 load word
LDC1	9	Coprocessor 1 load double
SWC1	10	Coprocessor 1 store word
SDC1	11	Coprocessor 1 store double
BASIC_BLOCK	12	Entered a basic block
ANNUL	13	Annulled the branch delay slot

8.3 Mache

The `Mache` trace paper Samples (1989), predates several patents on trace compression. The paper described compacting a trace by two orders of magnitude for instruction only traces without loosing data, by taking the delta between address traces and then using the LZ77 (Lempel-Ziv) compression.

`Mache` first converted the memory references into a *cache difference* trace, where a hit was encoded in five bytes and a miss in two bytes (for 32-bit address trace). Afterwards, the LZ77 compression scheme was used to reduce the trace. The reason for such high compression ratios in the programs traced, was that difference data has more repetition than a raw trace. Also, encoding differences between 32-bit addresses often required fewer bits than encoding the addresses. With less data to compress, the LZ77 algorithm also runs faster–comparisons were made to other trace systems that used *compress* (which also used LZ77) but achieved less compression ratios due to having larger input data sets.

Where the difference was less than a certain threshold, the data was packed into 16-bits–two bits reserved for the label and fourteen bits for the delta, which implied a threshold of 8192 for the difference. In the PDATS format, the delta was much less. Although the sample trace was shown in ASCII and the address was encoded in hexadecimal, I assume the trace format was in binary as a 32-bit address would take four bytes and then the 5^{th} byte would be used to encode the label.

8.4 PDATS

The PDATS trace format from NMSU was described in Johnson and Ha (1994). Similar to `Mache`, PDATS saved address differences between successive references of the same type (e.g., fetch, read and write). Source code was given to convert `dinero` trace format (ASCII with full address) to PDATS.

PDATS binary file format consisted of a header followed by variable length records. There were two types of files: with and without time stamps. Bits in the header byte identify the type of access (user read/write, supervisor read/write, interrupt acknowledge), time stamp code (increase by 0, 1, within 2–255, within 256–65536), the address code (offset by 4, -128 to +127, offset in 2 bytes, offset in 4 bytes), and a repeat bit. Some statistics were:

- Approximately 85% of instruction fetches on 32-bit RISC/ CISC μPs reference sequential memory words, as do from 9% to 38% of data loads and stores, making 4 byte offsets extremely common.
- In RISC processors, the time between instruction references is nearly always 1 clock cycle, making a time stamp offset of 1 very likely in RISC traces.
- In Harvard-architecture microprocessors, data references can occur simultaneously with instruction references, leading to frequent occurrences of time stamp offsets of 0.

Comparisons against long `dinero` traces and the PDATS compression gave ratios of 6.4 $\rightarrow$ 8.49, and when combined with LZ77 compression yielded ratios from 20.3 $\rightarrow$ 61.2$\times$ better. Speed improvements for access time results were also impressive — roughly ten times faster.

8.5 PDATS II

PDATS II was a follow on to earlier PDATS trace work at NMSU Johnson (1999). The improvement in trace compression over their earlier format was double without loosing information. The jump/ post-jump encoding relied on the observation that few jump-initiated sequences were longer than 15 contiguous instructions. Thus, at least 4 bits in the header byte could be used to encode the run length, usually eliminating the second record used by

PDATS to represent a jump-initiated sequence. Data-instruction clusters were the dominant feature of most traces, representing an even larger fraction of all references than the jump-initiated sequences. These were also encoded in a single byte where possible (using three bits). Although 90% of instruction offsets were exactly +4 bytes for RISC processors commonly used at the time the paper was written, the other 10% had offsets that were in units of the default instruction stride (i.e., dividing instruction offsets by 4 bytes). This sometimes allowed an offset to fit in one or two bytes which would otherwise require two to four bytes for 2's-complement representation.

The binary format was similar to PDATS, however, the time stamp and supervisor/ system reference features were removed to free some bits in the header byte. 64-bit address versions support up to 8-byte headers, whereas 32-bit versions were 5-byte headers.

8.6 Stream-Based Trace Compression

SBC (Stream-Based Compression) was described in Milenkovic and Milenkovic (2003) as a single-pass compression of combined address and instruction traces. The compression relied on separating the data and instruction accesses. The instruction stream generally consists of a limited number of different instruction streams while most of the memory references exhibit strong spacial and/or temporal locality. The SBC consists of three files; stream table file, stream-based instruction trace, and a stream-based data trace. The authors analyzed the SPEC CPU2000 traces for the stream table and noted the following, "...almost all benchmarks have fewer than 5000 different instruction streams, and 9 out of 23 benchmarks have fewer than 1000 streams, while the average stream length is fewer than 30 instructions for 18 benchmarks." The compression method was shown using the `Dinero+` trace, however, the authors mentioned that any trace input could be used (obviously a conversion function would be required to save in the SBC format).

The SBC format is:

DataHeader 1B	AddrOffset	Stride	RepCount
	1,2,4 or 8B	0,1,2,4 or 8B	0,1,2,4 or 8B

where the DataHeader bits are broken up as follows (Milenkovic and Milenkovic, 2003, Fig. 2):

Bits 7 → 5: RepCount size	Bits 4 → 2: Stride size	Bits 0 → 1: AddrOffset size
[0.5ex] 000: = 0 - 0B	000: = 0 - 0B	00: 1B
001: 1B	001: 1B	01: 2B
010: 2B	010: 2B	10: 4B
011: 4B	011: 4B	11: 8B
100: 8B	100: 8B	
101: = 1 - 0B	101: = 1 - 0B	
110: unused	110: = 4 - 0B	
111: unused	111: = 8 - 0B	

To keep the algorithm one-pass, the instruction stream blocks were not reduced to single locations in the trace file, as the later compression pass through `gzip` could pick up those repetitive patterns. The decompression was described in detail, and used a FIFO. In the (§Results), the FIFO size was 4000 entries for the compression. Note that the DataHeader

byte above is the index of the data access from the last memory-accessing instruction (there are two indices — one for addresses of instructions, and one for data accesses).

Decompression speedup times were compared against gzipped `Dinero+` traces. The decompression time for several traces formats is a large proportion of the trace analysis, and is seldom given. SBC attempted to reduce the decompression time by making the input to gzip small, which would also improve the speeds.

The impressive compression ratios were given for the SPEC CPU2000 benchmarks, however, the authors claim that due to the one-pass nature of the algorithm, it can easily be implemented in hardware to run in real-time. They were going to investigate a two-level scheme to reduce the size of the stream table; instead of instructions, each entry in the stream table keeps an index of the basic block from the Basic Block Table.

8.7 ALOG and CLOG

These formats were mentioned in Wu *et al.* (2000) and originate from Argonne National Laboratory. They both have predefined formats.

8.8 Pablo SDDF

The Pablo SDDF (Self-Defining Data Format) originated from the University of Illinois.

8.9 Jumpshot

Jumpshot is actually a visualization tool described together with a SDDF trace record. The system is described in Wu *et al.* (2000). Several problems in saving and reading trace files were addressed; namely

- Breaking up large files into multiple frames and frame directories.
- Being able to merge separate processor trace files.
- Not having to read from the beginning of a file for long traces.
- Saving the data into intervals with methods of correlating data that spans more than one file (at the beginning or end of one file with data in the neighboring file).
- A description profile file contains header information followed by interval record specifications.

Note that Jumpshot involves patented work.

8.10 Nexus

Nexus was also described in sections 6.7 and 6.5.4, but included in here as a space holder for the actual format details. The trace collection hardware is likely to use the Nexus protocol. There are two reasons for this — the Altium LiveDesign hardware supports this, and to avoid patent issues if the hardware is commercialized. The data collected can essentially be written in any format. If a 64-bit processor is used, then writing files larger than 32-bit signed numbers (2 GBytes) is not an issue.

8.11 RATCHET

In the RATCHET (Real-time Address Trace Compression Hardware for Extended Traces) system Schieber and Johnson (1994), hardware was used to compress the trace on-line, unlike `Mache` which compressed traces after they were captured. Two custom designed boards, a cache filter- and a trace acquisition board were fed into a Tektronix DAS 9200 logic analyzer, which had four channels of 90 signals with a depth of 128K samples. The normal channel width was for address-, data- and control signals, but if each sample stored two addresses and all channels were connected together, then the trace could be a million references long. The research was intended for tracing multiprocessors.

The target was a 20 MHz 68020 in a Sun 3/60 workstation. Only the address, function code, transfer size and read/write signals were recorded. The trace acquisition board packed two address references into one sample and joined the four DAS probes together. The cache filter board acted as a 16 Kbyte direct-mapped cache with a 4 byte line size. Only the addresses that missed in the cache filter board were passed on to the trace acquisition board. A processor reference counter was added that pulsed every 2^{16} processor references, as not all addresses appear in the trace. The pulse record was stored in the trace.

To determine the trace compression ratio, the number of pulses multiplied by 2^{16} gave the actual references, and counting the references in the trace gave the count of filtered references. The cache line size could be varied by disabling the lower order address lines to simulate a cache line size from $4 \rightarrow 256$ bytes. Various compression ratios were given for the ten SPEC89 benchmarks. The one million storage capacity allowed filtered traces at the beginning and middle of the SPEC benchmark suite to represent $12 \rightarrow 85$ million references, with a median of 36 million.

In the *Future Work* section, RATCHET II would plug into the backplane of a global memory multiprocessor and was being tested on a Sequent Symmetry multiprocessor. RATCHET-III would replace the logic analyzer and write to an array of disks at the same rate as the trace was captured.

8.12 Summary

Earlier trace formats were in ASCII, possibly to make them easier for people to read for debugging or to simplify parsers for various SDDF outputs. Tracing larger applications on faster computers generates more data, making the storage a problem. Tracing moved to binary formats and often includes compression. Compressing a trace at the source is not that easy at the high data rates of detailed traces on fast processors. There are several patents to compress a trace inside an integrated circuit before transmitting it to an external host.

Chapter 9

Summary

The literature mainly deals with profiling and tracing of software for single-user Unix applications. Software tracing was popular as no additional hardware was required, however, there is a price in code dilation and slow down. For a workstation, this is not a problem in my opinion, as the trace can be collected overnight. Even if collected during normal hours, the collection time for software traces should be much shorter than the subsequent analysis. We are also fortunate to have 2→3 GHz multi-core workstations in 2008, at a fraction of the earlier prices for DEC Alphas.

Many researchers modified compilers, or instrumented binary code to generate traces. Encoding methods could be carried over to hardware (software assisted) collection. As an example, distances between instructions rather than the whole address bus will mostly only require a single bit (sequential access), and jumps within a function are fairly close — normally well within 64 K locations. With FPGAs, 16-bit adders/subtracters could determine address gaps. Anything further would require a snapshot of the whole address.

Profiling will not be considered for the target, however, instruction histogramming would be fairly easily added for fixed instruction length RISC μPs (decode instructions and increment counters within the FPGA). It is not too difficult to add this into the FPGA using the adders in the newer DSP-enhanced devices, and counting instructions of interest. Loads, stores and branches would use larger counters, as these instructions are expected to occur more often in real-time work.

Even with hardware only monitoring, to gain access to certain internal processor registers, software intrusion is required. However, with internal counters that are user programmable for thresholds and can interrupt the operating system, software intrusion is minimized.

An earlier trace article Pinkerton (1969) gave a wish-list of the ideal trace tool. These are given below as they are still relevant after almost forty years.

- As much of the overhead as possible should be associated with processes independent of those being measured, and as much analysis as possible should be deferred for subsequent processing.
- A flexible selection capability for choosing among data items can significantly reduce collection overhead, reduction

time and can affect the feasibility of many applications.

- A facility which records *all* occurrences of an event type removes both uncertainty surrounding a sampling process and the analysis required to justify it.
- The most valuable monitoring tool is one which can be used during normal system operation and easily augmented to provide information about unforeseen events. This usually requires that it be carefully designed as an integral part of the system at a low level.

9.1 Increasing Device Complexity

Embedded targets for commodity goods like MP3 players, cell phones and high volume computer peripherals are both cost– and power sensitive. The SoC designs that target these markets trim the non-essential transistors. The systems market for high-end embedded processors is less cost– or power sensitive, however, it appears that the demand is not there for highly instrumented single-chip devices (otherwise semiconductor companies would make them). The cost to system integrators for logic analyzers and instrumented test benches is amortized over many products, whereas built-in instrumentation that end-users might not care about has to be included in the selling price.

9.2 Challenges of Tracing Embedded Targets

It is unreasonable to assume that embedded targets will continue to trail the desktop for speed by almost an order of magnitude. Even if they do, standard operating systems and commodity hard drives on PC class workstations will find this data rate impossible to handle. A routine backup of copying about 39 GBytes of data from one hard drive to another on a dual, 64-bit PowerPC Apple G5 workstation took almost seventy minutes. The command issued was:

```
tar cf - ./* | (cd /Volume/disks03/home ; tar xf
```

which maintains the file permissions and date information[1]. However, this elapsed seventy minutes represents 8400 GBytes of data ($70 \times 60 \times 2$) if collected at 2 GBytes per second. Even if the data was reduced ten fold, the workstation ("state-of-the-art" for a commodity workstation at the time), was not able to write 840 GBytes as this was twenty times more than the 39 GBytes measured above. Clearly, continuous tracing will be very expensive even for a single target. For distributed systems, each processor requires a trace. On multi-core and SMT devices, several trace ports will be necessary per device or a very high speed single trace port. Real-time systems also require periodic time stamps interleaved with the data.

Although a trace port only emits address data on branches and whether the branch was taken, embedded systems also require a snapshot of the data read or written between peripherals. Memory accesses are deterministic and could be reconstructed from a trace, however, when interacting with external devices, many branches result from values read in from an analog-to-digital converter or digital sensors. These values cannot be reconstructed from an "address only" trace. A trace port does not emit any timing data either. There are several trace compression schemes; (PDATS and SBC), but these were designed for deterministic workloads on standard computers. The PDATS II format could be used with FPGA assistance.

If we cannot trace continuously over long periods of time, then the next best alternative is to trace into a circular buffer at a high level of detail and to external storage at the procedure call level. When an event of interest is detected, the trace is halted while the circular buffer is drained to a storage device. An area of interest is debugging transient overload conditions

[1] Admittedly not a "raw" device, but then end-users would need a better Linux/Unix background.

to see where industrial systems fail to meet design specifications. The circular buffer can use plug-in commodity DDR memory modules. If the target does not require any trace memory, simply remove the DIMM (Dual In-line Memory Module) memory. The trace memory is not directly connected to the processor buses, but via a FPGA that has access to the peripheral or local bus and the trace port if available. The trace can be compressed on the FPGA and comparators can select events to capture (data reads or writes, which devices, time stamps). The watchdog should be placed in the FPGA so that a transient overload can automatically terminate any tracing. Context switches can also be made to internal FPGA dual-ported memory which can be included in the trace without processor intervention. This will provide a level of visibility that even a trace port cannot provide and does not require modifying any source code except the location of the context save area. For debugging any embedded target, the context area location must be known anyway and is simply given its address in a linker file. If a transient overload does not cause the watchdog to trigger, the scheduler will require some logic to detect a missed deadline. We assume any hard real-time system is able to detect a missed deadline. This can be used to write to a single address in the FPGA to halt the circular trace. How the trace memory is drained is purely a cost issue; if the main processor is used, then the system is no longer able to control any process during this time. This could be used during debugging, commissioning or upgrading. For a production system, if a missed deadline causes the system to halt, then it is possible to use the main processor, however, it is more likely that the machine specification will call for a degraded mode of operation or a lower product throughput. In this case, a small ARM connected to the FPGA or even one of the free processors from a FPGA vendor embedded into the device could extract the data. The data is best sent over a SERDES link to a central collection instrument as this does not require elaborate Ethernet links or stacks. Ethernet could also be used with one of the highly integrated 32-bit cores at minimal cost, which is independent of any main processor Ethernet communications or overhead.

Hardware for very large data acquisition systems is available from Vmetro at data rates that are suitable for branch traces. Vmetro's multiple disk arrays were designed for radar data acquisition and replay in well funded development laboratories. However, in early 2006 IBM reported reaching speeds of just over 5 GHz for the Power6+ Shankland (2006), so the faster devices will stay ahead of commercial trace collection hardware. Admittedly this is not targeting embedded use, but performance monitoring will be critical.

For embedded systems with circular buffers and no Ethernet connection, low cost 8 GByte USB (Universal Serial Bus) Flash drives or SD (Secure Digital) Flash cards could be used. These are becoming popular for saving MP3 songs and transporting data between PCs, helping to reduce their prices due to high volumes, however, transfer rates are slow.

From a technology viewpoint, the commodity items are pedestrian, but still difficult to debug. High speed programmable devices have been announced by various vendors (not Xilinx, Altera, Lattice or Actel), but support for low volume is not likely to be good. For those with deep pockets; some time ago, 20 GHz FPGA work was described at Renssellaer in Goda *et al.* (2000, 2001); Guo *et al.* (2003). Around the same time (also from research done at Rensselaer), 37 GHz SERDES chips were announced EE Times (2002). There are many SERDES suppliers, so taking signals at high data rates off narrow lines from trace ports should not be too difficult. In Cisco Systems (2004); Halfhill (2004); Tensilica (2004), IBM Microelectronics manufactured a router chip for Cisco's 40 GB/s CRS-1 ASIC that integrated 100 2,5 Gbps SERDES channels on a chip and as many as 188 embedded parallel Tensilica Xtensa CPU cores.

9.3 Modifying Compilers

In Chapter 4, several tools modified compilers to output profiling or trace code. We do not have an interest in compilers to the extent of being able to quickly modify a complete tool chain. In many embedded projects, developers change between architectures and often work on different processors (PowerPC on one six month project, followed by perhaps a three month ARM project). Even within an architecture, 32-bit or 64-bit requires different compiler settings, and here both MIPS and PowerPC have compelling offerings, so concentrating only on 32-bits is not an option either.

The assembler level changes are not that simple on processors that modify the instructions for pipeline hazards or expand simplified instructions–fairly common on MIPS for shuffling instructions and using the branch delay slot, and both MIPS and PowerPC for expanding 32-bit constant loads into two instructions. The optimizer or linker could change register usage if optimizing over the whole program.

At the linked level, the object code will not go through any further changes for optimization, however, inserting instrumentation in the binary must change several branches or jumps. On MIPS the delay slot is a particularly difficult problem to instrument for branch traces.

After the ideal of being able to dump for minutes into large gigabytes of high-speed RAM met with the harsh reality of a not so portable hardware design (at great cost in time and money), we had to start looking into filtering and trace compression schemes. The instrumentation would also preferably be detachable so that not all customers had to pay the penalty of instrumentation if they had no interest in changing any delivered project.

The software tracing work by Ball and Larus in optimally tracing or profiling by careful placement of counters was extremely useful, and will be used to instrument source code manually with `#define` and `#ifdef` brackets for instrumentation code. The accompanying hardware will act like a logic analyzer with several regions of code that can be profiled, traced, recorded for waveform capture, or timed. The manual source code instrumentation can conditionally enable or disable the different hardware instrumentation options with very little overhead–globally setting or disabling markers similar to the work in section 3.10 on *iWatcher*.

The hardware work on replay was more for viewing the program in "slow motion" to see the order of procedures or code coverage. The idea of deterministic execution by simulation of a program guided by traces is interesting, but the effort of modeling the environment would be more than making a highly instrumented testbed. Being able to scan waveforms similar to WindRiver's WindView in both the forward and reverse direction through a trace is probably enough to debug embedded real-time systems, but the final proof will be in the testing.

Reading through the vast literature on the subject of tracing and profiling did tend to alter my course, but eventually when faced with a large legacy system to upgrade on a steel mill, very flexible hardware instrumentation using big FPGAs with light software assistance became more important than trying to sift through simulated data. Even the hardware probes on IDE front-ends failed when trying to read out a trace generated into local memory by a nano trace feature of a processor (AVR32 with IAR compiler and MkII JTAG probe)–the trace could not be saved to disk or filtered, or even printed out and was just too short to be useful for real-time work. WindRiver produced a new JTAG probe with 100 MHz advertised data rates (and using patented technology!), but renting the tool chain at the price of the lease on a motor vehicle was not attractive for exploratory work without major contracts. GHS was not much better–their software had annual renewal licence fees which even at academic prices was harsh, and the one upgrade no longer worked with their own probe on the PowerPC.

There are many problems that only a logic ana-

lyzer can solve. FPGAs made it possible to embed a reasonable logic analyzer with powerful trigger options into an instrumented testbed, plus a small embedded soft core makes it possible to separate the loading of test trigger conditions and extracting the trace without any interaction from the processor under test. During development, downloads consume a large portion of any edit-compile-test loop, and the BSP supplied with most boards is not very useful for real time work when the monitor size is over a megabyte and the RTOS is meant to by under a megabyte. Loading up into bootable RAM, made it easier to transition to deeply embedded systems using Flash, as there were no surprises when the Flash code "forgot" to initialize the boot chip select or memory refresh controller.

Trace probes never came down in price during the long search for a reasonable testbed solution and the gap in the vendor's sales staff knowledge and the available silicon became much larger as the brochures became more impressive. The two largest RTOS vendors' representatives were asked for trace files and demonstrations, but these never materialized, even when making an appointment to visit GHS in the UK (which they cancelled the week before when I was already in the UK). The testbed become the only viable solution able to track the large number of targets that would be used with or without trace ports over the next fifteen years of my useful design life–the FPGA only has to track the memory access speeds of the processors used, there is no need to sample at much higher rates as for general purpose logic analyzers and the graphics on several screens driven by Mac OS X and hundreds of gigabytes of commodity storage on SATA (Serial ATA) or Firewire hard drives was less than anything from an instrumentation vendor. The flexibility of being able to instrument at the source code level without any concern for compiler tool chains (or even which target processor was used), and being able to modify the Verilog or VHDL for the trace capture was especially beneficial and worth the development effort.

A lot of money was wasted in this search; some my own, other from research grants. Expensive software at the price of a small vehicle and a drawer full of evaluation boards and JTAG probes will hopefully convince others who might read this report to investigate a highly instrumented testbed. Software has no resale value and in many cases the license cannot be transferred — which might not be such an issue in an academic setting where the price is a fraction of the resale value to industry. To your pension fund, family holiday budgets, and lost opportunity, expensive software that does not work and the many looming patents (software particularly), hopefully the papers referenced in this report give prior art references in the unfortunate event of having to defend a suspect patent suite, and to avoid many claims of simple debugging in big legacy upgrades.

Finally, to those authors who chose to publish rather than patent, a big thank you. This is hopefully the preview for the actual instrumentation platform, the *SLAprobe — Software Logic Analyzer Probe*.

Bibliography

Advani, D.M., Byron, M.J., Hansell, S.R., Li, T.M.C., Marino, J.P., Panda, R.D., Pierce, J.A., Wang, K., Weinel, D.G. and Welch, R.S. (2000 2 May). Visualization tool for graphically displaying trace data produced by a parallel processing computer — US Patent N° 6057839. Assignee IBM, filed 26^{th} Nov, 1996.
Available at: `www.freepatentsonline.com/6057839.html` 30

Agarwal, A., Sites, R.L. and Horowitz, M. (1986). ATUM: A New Technique for Capturing Address Traces Using Microcode. In: *ISCA '86: Proceedings of the 13th annual international symposium on Computer architecture*, pp. 119–127. IEEE Computer Society Press, Los Alamitos, CA, USA. ISBN 0-8186-0719-X. 25, 61, 62

Agilent Technologies (1973 May). The Logic Analyzer (AN 167-1. 67

Alexandrov, A., Bratanov, S., Fedorova, J., Levinthal, D., Lopatin, I. and Ryabtsev, D. (2007 15 Nov). Parallelization Made Easy with Intel Performance-Tuning Utility. *Intel Technology Journal*, vol. 11, no. 4, pp. 275–286. ISSN 1535-864X.
Available at: `http://www.intel.com/technology/itj/archive.htm` 14

Altium Inc. (2005 15 Dec). LAX Configurable Logic Analyzer. Core Reference CR0158 (v1.0). 73

AMD Inc. (2008*a*). AMD SimNow^{TM} simulator.
Available at: `http://developer.amd.com/tools/simnow/default.aspx` 43

AMD Inc. (2008 20 Mar*b*). *AMD SimNow^{TM} Simulator 4.4.3 User's Manual*. Revision 1.90.
Available at: `http://developer.amd.com/tools/simnow/default.aspx` 43

Ammons, G., Ball, T. and Larus, J.R. (1997). Exploiting Hardware Performance Counters with Flow and Context Sensitive Profiling. In: *SIGPLAN Conference of Programming Language Design and Implementation*, pp. 86–96.
Available at: `citeseer.nj.nec.com/ammons97exploiting.html` 54

Anderson, J., Berc, L., Chrysos, G., Dean, J., Ghemawat, S., Hicks, J., Leung, S.-T., Lichtenberg, M., Vandevoorde, M., Waldspurger, C.A. and Weihl, W.E. (1998 Oct). Transparent, Low-Overhead Profiling on Modern Processors. Compaq Computer Corporation.
Available at: `citeseer.nj.nec.com/18302.html` 10

Anderson, J.M., Berc, L.M., Dean, J., Ghemawat, S., Henzinger, M.R., Leung, S.-T.A., Sites, R.L., Vandevoorde, M.T., Waldspurger, C.A. and Weihl, W.E. (1997 Nov). Continuous Profiling: Where Have All the Cycles Gone? *ACM Transactions on Computer Systems*, vol. 15, no. 4, pp. 357–390. 10, 24

ARM Limited (2004). *CoreSight Technology: System Design Guide*. Revision r0p0. 72

Arvind, Asanović, K., Chiou, D., Hoe, J.C., Kozyrakis, C., Lu, S.-L., Oskin, M., Patterson, D., Rabaey, J. and Wawrzynek, J. (2005). Project Summary for NSF Solicitation 04-588. CRI:RAM: Research Accelerator for Multiple Processors – A Community Vision for a Shared Experimental Parallel HW/SW Platform.
Available at: `http://ramp.eecs.berkeley.edu/index.php?publications` 48, 60

Atmel Corp. (2006 Feb). AVR32 32-bit MCU/DSP Introduction.
Available at: `www.atmel.com/products/AVR32/overview/overview1.asp` 5, 71

Atmel Corp. (2007 Jul). *AVR32 32-bit Microcontroller. AT32AP7000 Preliminary*. Atmel. Doc32003.pdf. 71

Atmel Corp. (2008 Apr). *AVR ONE! Quick-start Guide*. Atmel. Doc32103.pdf. 71

Austin, T., Larson, E. and Ernst, D. (2002). SimpleScalar: An Infrastructure for Computer System Modeling. *Computer*, vol. 35, no. 2, pp. 59–67. ISSN 0018-9162. 4, 45

Ball, T. and Larus, J.R. (1992). Optimally Profiling and Tracing Programs. In: *POPL '92: Proceedings of the 19th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pp. 59–70. ACM Press, New York, NY, USA. ISBN 0-89791-453-8. 19, 53

Berrendorf, R. and Mohr, B. (2003 Jan). PCL – The Performance Counter Library: A Common Interface to Access Hardware Performance Counters on Microprocessors (version 2.2).
Available at: `www.fz-juelich.de/zam/PCL/PCLcontent.html` 19

Bhandarkar, D. and Clark, D.W. (1991 Apr). Performance from architecture: Comparing a RISC and a CISC with similar hardware organization. *ACM SIGARCH Computer Architecture News*, vol. 19, no. 2, pp. 310–319. 21

Black, B. and Shen, J.P. (1998). Calibration of Microprocessor Performance Models. *Computer*, vol. 31, no. 5, pp. 59–65. ISSN 0018-9162. 35

Boulton, C. (2005 6 Dec). Sun Opens Design to New T1 Chip. internetnews.com.
Available at: `www.internetnews.com/ent-news/article.php/3568991` 32

Brandow, G. (2005 14 Dec). IBM takes next step in opening Power Architecture to research and education community. Power.org Press Release. 37

Brown, J. (2005 Dec). Just like being there: Papers from the Fall Processor Forum 2005: Application-customized CPU design. The Microsoft Xbox 360 CPU story. Take off the IBM developerWorks site. 17, 36, 70

Burger, D., Austin, T.M. and Bennett, S. (1996). Evaluating Future Microprocessors: The SimpleScalar Tool Set. Tech. Rep. CS-TR-1996-1308.
Available at: `citeseer.ist.psu.edu/burger96evaluating.html` 44

Burger, D.C. (1998). *Hardware Techniques to Improve the Performance of the Processor/Memory Interface*. Ph.D. thesis, Computer Sciences, Wisconsin-Madison. 44

Business Wire (2008 16 Apr*a*). eASIC Shatters FPGA Performance With 235MHz LEON3 Processor.
Available at: `http://www.businesswire.com/portal/site/google/?ndmViewId=news_view&newsId=20080416006402&newsLang=en` 33

Business Wire (2008 21 Jan*b*). Vitesse Unveils VScope Embedded Waveform Viewing Technology to Revolutionize Signal Integrity Analysis. 68

Byrne, D.A. and Holm, J. (2006 28 Feb). Shared Embedded Trace Macrocell – US Patent N° 707201 B1. Assignee LSI Logic, filed 16th Nov 2001.
Available at: `http://www.freepatentsonline.com/7007201.html` 66, 84, 85

Callahan, T., Hauser, J. and Wawrzynek, J. (2000 Apr). The Garp Architecture and C Compiler. *IEEE Computer*, vol. 33, no. 4, pp. 62–69. 5

Carver, K., Fleckenstein, C., Vasseur, J.L. and Zeisset, S. (1999 Nov). Porting Operating System Kernels to the IA-64 Architecture for Presilicon Validation Purposes. *Intel Technology Journal*, vol. 3, no. 4.
Available at: `http://www.intel.com/technology/itj/archive/1999.htm` 27, 41

Casmira, J., Fraser, J., Kaeli, D. and Meleis, W. (1998). Operating System Impact on Trace-Driven Simulation. In: *SS '98: Proceedings of the The 31st Annual Simulation Symposium*, p. 76. IEEE Computer Society, Washington, DC, USA. ISBN 0-8186-8418-6. 49

Cavanagh, C., Sine, C. and Warner, L. (2008 3 Apr). Verification Patterns in Addition to RVM. In: *Synopsys Users Group (SNUG)*.
Available at: `www.opensparc.net/publications/presentations/sung08-verification-patterns-in-addition-to-rvm.html` 34

Chen, C.-H., Johnson, M.C. and Lang, D.J. (1997 24 Jun). Distributed Trace Data Acquisition System – US Patent N° 5 642 478. Assignee IBM, filed 29th Dec, 1994.
Available at: `http://www.patentstorm.us/patents/5642478-claims.html` 66

Chen, J.B. and Eustace, A. (1995 Nov). Kernel Instrumentation Tools and Techniques. Tech. Rep. TR-26-95, Center for Research in Computing Technology, Harvard University.
Available at: `citeseer.nj.nec.com/159910.html` 49

Choquette, J., Gupta, M., McCarthy, D. and Veenstra, J. (1999 July-Aug). High-Performance RISC Microprocessors. *IEEE Micro*, vol. 19, no. 4, pp. 48–55. 31, 38

Christie, D. (1996 Apr). Developing the AMD-K5 Architecture. *IEEE Micro*, vol. 16, no. 2, pp. 16–26. ISSN 0272-1732. 42

Chung, S.W., Park, G.H., Suh, H.-J., Kim, H.J., Im, J.B., Park, J.W. and Park, S.B. (2006 5 May). Sim-ARM1136: A case study on the accuracy of the cycle-accurate simulator. *Microprocessors and Microsystems*, vol. 30, no. 3, pp. 137–144. 38, 39

Cisco Systems (2004). Rajiv Deshmukh: Brawny New Chips Push CRS-1 to Heights of Processing Power and Scalability.
Available at:
`http://newsroom.cisco.com/dlls/innovators/Core_IP/rajiv_deshmukh_profile.html` 99

Clark, I. (2007 Apr). Debugging Embedded Systems.
Available at: `www.secondvalleysoftware.com/research/pdfs/debugEmbed.pdf` 5

Clark, I. (2008 Sep). Instrumentation and Measurement of Real-Time Kernels, Part II – A Visualisation Survey. Unpublished work. 6

Cmelik, B. and Keppel, D. (1994). Shade: A Fast Instruction-Set Simulator for Execution Profiling. In: *SIGMETRICS '94: Proceedings of the 1994 ACM SIGMETRICS conference on Measurement and modeling of computer systems*, pp. 128–137. ACM Press, New York, NY, USA. ISBN 0-89791-659-X. 32, 33

Cmelik, R.F. and Keppel, D. (1993). Shade: A Fast Instruction-Set Simulator for Execution Profiling. Tech. Rep. UWCSE 93-06-06, University of Washington, Seattle.
Available at: `Citeseercmelik93shade.html` 30, 32, 52, 62

Corcoran, J. and Poulton, K. (2007). Analog-to-Digital Converters. *Agilent Measurement Journal*, vol. Issue 1, no. First Quarter, pp. 34–40.
Available at: `http://cp.literature.agilent.com/litweb/pdf/5989-5911EN.pdf` 68

Couleur, J.F., Gudenschwager, P.F., Shelly, W.A. and Bahrs, D.L. (1969 Oct). Data Processing Unit for Providing Sequential Memory Access and Record Thereof – US Patent n° 3473154. Assignee General Electric Company, filed 4^{th} May, 1964. 67

Daigle, R., Xia, C. and Torrellas, J. (1996 Mar). Low Perturbation Address Trace Collection for Operating System, Multiprogrammed, and Parallel Workloads in Multiprocessors. Tech. Rep., Center for Supercomputing Research and Development, University of Illinois at Urbana−Champaign.
Available at: `citeseer.nj.nec.com/daigle96low.html` 4, 75

Daigle, R., Xia, C. and Torrellas, J. (Date unknown). Low Perturbation Address Trace Collection with Simple Hardware Performance Monitors. Center for Supercomputing Research and Development, University of Illinois at Urbana−Champaign.
Available at: `citeseer.nj.nec.com/314664.html` 75

Danzig, P.B. and Melvin, S. (1990 Jan). High Resolution Timing with Low Resolution Clocks and A Microsecond Resolution Timer for Sun Workstations. *Operating Systems Review, ACM Press*, vol. 24, no. 1, pp. 23–26. 8

Davis, S. (2000). On-chip, Real-time Logic Analysis with ChipScope ILA. *Xcell Journal*, vol. 36, no. Second Quarter, pp. 19 – 21. 68

Dean, J., Hicks, J.E., Waldspurger, C.A., Weihl, W.E. and Chrysos, G.Z. (1997). ProfileMe: Hardware Support for Instruction-Level Profiling on Out-of-Order Processors. In: *International Symposium on Microarchitecture*, pp. 292–302.
Available at: `citeseer.nj.nec.com/dean97profileme.html` 10, 21

Dellarocas, C.N. (1991 Jun). *A High-Performance Retargetable Simulator for Parallel Architectures*. Master's thesis, Electrical Engineering and Computer Science, MIT.
Available at: `http://publications.csail.mit.edu/lcs/pubs/pdf/MIT-LCS-TR-505.pdf` 30

Design & Reuse (2007 22 May). Fs2 introduces system navigator tools for mips32 74k core debug.
Available at: `http://www.us.design-reuse.com/news/15928/fs2-system-navigator-tools-mips32-74k-core-debug.html` 69

Desikan, R., Burger, D., Keckler, S. and Austin, T. (2001*a*). Sim-alpha: a Validated Execution-Driven Alpha 21264 Simulator. Technical Report TR-01-23, Department of Computer Sciences, University of Texas at Austin.
Available at: `citeseer.ist.psu.edu/desikan01simalpha.html` 45

Desikan, R., Burger, D. and Keckler, S.W. (2001 Jun*b*). Measuring Experimental Error in Microprocessor Simulation. In: *28th Annual International Symposium on Computer Architecture*, pp. 266–277.
Available at: `citeseer.ist.psu.edu/desikan01measuring.html` 44

Dropsho, S. (1995 Dec). Real-Time Penalties in RISC Processing. Tech. Rep. TR-95-110, University of Massachusetts – Amherst. 74

Dulong, C., Shrivastav, P. and Refah, A. (2001 Aug). The Making of a Compiler for the Intel Itanium Processor. *Intel Technology Journal*, vol. 5, no. 3.
Available at: `http://www.intel.com/technology/itj/archive/2001.htm` 41

EE Times (2002 18 Feb). Multirate serdes chip set takes aim at OC-768 apps. EE Times.
Available at: `www.eetimes.com/story/OEG20020215S0032` 99

Eggers, S.J., Keppel, D., Koldinger, E.J. and Levy, H.M. (1990 May). Techniques for Efficient Inline Tracing on a Shared-Memory Multiprocessor. *Proceedings of the 1990 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, pp. 37–47. 52

Emer, J., Ahuja, P., Borch, E., Klauser, A., Luk, C.-K., Manne, S., Mukherjee, S.S., Patil, H., Wallace, S., Binkert, N., Espasa, R. and Juan, T. (2002). Asim: A Performance Model Framework. *Computer*, vol. 35, no. 2, pp. 68–76. ISSN 0018-9162. 38

Eranian, S. (2005 7 Feb). The perfmon2 interface specification. Tech. Rep. HPL-2004-200(R.1), HP Labs.
Available at: `www.hpl.hp.com/techreports/2004/HPL-2004-200R1.pdf` 19

Eustace, A. and Chen, J.B. (1995 Aug). ATOM Kernel Instrumentation Guide Version 0.3: Preliminary DRAFT. Tech. Rep., Center for Research in Computing Technology, Harvard University.
Available at: `citeseer.nj.nec.com/21256.html` 48, 49

Feigin, E.J. (1999 5 Apr). A Case for Automatic Run-Time Code Optimization. Honours Thesis, Harvard. 8

First Silicon Solutions (2002 28 Jan). News release: First Silicon Solutions announces the first Real-Time trace enabled tool chain for the new MIPS32 4KE and 4KSc cores.
Available at: `www.fs2.com/1-28-2002.htm` 69

Flanagan, J.K., Nelson, B.E., Archibald, J.K. and Grimsrud, K. (1992 Sep). BACH: BYU Address Collection Hardware, The Collection of Complete Traces. In: *Proceedings of the 6th International Conference on Modeling Techniques and Tools for Computer Performance Evaluation*, pp. 128–137.
Available at: `http://pel.cs.byu.edu/techreports/BACH.pdf` 24, 79

Flower, R., Luk, C.-K., Muth, R., Patil, H., Shakshober, J., Cohn, R. and Lowney, P.G. (2001). Kernel Optimizations and Prefetch with the Spike Executable Optimizer. In: *4th Workshop on Feedback-Directed and Dynamic Optimization (FDDO-4)*.
Available at: `citeseer.nj.nec.com/478950.html` 12

Freescale Semiconductor (2007 3 Oct). Evaluate the Freescale MPC8572E SoC Today – Without the Chip! Freescale news letter. 45

Frieden, B. (2005). Debugging with Combined Oscilloscope and Logic Analyzer Measurements. *Xcell Journal*, vol. 53, no. Second Quarter, pp. 67 – 69. 68

Frieden, B. (2006 5 Jul*a*). Fast insight into MicroBlaze-based FPGA designs with the MicroBlaze Trace Core (MTC). Programmable Logic Design Line. Agilent article.
Available at: `www.pldesignline.com` 73

Frieden, B. (2006 19 Jul*b*). How to use the Trace Port on PowerPC 405 cores. Programmable Logic Design Line. Agilent article.
Available at: `www.pldesignline.com` 72

Fryer, R. (2005 Apr). FPGA Based CPU Instrumentation for Hard Real-Time Embedded System Testing. *ACM SIGBED Review. Special Issue: IEEE RTAS 2005 work-in-progress*, vol. 2, pp. 39–42. ISSN 1551-3688. 21, 73

FS^2 and Tektronix (2006 20 Sep). Tektronix and FS^2 Collaborate on Real-Time Logic Debug Solution for Xilinx FPGAs. *FPGA and Structured ASIC Journal*.
Available at: `www.fpgajournal.com` 68

Ganapathy, G., Narayan, R., Jorden, G., Fernandez, D., Wang, M. and Nishimura, J. (1996). Hardware emulation for functional verification of K5. In: *DAC '96: Proceedings of the 33rd annual conference on Design automation*, pp. 315–318. ACM, New York, NY, USA. ISBN 0-89791-779-0. 42

Gibson, J., Kunz, R., Ofelt, D., Horowitz, M., Hennessy, J. and Heinrich, M. (2000). FLASH vs. (Simulated) FLASH: closing the simulation loop. In: *ASPLOS-IX: Proceedings of the Ninth International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 49–58. ACM Press. ISBN 1-58113-317-0. 29

Goda, B., McDonald, J.F., Carlough, S., Krawczyk, T. and Kraft, R. (2000 May). SiGe HBT BiCMOS FPGAs for fast reconfigurable computing. *IEE Proc.-Comput. Digit. Tech.*, vol. 147, no. 3.
Available at: `http://inp.cie.rpi.edu/research/mcdonald/frisc/publications&patents/Goda_SiGe_HBT_BiCMOS_FPGAS.pdf` 99

Goda, B.S., Kraft, R.P., Carlough, S.R., Krawczyk, T.W. and MacDonald, J.F. (2001 Aug). Gigahertz Reconfigurable Computing using SiGe HBT BiCMOS FPGAs. In: *Field-Programmable Logic and Applications, 11th International Conference FPL*.
Http://inp.cie.rpi.edu/research/mcdonald/frisc/publications&patents/GigahertzReconfigComputing.pd
Available at: `http://inp.cie.rpi.edu/research/mcdonald/frisc/publications&patents/GigahertzReconfigComputing.pdf` 99

Goldberg, A. and Hennessy, J. (1993 Jan). Mtool: An Integrated System for Performance Debugging Shared Memory Multiprocessor Applications. *IEEE Transactions on Parallel & Distributed Systems*, vol. 4, no. 1, pp. 28–40. 8, 50

Gonsalves, A. (2006 14 Dec). Google launches patent search. Embedded.com. 84

Gould, J. and Veneman, S. (2006). Tracing Your Way to Better Embedded Software. *Xcell Journal*, vol. 58, no. Third Quarter, pp. 70 – 72.
Available at: `www.xilinx.com/publications/xcellonline/xcell_58/xc_pdf/p070-072_58-windriver.pdf` 72

Graham, S.L., Kessler, P.B. and McKusick, M.K. (1982). Gprof: A Call Graph Execution Profiler. In: *SIGPLAN '82: Proceedings of the 1982 SIGPLAN Symposium on Compiler Construction*, pp. 120–126. ACM Press, New York, NY, USA. ISBN 0-89791-074-5. 9

Groves, A. (1996 16 Jul). Apparatus and method for debugging electronic components through an in-circuit emulator – US Patent N° 5537536. Assignee Intel, 20th Dec, 1995. Available at: `www.freepatentsonline.com/5537536.html` 65

Guo, J.R., You, C., Zhou, K., Goda, B.S., Kraft, R.P. and McDonald, J.F. (2003). A scalable 2 V, 20 GHz FPGA using SiGe HBT BiCMOS technology. In: *Proceedings of the 2003 ACM/SIGDA eleventh International Symposium on Field programmable gate arrays*, pp. 145–153. ACM Press. ISBN 1-58113-651-X. 99

Hailpern, B. and Santhanam, P. (2002). Software debugging, testing, and verification. *IBM Systems Journal*, vol. 41, no. 1, pp. 4–12. 3

Halfhill, T.R. (2004 May, 31). Tensilica Tackles Bottlenecks. New Xtensa LX Configurable Processor Shatters Industry Benchmarks. *In-Stat/MDR Microprocessor Report*. Available at: `www.tensilica.com/MDR_LX.pdf` 99

Hammond, L., Hubbert, B.A., Siu, M., Prabhu, M.K., Chen, M. and Olukotun, K. (2000 Mar-Apr). The Stanford Hydra CMP. *IEEE Micro*, vol. 20, no. 2, pp. 71–84. 4

Hansen, L. (2004). Xilinx 6.2i Design Tools. *Xcell Journal*, vol. 49, no. Summer, pp. 50–51. 68

Hansen, L. and Przybus, B. (2005). Real-Time Debug That Dominates. *Xcell Journal*, vol. 53, no. Second Quarter, pp. 70–72. 68

Hauser, J.R. (2000). *Augmenting a Microprocessor with Reconfigurable Hardware*. Ph.D. thesis, Computer Science, University of California, Berkeley. Available at: `http://brass.cs.berkeley.edu/documents/AugmentingProcWithReconfigHardware.pdf` 5

Hauser, J.R. and Wawrzynek, J. (1997). Garp: A MIPS Processor with a Reconfigurable Coprocessor. In: Pocek, K.L. and Arnold, J. (eds.), *IEEE Symposium on FPGAs for Custom Computing Machines*, pp. 12–21. IEEE Computer Society Press, Los Alamitos, CA. Available at: `citeseer.nj.nec.com/hauser97garp.html` 5

Heape, J.E. and Stollon, N. (2004). Instrumentation-Based Analysis of System FPGAs. DesignCon. 69

Heinrich, M., Ofelt, D., Horowitz, M. and Hennessy, J. (1997 Mar). Hardware/Software Codesign of the Stanford FLASH Multiprocessor. In: *Proceedings of the IEEE Special Issue on Hardware/Software Co-design*, vol. 85. Available at: `citeseer.nj.nec.com/heinrich97hardwaresoftware.html` 28

Heisch, R.R. (1998 30 Jun). System and method for acquiring high granularity performance data in a computer system – US Patent N° 5774724. Assignee IBM, filed 20th Nov, 1995. Available at: `www.freepatentsonline.com/5774724.html` 17

Hennessy, J.L. and Patterson, D.A. (1990). *Computer Architecture. A Quantitative Approach*. Morgan Kaufmann Publishers. ISBN 1-55860-069-8. 4

Heo, S. (2000 Aug). *A Low-Power 32-bit Datapath Design*. Master's thesis, Electrical Engineering and Computer Science, MIT. 5

Hewlett Packard (1994 Feb). HP B3740A Software Analyzer. Doc. N° 5962-7114E. 66

Hill, M.D. (1987 Nov). *Aspects of Cache Memory and Instruction Buffer Performance*. Ph.D. thesis, University of California, Berkeley. Available as Report No: UCB/CSD/ 87/381. 34

Hohl, W., Circello, J. and Riedel, K. (date unknown (probably 1995)). Debug Support on the ColdFire Architecture.
Available at: `www.freescale.com` 71

Horowitz, M., Martonosi, M., Mowry, T.C. and Smith, M.D. (1998 May). Informing Memory Operations: Memory Performance Feedback Mechanisms and Their Applications. *ACM Transactions on Computer Systems*, vol. 16, no. 2, pp. 170–205. 50

Hosseini, A., Mavroidis, D. and Konas, P. (1996). Code Generation and Analysis for the Functional Verification of Microprocessors. In: *DAC '96: 33rd Annual Conference on Design Automation*, pp. 305–310. ACM, New York, NY, USA. ISBN 0-89791-779-0. 31

Hunt, W.A. and Sawada, J. (1999 May-Jun). Verifying the FM9801 Microarchitecture. *IEEE Micro*, vol. 19, no. 3, pp. 47–55. 25

Hur, Y., Bae, Y., Lim, S., Kim, S.-K., Rhee, B.-D., Min, S., Park, C., Shin, H. and Kim, C. (1995 Dec). Worst case analysis of RISC processors: R3000/R3010 case study. In: *Proceedings of the 16th Real-Time Systems Symposium*, pp. 308–321.
Available at: `archi.snu.ac.kr/PUBLICATIONS/papers/real-time/An--hur-rtss95.ps` 80

IBM (1991 Dec). Tailorable Embedded Event Trace. *IBM Technical Disclosure Bulletin*, vol. 34, no. 7B, pp. 259–261. Serial number TDB1291.0092 the Software Patent Institute Database of Software Technologies. 66, 84

IBM Microelectronics (1998 10 Apr). *Real-Time Instruction Trace in the PowerPC 400 Family of Embedded Controllers*. PowerPC Embedded Processors Application Note.
Available at: `www.chips.ibm.com/products/powerpc/library/traceapp.pdf` 66, 70, 85

IBM Microelectronics (1999 12 Jul). *IBM PowerPC 400 Family Instruction Tracing Versus Traditional In-Circuit Emulators*, 1st edn. PowerPC Embedded Processors Application Note.
Available at: `www.chips.ibm.com/products/powerpc/library/trace_ice.pdf` 66, 70, 85

IBM Microelectronics (2004 Apr, 13). Applied Micro Circuits Corporation announces definitive agreement to acquire intellectual property and a portfolio of PowerPC 400 products from IBM, signs Power Architecture license.
Available at: `www-3.ibm.com/chips/news/2004/0413_power.html` 37, 71

IBM Press Release (2004 2 Dec). Momentum – For Power Architecture Technology. 4

Jin, W., Sun, X. and Chase, J.S. (2001 Apr). FastSlim: Prefetch-Safe Trace Reduction for I/O Cache Simulation. *ACM Transactions on Modeling and Computer Simulation*, vol. 11, no. 2, pp. 135–160. 24

John Jr., C.C. and Urquhart, R.J. (1999 17 Aug). System and method for tracing instructions in an information handling system without changing the system source code – US Patent N° 5938778. Assignee IBM, filed 10^{th} Nov, 1997.
Available at: `www.patentstorm.us/patents/5938778.html` 9

Johnson, E.E. (1999). PDATS II: Improved Compression of Address Traces. In: *Proceedings 18th IEEE International Performance, Computing, and Communications Conference*.
Available at: `http://tracebase.nmsu.edu/tracebase.html` 85, 93

Johnson, E.E. and Ha, J. (1994). PDATS Lossless Address Trace Compression For Reducing File Size And Access Time. In: *Proceedings IEEE International Phoenix Conference on Computers and Communications*.
Available at: `http://tracebase.nmsu.edu/tracebase.html` 85, 93

Jotwani, R. (2000 14 Nov). Single-port trace buffer architecture with overflow reduction – patent 6148381. Filed 20th Jan, 1998.
Available at: `http://www.freepatentsonline.com/6148381.html` 84

Karlin, S.C., Clark, D.W. and Martonosi, M. (1999 Mar). SurfBoard – A Hardware Performance Monitor for SHRIMP. Tech. Rep. TR-596-99, Princeton University.
Available at: `citeseer.nj.nec.com/karlin99surfboard.html` 77, 78

Keller, T.W. and Urquhart, R.J. (1994 11 Oct). Non-invasive trace-driven system and method for computer system profiling – US Patent N° 5 355 487. Assignee IBM, filed 23rd July, 1993.
Available at: `www.patentstorm.us/patents/5355487-description.html` 9

Kivett, T. (2008 14 Jan). IFI Patent Intelligence Announces 2007's Top U.S. Patent Asignees.
Available at: `www.ificlaims.com/IFIPatentRelease1-9-08.htm` 85

Krashinsky, R. (2001 May). *Microprocessor Energy Characterization and Optimization through Fast, Accurate, and Flexible Simulation*. Master's thesis, Electrical Engineering and Computer Science, MIT. 5

Krishnan, S. (2006 12 Dec). Variable Accuracy Mode in Microprocessor Simulation – US Patent n° 7 149 676 B2. Assignee Renesas Technology, filed 21st June, 2001. 27, 40, 41

Lango, J., Adams, K., Castelle, M. and Powell, D. (1998). The Brown Simulator (version 2). Brown University. 45

Larus, J.R. (1991 Dec). SPIM S20: A MIPS R2000 Simulator. Computer Science Dept, University of Wisconsin-Madison.
Available at: `www.cs.wisc.edu/$\sim$larus/spim.html` 29

Larus, J.R. (1993 May). Efficient Program Tracing. *IEEE Computer*, vol. 26, no. 5, pp. 52–61. ISSN 0018-9162. 25, 53, 54, 59, 90

Larus, J.R. and Ball, T. (1992 25 Mar). Rewriting Executable Files to Measure Program Behavior. Tech. Rep. CS-TR-92-1083, University of Wisconsin-Madison, Madison, WI, USA.
Available at: `citeseer.ist.psu.edu/larus94rewriting.html` 53, 110

Larus, J.R. and Ball, T. (1994 Feb). Rewriting Executable Files to Measure Program Behavior. *Software–Practice and Experience*, vol. 24, no. 2, pp. 197–218. Also available as Larus and Ball (1992). 53, 54

Lattice Semiconductor Corp. (2007 May). Programming and Logic Analysis Tutorial. 73

Lauterbach, G. (1993 Dec). Accelerating Architectural Simulation by Parallel Execution of Trace Samples. Tech. Rep. SMLI TR-93-22, Sun Microsystems Laboratories, Inc. 4

Leatherman, R. (2000). On-Chip Instrumentation Approach to System-On-Chip Development. White paper.
Available at: `www.fs2.com/pdfs/OCI_Whitepaper.pdf` 69

Lee, S., Lyul, M.L.S., Chong, M. and Kim, S. (1998). TimerMon: A Time-Tracing Hardware for Instrumenting Real-Time Software. In: *Proceedings of IEEE Real-Time Computing Systems and Applications Symposium*.
Available at: `http://citeseer.nj.nec.com/lee98timermon.html` 80

Lenoski, D., Laudon, J., Joe, T., Nakahira, D., Stevens, L., Gupta, A. and Hennessy, J. (1993). The DASH Prototype: Logic Overhead and Performance. *IEEE Transactions on Parallel & Distributed Systems*, vol. 4, no. 1, pp. 41–61.
Available at: `citeseer.nj.nec.com/lenoski93dash.html` 80

Levine, F. and Roth, C. (1997). A programmer's view of performance monitoring in the PowerPC microprocessor. *IBM Journal of Research and Development*, vol. 41, no. 3, pp. 331–344.
Available at: `www.research.ibm.com/journal/rd/413/levine.html` 14, 15

Lindenmaier, G., McKinley, K.S. and Temam, O. (2000 29 Aug – 1 Sep). Load Scheduling with Profile Information. In: *Euro-Par 2000, Parallel Processing, 6th International Euro-Par Conference*, vol. 1900 of *Lecture Notes in Computer Science*, pp. 223–233. Springer, Munich, Germany. ISBN 3-540-67956-1.
Available at: `citeseer.ist.psu.edu/lindenmaier00load.html` 10

Lo, J.L., Barroso, L.A., Eggers, S.J., Gharachorloo, K., Levy, H.M. and Parekh, S.S. (1998 Jun). An Analysis of Database Workload Performance on Simultaneous Multithreaded Processors. In: *Proceedings of the 25th Annual International Symposium on Computer Architecture*. Smtdatabase.pdf from University of Washington. 49

Luk, C.-K., Cohn, R., Muth, R., Patil, H., Klauser, A., Lowney, G., Wallace, S., Reddi, V.J. and Hazelwood, K. (2005). Pin: Building Customized Program Analysis Tools with Dynamic Instrumentation. In: *PLDI '05: Proceedings of the 2005 ACM SIGPLAN conference on Programming language design and implementation*, pp. 190–200. ACM Press, New York, NY, USA. ISBN 1-59593-056-6. 55

Maemura, K. (1995 3 Oct). Debugger operable with only background monitor – US Patent N° 5 455 936. Assignee NEC, filed 28th Apr, 1994.
Available at: `www.freepatentsonline.com/US5455936.html` 22

Magnusson, P.S., Christensson, M., Eskilson, J., Forsgren, D., Hallberg, G., Hogberg, J., Larsson, F., Moestedt, A. and Werner, B. (2002). Simics: A Full System Simulation Platform. *Computer*, vol. 35, no. 2, pp. 50–58. ISSN 0018-9162. 45, 46

Mann, D. (2001 16 Jan). Processor including a combined parallel debug and trace port and a serial port – US Patent N° 6175914. Assignee AMD, filed 17th Dec, 1997.
Available at: `http://www.patentstorm.us/patents/6175914-claims.html` 85, 90

Marshall, J. (1999). New Architectures for Solving HW/SW Integration Problems. *Hewlett-Packard Insight Magazine*, vol. 4, no. 4.
Available at: `www.agilent.com` 67

Marshall, J. (2000). Debug and Integration of Complex Embedded Systems: Improving Hardware and Software Debug of Digital Systems Symposium. *Hewlett-Packard Insight Magazine*. 67

Martonosi, M., Clark, D.W. and Mesarina, M. (1996 May*a*). The SHRIMP Performance Monitor: Design and Applications. In: *ACM SIGMETRICS Symposium on Parallel and Distributed Tools*.
Available at: `citeseer.nj.nec.com/martonosi96shrimp.html` 76, 77

Martonosi, M., Gupta, A. and Anderson, T. (1993). Effectiveness of Trace Sampling for Performance Debugging Tools. In: *SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, pp. 248–259. ACM. Citeseer.nj.nec.com/515615.html. 56

Martonosi, M., Ofelt, D. and Heinrich, M. (1996*b*). Integrating Performance Monitoring and Communication in Parallel Computers. In: *Measurement and Modeling of Computer Systems*, pp. 138–147.
Available at: `citeseer.nj.nec.com/martonosi96integrating.html` 29

Martorell, X., Smeds, N., Walkup, R., Brunheroto, J.R., Almásian, G., Gunnels, J.A., DeRose, L., Labarta, J., Escalé, F., , Giménez, J., Servat, H., and Moreira, J.E. (2005 Mar/May). Blue Gene/L performance tools. *IBM Journal of Research and Development*, vol. 49, no. 2/3, pp. 407–424. 17, 36

Mauer, C.J., Hill, M.D. and Wood, D.A. (2002). Full-system timing-first simulation. In: *SIGMETRICS '02: Proceedings of the 2002 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, pp. 108–116. ACM Press, New York, NY, USA. ISBN 1-58113-531-9. 23, 26, 34

Mazières, D. and Smith, M.D. (1994 Nov). Abstract Execution in a Multi-Tasking Environment. Tech. Rep. TR-32-94, Harvard University. (Senior thesis), available from http://eecs.harvard.edu/hube/publications/ mazieres_thesis.ps. 24, 54

McLear, R.E., Scheibelhut, D.M. and Tammaru, E. (1982). Guidelines for creating a debuggable processor. In: *ASPLOS-I: Proceedings of the first international symposium on Architectural support for programming languages and operating systems*, pp. 100–106. ACM Press. ISBN 0-89791-066-4. 74

McMinn, B.D. and Ganapathy, G. (1998 24 Feb). Simulation by emulating level sensitive latches with edge trigger latches – US Patent N° 5721695. Assignee AMD, filed 17th Oct, 1994.
Available at: `http://www.patentstorm.us/patents/5721695-description.html` 42

Mercury Computer Systems (2001). RACE++ Series. TATL Trace Analysis Tool and Library. http://www.mc.com. 51

Milenkovic, A. and Milenkovic, M. (2003 Sep). Stream-Based Trace Compression. *Computer Architecture Letters*, vol. 2.
Available at: `www.cs.virginia.edu/$\sim$tcca/2003paps.html` 85, 94

MIPS Technologies Inc. (2002 21 Mar*a*). *EJTAG Trace Control Block Specification*. Document N° MD00148 Rev: 1.04.
Available at: `www.mips.com` 69

MIPS Technologies Inc. (2002 21 Mar*b*). *PDtrace Interface Specification*. Document N° MD00136 Rev: 2.07.
Available at: `www.mips.com` 69

Mogul, J.C. and Ramakrishnan, K. (1997 Aug). Eliminating receive livelock in an interrupt-driven kernel. *ACM Transactions on Computer Systems*, vol. 15, no. 3, pp. 217–252.
Available at: `www.acm.org/pubs/citatons/journals/tocs/1997-15-3/p217-mogul` 49

Moreno, J., Moudgill, M., Ebcioglu, K., Altman, E., Hall, C., Marinda, R., Chen, S.-K. and Polyak, A. (1997). Simulation/evaluation environment for a VLIW processor architecture. *IBM Journal of Research and Development*, vol. 41, no. 3, pp. 287–302.
Available at: `www.research.ibm.com/journal/rd/413/moreno.html` 35

Motorola (1997 Oct). Motorola Version Three ColdFire Processor. White Paper from www.freescale.com. 71

Motorola Inc. (1997). *PowerPC Microprocessor Family: The Programming Environments For 32-Bit Microprocessors*. Part N°: MPCFPE32B/AD 1/97 Rev 1. 18

Moudgill, M. (1998 Apr). An Overview of the LeProf Profiling Tool. Research Report RC21169, IBM.
Available at: `www.research.ibm.com/MET/Publications/leprof.pdf` 16

Moudgill, M., Wellman, J.-D. and Moreno, J.H. (1999 May-Jun). Environment for PowerPC Microarchitecture Exploration. *IEEE Micro*, vol. 19, no. 3, pp. 15–25. 35

Murphy, J., Conlon, M., O'Keeffe, H., O'Loughlin, K., Carney, L., Richardson, B., Nolan, M., Cosgrove, R., Noonan, P., Hannon, M., Conlon, J., Burke, C., Garvey, T., Blackburn, P., Nicholls, D., Barry, D., O'Meara, P., O'Neill, E., Hall, D., Lane, B., McGourty, O., Beekman, J., Healy, M., Hickey, A. and Lavin, P. (2003 3 Dec). EP1367489 Ashling European Software Patent – A Microprocessor Development System. Filed 31st May, 2002.
Available at: `http://gauss.ffii.org/PatentView/EP1367489` 65, 66

Nagle, D., Uhlig, R. and Mudge, T. (1992 6 May). Monster: A Tool for Analyzing the Interaction Between Operating Systems and Computer Architectures. Tech report, University of Michigan.
Available at: `www.eecs.umich.edu/UMichMP/Publications/monster.ps` 24, 78

Narayanasamy, S., Pokam, G. and Calder, B. (2006 Jan/Feb). BugNet: Recording Application-Level Execution for Deterministic Replay Debugging. *IEEE Micro*, vol. 26, no. 1, pp. 100–109. 3

Netzer, R.H. (1993). Optimal tracing and replay for debugging shared-memory parallel programs. In: *Proceedings of the 1993 ACM/ONR workshop on Parallel and distributed debugging*. ACM Press, San Diego, California. 58, 59

Nielsen, M.R., Conquet, E. and Terraillon, J.-L. (2003 18 June). The European Space Agency's involvement and interest in WCET and scheduling analysis. In: *WCET'2002 2nd International Workshop on Worst-Case Execution Time Analysis (Satellite Event to ECRTS'02)*. Vienna, Austria. 3

Njoroge, N. (2004 15 Sep). Statistical Profiler for Embedded IBM PowerPC. Xilinx Application Note XAPP545. 15, 72

O'Keeffe, H. (2006). Embedded Debugging: A White Paper. Ashling Microsystems Ltd. 71

Olsen, D.W. (2004). Think Outside the Chip. *Xcell Journal*, vol. 48, no. Spring, pp. 75 – 78. 68

Orme, W. (probably 2006). How CoreSight Technology Gets Higher Performance, More Reliable Product to Market Quicker. 72

Peterson, J., Bohrer, P., Chen, L., Elnozahy, E., Gheith, A., Jewell, R., Kistler, M., Maeurer, T., Malone, S., Murrell, D., Needel, N., Rajamani, K., Rinaldi, M., Simpson, R., Sudeep, K. and Zhang, L. (2006). Application of full-system simulation in exploratory system design and development. *IBM Journal of Research and Development*, vol. 50, no. 2/3, pp. 321–332.
Available at: `http://www.research.ibm.com/journal/rd/502/peterson.html` 36

Pierce, J., Smith, M.D. and Mudge, T. (1995). *Instrumentation Tools*. Fast Simulation of Computer Architectures. Kluwer Academic Publishers, Boston, MA.
Available at: `http://eecs.harvard.edu/hube/publications/kluwer95.ps` 24, 54, 55, 56, 57, 62

Pinkerton, T.B. (1969 Nov). Performance Monitoring in a Time-Sharing System. *Communications of the ACM*, vol. 12, no. 11, pp. 608–610. 24, 97

Pollack, A. (1990 29 Aug). A Chip Patent Is Granted That May Rewrite History. The New York Times. Section D, pg 1, column 1, Financial Desk. 84

Popescu, V. and McNamara, B. (1996). Innovative Verification Strategy Reduces Design Cycle Time for High-end SPARC Processor. In: *DAC '96: Proceedings of the 33rd annual conference on Design Automation*, pp. 311–314. ACM, New York, NY, USA. ISBN 0-89791-779-0. 33

Poret, M. and Mckinley, J. (1987 16 Jun). In-circuit emulator – US Patent N° 4674089. Assignee Intel, filed 16th Apr, 1985.
Available at: `www.freepatentsonline.com/4674089.html` 65, 85

Redstone, J.A., Eggers, S.J. and Levy, H.M. (2000 Nov). An Analysis of Operating System Behavior on a Simultaneous Multithreaded Architecture. In: *Proceedings of the 9th International Conference on Architectural Support for Programming Languages and Operating Systems*. Washington University.
Available at: `www.cs.washington.edu/research/smt/papers/os.pdf` 28

Reilly, M. and Edmondson, J. (1998). Performance Simulation of an Alpha Microprocessor. *Computer*, vol. 31, no. 5, pp. 50–58. ISSN 0018-9162. 38

Reinhardt, S.K., Hill, M.D., Larus, J.R., Lebeck, A.R., Lewis, J.C. and Wood, D.A. (1993 Jun). The Wisconsin Wind Tunnel: virtual prototyping of parallel computers. *ACM SIGMETRICS Performance Evaluation Review*, vol. 21, no. 1, pp. 48–60. 34

Rich, A.W. and Edwards, D.A. (2005 12 July). Method for compressing and decompressing trace information – US Patent N° 6918065. Hitachi, filed 1st Oct, 1999.
Available at: `http://www.freepatentsonline.com/6918065.html` 85

Ristelhueber, R. (2001 14 June). Updated: Lextra wins skirmish in MIPS patent case. EBN News.
Available at: `www.ebnews.com/digest/story/OEG20010614S0034` 27, 83

Rose, C.D. and Flanagan, J.K. (). Constructing Instruction Traces from Cache-filtered Address Traces (CITCAT).
Available at: `http://pel.cs.byu.edu/techreports/CITCAT_ACM_article.pdf` 80

Rosenblum, M., Bugnion, E., Devine, S. and Herrod, S.A. (1997). Using the SimOS Machine Simulator to Study Complex Computer Systems. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, vol. 7, no. 1, pp. 78–103.
Available at: `citeseer.nj.nec.com/rosenblum97using.html` 27, 28

Rosenblum, M., Bugnion, E., Herrod, S., Witchel, W. and Gupta, A. (1995 Dec). The impact of architectural trends on operating system performance. *ACM SIGOPS Operating Systems Review*, vol. 29, no. 5, pp. 285–298. 28

Rosenblum *et al.* (). SimOS: The Complete Machine Simulator. Http://simos.stanford.edu.
Available at: `http://simos.stanford.edu` 28

Rosqvist, L. (2001). New Strategies for Verification of SoC Designs. *Agilent Application Resource Central*, vol. 6, no. 1. 67

Samples, A.D. (1989). Mache: No-Loss Trace Compaction. In: *SIGMETRICS '89: Proceedings of the 1989 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, pp. 89–97. ACM Press, New York, NY, USA. ISBN 0-89791-315-9. 93

Sandon, P., Liao, Y.-C., Cook, T., Schultz, D. and de Nicolas, P.M. (1997). NStrace: A bus-driven instruction trace tool for PowerPC microprocessors. *IBM Journal of Research and Development*, vol. 41, no. 3, pp. 331–344.
Available at: `www.research.ibm.com/journal/rd/413/sandon.html` 27, 78, 85

Schieber, C.D. and Johnson, E.E. (1994 Apr). RATCHET: Real-time Address Trace Compression Hardware for Extended Traces. *Performance Evaluation Review*, vol. 21, no. 3, pp. 22–29. 95

Schmidt, W., Roediger, R., Mestad, C., Mendelson, B., Shavit-Lottem, I. and Bortnikov-Sitnitsky, V. (1998). Profile-directed restructuring of operating system code. *IBM Systems Journal*, vol. 37, no. 2, pp. 270–297. 16

Shankland, S. (2006 14^{th} Feb). IBM chip architect guns for gigahertz.
Available at: `www.news.com` 99

Shannon, L. and Chow, P. (2004). Using Reconfigurability to Achieve Real-Time Profiling for Hardware/Software Codesign. In: *Proceeding of the 2004 ACM/SIGDA 12^{th} International Symposium on Field Programmable Gate Arrays*, pp. 190–199. ACM Press, Monterey, California. ISBN 1-58113-829-6. 21

Sherman, S., Baskett, F. and Brown, J. (1972 Dec). Trace-Driven Modeling and Analysis of CPU Scheduling in a Multiprogramming System. *Communications of the ACM*, vol. 15, no. 12, pp. 1063–1069. 24

Shobaki, M.E. (1998 Aug). Verification of Embedded Real-Time Systems Using Hardware/Software Co-simulation. In: *24th Euromicro Conference Proceedings*, Vol I, pp. 46–50. IEEE, Västerås, Sweden.
Available at: `www.mrtc.mdh.se/publications/0109.pdf` 80

Shobaki, M.E. (1999 Aug). Observability in Multiprocessor Real-Time Systems with Hardware/Software Co-Simulation. In: *Swedish National Real-Time Conference SNART'99*. Linköping, Sweden.
Available at: `www.mrtc.mdh.se/publications/0111.pdf` 80

Shobaki, M.E. (2002 Mar). On-Chip Monitoring of Single- and Multiprocessor Hardware Real-Time Operating Systems. In: *Proceedings of the 8th International Conference on Real-Time Computing Systems and Applications (RTCSA)*. IEEE.
Available at: `citeseer.nj.nec.com/502484.html` 80

Shobaki, M.E. and Lindh, L. (2001 Mar). A Hardware and Software Monitor for High-Level Sytem-on-Chip Verification. In: *Proceedings of the IEEE International Symposium on Quality Electronic Design*. San Jose, CA, USA.
Available at: `citeseer.nj.nec.com/379790.html` 80

Simply RISC (2006 8 Sep). Simply RISC ships the S1 Core.
Available at: `www.srisc.com` 32

Sirer, E.G. (1993 Jun). Measuring Limits of Fine-grained Parallelism. Senior undergraduate thesis, Princeton University. Found using google - could not find at Princeton, or Cornell. Available at: `www.cs.washington.edu/homes/egs/mipsi/papers/mipsi.ps` 30

Sirer, E.G. (1997?). MIPSI – MIPS Simulator. Computer Science Dept, Washington University. Available at: `www.cs.washington.edu/homes/egs/mipsi/mipsi.html` 29

Sites, R.L. and Agarwal, A. (1988 30 May– 2 Jun). Multiprocessor Cache Analysis Using ATUM. In: *ISCA '88: Proceedings of the 15th Annual International Symposium on Computer architecture*, pp. 186–195. IEEE Computer Society Press, Los Alamitos, CA, USA. ISBN 0-8186-0861-1. 61, 62

Smith, M.D. (1991 Nov). Tracing with *pixie*. Tech. Rep. CSL-TR-91-494, Stanford University. 50, 51, 92

Sprunt, B. (2002 Jul/Aug*a*). The Basics of Performance-Monitoring Hardware. *IEEE Micro*, vol. 22, no. 4, pp. 64–71. ISSN 0272-1732. 18, 20

Sprunt, B. (2002 Jul/Aug*b*). Pentium 4 Performance-Monitoring Features. *IEEE Micro*, vol. 22, no. 4, pp. 72–83. ISSN 0272-1732. 14

Srivastava, A. and Eustace, A. (1994 Mar). ATOM: A System for Building Customized Program Analysis Tools. Tech. Rep. 94/2, Digital Western Research Laboratory. Available at: `www.hpl.hp.com/techreports/Compaq-DEC/WRL-94-2.html` 47, 48, 49

Stankovic, J.A., Ramamritham, K., Niehaus, D., Humphrey, M. and Wallace, G. (1999 May). The spring system: Integrated support for complex real-time systems. *Real-Time Systems*, vol. 16, no. 2-3, pp. 223–251. Article ID: 204029. 34

Steffora, A. (1999 8 Mar). HP and ARM team up for ARM CPU trace tools – Company Business and Marketing. Electronic News. Found via Google. 66, 71

Stewart, D.A. and Gentleman, W. (1997 17-18 May). Non-Stop Monitoring and Debugging on Shared-Memory Multiprocessors. In: *Proceedings of the IEEE 2nd International Workshop on Software Engineering for Parallel and Distributed Systems*, pp. 263–269. National Research Council of Canada, Boston, MA. Also available as NRC N° 40147. Available at: `http://wwwsel.iit.nrc.ca/seldocs/dbjdocs/NRC40147.pdf` 25, 81

Stollon, N., Uvacek, B. and Laurenti, G. (2007 26 Mar). Standard Debug Interface Socket Requirements For OCP-Compliant SoC. Available at: `www.ocpip.org/socket/whitepapers/OCP-IP_Debug_Working_Group_Whitepaper_3_26_2007.pdf` 75

Stunkel, C. and Fuchs, W. (1992). An Analysis of Cache Performance for a Hypercube Multicomputer. *IEEE Transactions on Parallel and Distributed Systems*, vol. 03, no. 4, pp. 421–432. ISSN 1045-9219. 52, 53

Stunkel, C.B. and Fuchs, W.K. (1989 23-26 May). TRAPEDS: Producing Traces for Multicomputers via Execution Driven Simulation. In: *SIGMETRICS '89: Proceedings of the 1989 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, pp. 70–78. ACM Press, New York, NY, USA. ISBN 0-89791-315-9. 52, 53

Swartley, C. (2003 21 Nov). FS^2 core size. Personal communication – email. 69

Tamches, A. and Miller, B.P. (1999). Fine-Grained Dynamic Instrumentation of Commodity Operating System Kernels. *USENIX Third Symposium on Operating Systems Design and Implementation (OSDI)*, pp. 117–130. 52

Taylor, M.B., Kim, J., Miller, J., Wentzlaff, D., Ghodrat, F., Greenwald, B., Hoffman, H., Johnson, P., Lee, J.-W., Lee, W., Ma, A., Saraf, A., Seneski, M., Shnidman, N., Strumpen, V., Frank, M., Amarasinghe, S. and Agarwal, A. (2002 Mar/Apr). The RAW Microprocessor: A Computational Fabric for Software Circuits and General Purpose Programs. *IEEE Micro*, vol. 22, no. 2, pp. 25–35. ISSN 0272-1732. 5

Tektronics, Inc. (2002 8 Feb). PowerPC Debug: Integrated Solutions for Hardware and Software Challenges. Parts I & II. Application Note, part number 52W-15469-0. 68

Tektronics, Inc. (2006). Tektronix and FS2 Answer Debug Challenges for Xilinx FPGAs. Available at: `www.tektronix.com/fpga` 68

Tensilica (2004 Aug 2). Tensilica Technology Helps Power World's Fastest Router. Available at: `www.tensilica.com/html/pr_2004_08_02.html` 99

The TORCHers (1994 25 Aug). TORCH Architectural Specification. 30

The Wave Report (1999 5 April). HP and ARM jointly develop real-time execution trace tools for ARM CPUs.
Available at: `www.wave-report.com/1999_Wave_Issues/wave9035.html` 71

Thekkath, R., Treue, F., Edgar, E.L. and Leatherman, R.T. (2007 7 Nov). External trace synchronization via periodic sampling – US Patent n° 7134116. Assignee MIPS, filed 30th April, 2001. 69

Theurich, K., Albus, A., Eickhoff, F., Immel, D., Kohler, A., Lange, E. and von Buttlar, J. (2007). Advanced firmware verification using a code simulator for the IBM System z9. *IBM Journal of Research and Development*, vol. 51, no. 1-2, pp. 207–216.
Available at: `www.research.ibm.com/journal/rd/511/theurich.pdf` 26

Thornock, N.C. and Flanagan, J.K. (). Using the BACH Trace Collection Mechanism to Characterize the SPEC 2000 Integer Benchmarks.
Available at: `http://pel.cs.byu.edu/techreports/wwc_bach.pdf` 79

Tien, C.-K.V., Lewis, K., Greub, H.J., Tsen, T. and McDonald, J.F. (1997 Jun). Design of a 32 b Monolithic Microprocessor Based on GaAs HMESFET Technology. *IEEE Transactions on Very Large Scale Integration Systems*, vol. 5, no. 2, pp. 238–243.
Available at: `http://inp.cie.rpi.edu/research/mcdonald/frisc/publications&patents/Tien00585228.pdf` 4

Traub, O., StuartSchechter and Smith, M. (2000 Jun). Ephemeral Instrumentation for Lightweight Program Profiling. Unpublished technical report, Dept. EECS Harvard University.
Available at: `http://eecs/harvard.edu/hube/publications/publications.html` 13

Tsai, J.J., Bi, Y., Yang, S.J. and Smith, R.A. (1996). *Distributed Real-Time Systems. Monitoring, Visualization, Debugging and Analysis*. Wiley-Interscience. ISBN 0-471-16007-5. 65

Uhlig, R. (1995). *Trap-driven Memory Simulation*. Ph.D. thesis, Computer Science and Engineering, University of Michigan. 5, 24, 25, 56, 79

Uhlig, R., Fishstein, R., Gershon, O., Hirsh, I. and Wang, H. (1999 Nov). SoftSDV: A Presilicon Software Development Environment for the IA-64 Architecture. *Intel Technology Journal*, vol. 3, no. 4.
Available at: `http://www.intel.com/technology/itj/archive/1999.htm` 40

Uhlig, R., Nagle, D., Mudge, T. and Sechrest, S. (1994). Trap-driven Simulation with Tapeworm II. In: *ASPLOS-VI: Proceedings of the sixth international conference on Architectural support for programming languages and operating systems*, pp. 132–144. ACM Press, New York, NY, USA. ISBN 0-89791-660-3. 56, 57

University of Tennessee (2004 Mar). Performance Application Programming Interface User Guide – Version 3.0.6.
Available at: `http://icl.cs.utk.edu/papi` 18, 19

Veenstra, J.E. and Fowler, R.J. (1994 Jan*a*). MINT: A Front End for Efficient Simulation of Shared-Memory Multiprocessors. In: *Proceedings of the Second International Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, pp. 201–207. 5, 31, 38

Veenstra, J.E. and Fowler, R.J. (1994 Aug*b*). MINT Tutorial and User Manual. Tech. Rep. TR452, Computer Science Dept., University of Rochester.
Available at: `http://citeseer.nj.nec.com/49919.html` 31

Vestal, S. (1994 Apr). Fixed-Priority Sensitivity Analysis for Linear Compute Time Models. *IEEE Transactions on Software Engineering*, vol. 20, no. 4, pp. 308–317. 2

Waingold, E., Taylor, M., Srikrishna, D., Sarker, V., Lee, W., Lee, V., Kim, J., Frank, M., Finch, P., Barua, R., Babb, J., Amarasinghe, S. and Agarwal, A. (1997 Sep). Baring It All to Software: Raw Machines. *IEEE Computer*, vol. 30, no. 9, pp. 86–93. ISSN 0018-9162. 5

Wall, D.W. (1989 Sep). Link-Time Code Modification. Technical Report WRL-89-17, Digital Western Research Laboratory.
Available at: `www.hpl.hp.com/techreports/Compaq-DEC/WRL-89-17.pdf` 9, 51, 58

Wall, D.W. (1992 May). Systems for late code modification. Technical Report WRL-92-3, Digital Western Research Laboratory. Was also published as TN-19 in June 1991.
Available at: `www.hpl.hp.com/techreports/Compaq-DEC/WRL-92-3.pdf` 51, 57, 58

Wang, H., Manor, S., LaFollette, D., Nesher, N. and King, K. (2003 Mar). Inferno: A Functional Simulation Infrastructure for Modeling Microarchitectural Data Speculations. In: *International Symposium on Performance Analysis of Systems and Software*, vol. 0, pp. 11–21. IEEE Computer Society, Los Alamitos, CA, USA. ISBN 0-7803-7756-7. 41

Wikipedia (Accessed April 2008). Microprocessor.
Available at: `http://en.wikipedia.org/wiki/Microprocessor` 84

Wilburn, D. (2004). Nohau Shortens Debugging Time for MicroBlaze and Virtex-II Pro PowerPC Users. *Xilinx Xcell Journal*, vol. 51, no. Winter, pp. 23–25. 73

WindRiver Systems (1999 Mar). *CodeTEST for Tornado User's Guide*. 1.1. 19, 20

Witchel, E. and Rosenblum, M. (1996 Jan). Embra: Fast and Flexible Machine Simulation. In: *Conference on Measurement and Modeling of Computer Systems*, pp. 68–79. ACM SIGMETRICS, Philadelphia.
Available at: `//citeseer.nj.nec.com/witchel96embra.html` 28

Wittig, R.D. (1995). *OneChip; An FPGA Processor with Reconfigurable Logic*. Master's thesis, Electrical and Computer Engineering, University of Toronto.
Available at: `http://citeseer.ist.psu.edu/wittig95onechip.html` 4

Woodward, J. (2004). The FPGA Dynamic Probe. *Xcell Journal*, vol. 49, no. Summer, pp. 47 – 49. 68

Wu, C.E., Bolmarcich, A., Snir, M., Wootton, D., Parpia, F., Chan, A., Lusk, E. and Gropp, W. (2000). From Trace Generation to Visualization: A Performance Framework for Distributed Parallel Systems. In: *Supercomputing '00: Proceedings of the 2000 ACM/IEEE Conference on Supercomputing (CDROM)*, p. 50. IEEE Computer Society, Washington, DC, USA. ISBN 0-7803-9802-5. 46, 95

Yano, T. and Miyamori, T. (1999 24 Aug). Microprocessor, method for transmitting signals between the microprocessor and debugging tools, and method for tracing – US Patent N° 5943498. Assignee Hewlett-Packard Company, filed 28th Dec, 1995.
Available at: `www.patentstorm.us/patents/5943498-description.html` 67

Young, C. and Smith, M.D. (1996 Jan). Branch Instrumentation in SUIF. In: *Proceedings. First SUIF Compiler Workshop*, pp. 139–145. Division of Applied Sciences, Harvard University, Stanford, CA.
Available at: `http://eecs.harvard.edu/hube/publications/suif96-brprof.ps` 49

Yung, R. (1998 Jun). *Evaluation of a Commercial Microprocessor*. Ph.D. thesis, Computer Science, University of California, Berkeley. 4

Zagha, M., Larson, B., Turner, S. and Itzkowitz, M. (1996 Nov). Performance Analysis Using the MIPS R10000 Performance Counters. In: *Proceedings of Supercomputing*.
Available at: `citeseer.nj.nec.com/zagha96performance.html` 5, 13

Zhang, X., Wang, Z., Gloy, N., Chen, J.B. and Smith, M.D. (1997). System Support for Automatic Profiling and Optimization. In: *SOSP '97: Proceedings of the 16th ACM Symposium on Operating Systems Principles*, pp. 15–26. ACM Press, New York, NY, USA. ISBN 0-89791-916-5. 11, 12

Zhou, P., Qin, F., Liu, W., Zhou, Y. and Torrellas, J. (2004). iWatcher: Efficient Architectural Support for Software Debugging. *SIGARCH Comput. Archit. News*, vol. 32, no. 2, pp. 224–239. ISSN 0163-5964. 43

Zilles, C. and Sohi, G. (2001 Jan). A Programmable Co-processor for Profiling. In: *Proceedings of the 7th International Symposium on High Performance Computer Architecture (HPCA-7)*.
Available at: `citeseer.nj.nec.com/zilles01programmable.html` 13

Index

List of Acronyms

ACM ..... Association for Computing Machinery

AIX ..... Advanced Interactive Executive IBM's version of UNIX.

ALU ..... Arithmetic Logic Unit

AMD ..... Advanced Micro Devices

API ..... Application Program Interface

ARM ..... Advanced RISC Machines

ASCII ..... American National Standard Code for Information Interchange

ASIC ..... Application Specific Integrated Circuit

ATA ..... Advanced Technology Attachment

ATOM ..... Analysis Tools with OM

`ATUM` ..... Address Tracing Using Microcode

BACH ..... BYU Address Collection Hardware See also BYU

BDM ..... Background Debug Mode Processor debug interface from Motorola.

BGA ..... Ball grid array

BIOS ..... Basic Input Output System

BSP ..... Board Support Package

BYU ..... Brigham Young University

CAM ..... Content Addressable Memory

CASE ..... Computer Aided Software Engineering

CDC ..... Control Data Corporation

CFG ..... control-flow graph

CISC ..... Complex Instruction Set Computer vs RISC

CMU . . . . . Carnegie Mellon University

CPI . . . . . Cycles Per Instruction

CPU . . . . . Central Processing Unit

DCPI . . . . . Digital's Continuous Profiling Infrastructure

DDR . . . . . Double Data Rate

DEC . . . . . Digital Equipment Corporation

DIMM . . . . . Dual In-line Memory Module

DMA . . . . . Direct Memory Access

DRAM . . . . . Dynamic RAM Needs refreshing (RAM).

DSP . . . . . Digital Signal Processor

EDA . . . . . Electronic Design Automation

EDK . . . . . Embedded Development Kit

ELF . . . . . Executable and Linkable Format

EPP . . . . . Enhanced Parallel Port

ETM . . . . . Embedded Trace Macro

FIFO . . . . . First-in-first-out

FPGA . . . . . Field Programmable Gate Array

FPU . . . . . Floating-Point Unit

FSB . . . . . Front-side bus

FSDB . . . . . Fast Signal Database

GCC . . . . . GNU compiler collection

GDB . . . . . GNU debugger

GHS . . . . . Green Hills Software

GNU . . . . . GNU is Not Unix from the Free Software Foundation

GUI . . . . . Graphical User Interface

HDL . . . . . Hardware Description Language

HP . . . . . Hewlett Packard

HTML . . . . . HyperText Markup Language

IBM . . . . . International Business Machines

IC . . . . . Integrated Circuit

ICE ..... In-Circuit Emulation

IDE ..... Integrated Development Environment

IDT ..... Integrated Device Technology, Inc.

IEEE ..... Institute of Electrical and Electronic Engineers

I/O ..... input/ output

IP ..... Intellectual Property

IRQ ..... Interrupt Request

ISA ..... Instruction Set Architecture

ISS ..... Instruction Set Simulator

JIT ..... Just-in-time

JTAG ..... Joint Test Action Group

LAN ..... Local Area Network

LSI ..... large scale integration

LUT ..... LookUp Table

LZ77 ..... Lempel-Ziv

MAC OS X .... Apple Mac Operating System

MESI ..... Modified Exclusive Shared Invalid

MET ..... Microarchitecture Exploration Toolset

MIPS ..... Millions of Instructions per Second

MIT ..... Massachusetts Institute of Technology

MMU ..... Memory Management Unit

MPI ..... Message Passing Interface

MPTRACE ..... Multiprocessor Trace

MSI ..... medium scale integration

MSR ..... Machine Status Register

MTC ..... MicroBlaze Trace Core

NIST ..... National Institute of Standards and Technology

NMSU ..... New Mexico State University

NRE ..... Non-recoverable engineering costs

NSF ..... National Science Foundation

OCD ..... On-Chip Debug

OCI™ ..... On-Chip Instrumentation ™ of FS^2

OPB ..... On-chip Peripheral Bus

OS ..... Operating System

PAPI ..... Performance Application Programming Interface

PCB ..... printed circuit board

PCI ..... Peripheral Component Interconnect PC interface defined by Intel.

PCL ..... Performance Counter Library

PDA ..... Personal Digital Assistant Portable PC, diary.

PDATS ..... Packed Differential Address and Time Stamp

PDF ..... Portable Document Format Developed by Adobe who made the reader freely available.

PHY ..... Physical Layer Device

PID ..... process identifier

PISA ..... Portable Instruction Set Architecture

PLI ..... Programming Language Interface Verilog/ VHDL.

PM ..... Performance Monitor

PMU ..... Performance Monitoring Unit

PC ..... Program Counter

PVS ..... PowerPC Visual Simulator

QED ..... Quantum Effect Devices Previously Quantum Effect Design.

`QPT` ..... Quick Profiler/Tracer

RAM ..... Random Access Memory Read/write memory.

RAMP ..... Research Accelerator for Multiple Processors

RATCHET .... Real-time Address Trace Compression Hardware for Extended Traces

RISC ..... Reduced Instruction Set Computer

ROM ..... Read-only memory Non-volatile.

RTL ..... Register Transfer Logic

RTNI ..... Real Time Non-Intrusive

RTOS ..... Real-Time Operating System

RTU ..... Real-Time Unit

RVM ..... Reference Verification Methodology Synopsys Inc.

SARA ..... Scalable Architecture for Real-time Applications

SATA ..... Serial ATA ATA

SBC ..... Single Board Computer

SBC ..... Stream-Based Compression

SCBP ..... Static Correlated Branch Prediction

SD ..... Secure Digital

SDDF ..... Self-Defining Data Format

SDRAM ..... Synchronous Dynamic RAM See RAM.

SERDES ..... Serializer/Deserializer

SGI ..... Silicon Graphics Inc.

SHRIMP ..... Scalable High-performance Really Inexpensive Multi-Processor

SMP ..... Symmetric Multi-Processing

SMT ..... Simultaneous Multi-Threading

SNI ..... Shrimp Network Interface See also SHRIMP

SoC ..... System-on-a-Chip

SPARC ..... Scalable Processor Architecture SUN

SPEC ..... Standard Performance Evaluation Corporation

SQL ..... Structured Query Language

SRAM ..... static RAM

SSI ..... small scale integration

`SUIF` ..... Stanford University Intermediate Format

SURFBOARD ... SHRIMP Usage Reporting Facility See also SHRIMP.

SWD ..... Serial Wire Debug Subset of JTAG signals from ARM

TATL ..... Trace Analysis Tool and Library

`Tcl` ..... Tool command language J. Ousterhout originally at Berkeley.

TLB ..... Translation Lookaside Buffer

TQFP ..... Thin Quad Flat Pack

TRAPEDS .... Trace-Producing Execution-Driven Simulation

UART . . . . . Universal Asynchronous Receiver Transmitter A simple serial port, generally RS-232.

USB . . . . . Universal Serial Bus

VCD . . . . . Value Change Dump

VHDL . . . . . VHSIC High-level Definition Language See VHSIC.

VHSIC . . . . . Very high speed integrated circuit

VLIW . . . . . Very Long Instruction Word

VLSI . . . . . very large scale integration

VME . . . . . Versabus Motorola European

WCET . . . . . Worst Case Execution Time

WRL . . . . . Western Research Laboratory

XML . . . . . eXtended Markup Language

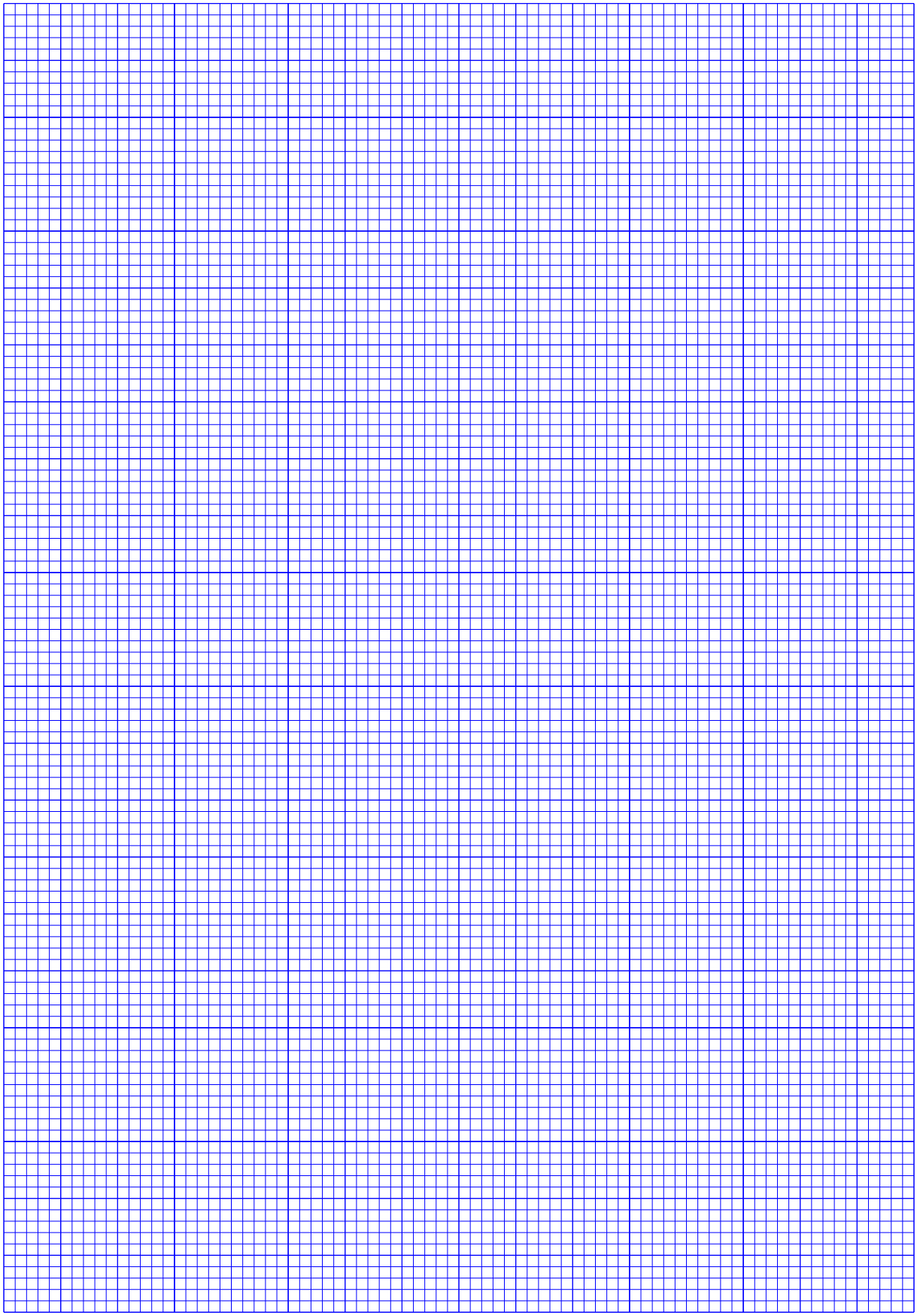


Figure 9.1: Design Notes