CITCAT: Constructing Instruction Traces *from* Cache-filtered Address Traces

A Thesis Proposal

presented to the

Department of Computer Science

Brigham Young University

In partial fulfillment

of the requirements for the

Degree Master of Science

by

Charlton D. Rose

January 1999

Introduction

Every second, a typical processor in a desktop computer completes hundreds of millions of simple operations called **instructions**. Because instructions are the basic elements of computer programs, researchers are interested in ways to make them execute faster and more efficiently.

An **instruction trace** is a record of instructions executed by a processor during a finite interval of time. Instruction traces are valuable to researchers because they provide insight into the demands placed on the processor while certain tasks are being performed [1]. This data allows researchers to identify bottlenecks in the system [2, 3], and guides them as they revise and update the processor [4], its supporting hardware, and the programs that run on it [5]. An informal survey of three major conference proceedings has revealed that:

> Approximately 23% of . . . accepted papers at . . . ISCA 24, ASPLOS VII, and MICRO-30 dealing with computer system architecture and performance use trace-driven simulation to acquire their experimental results. Most of the trace data used was representative of scientific workloads such as the integer and floating point SPEC benchmarks. [From this,] two observations can be made. . . . First, there is certainly a need for trace data since nearly a quarter of all the work found worthy of publication at these prestigious conferences use it. Second, there is a real need for trace data representative of other workloads such as transaction processing, real time applications, multimedia applications, teleconferencing, video streams, and entertainment software such as games [6].

Unfortunately, traditional trace collection techniques have failed to produce long, accurate traces that contain enough information to be called "statistically accurate" [7]. As a result, many researchers have been forced to settle for less, hoping (with some uncertainty) that the results of their research are still accurate.

Short instruction traces have limited value because they reflect the processor's behavior in a limited context. Long instruction traces, on the other hand, are more useful but difficult to obtain. The storage requirements for even 10 seconds worth of instruction trace data are enormous. What’s more, most processors do not “publish” enough data on their external pins to make a direct hardware monitor approach feasible [2]. Invariably, attempts to collect instruction traces have almost always resulted in system perturbation, thereby degrading the fidelity of the trace [7, 8]. It seems that the more closely we try to observe instruction execution, the less certain we are that we haven’t disrupted the system. This is a significant problem because studies based on inaccurate trace data often produce erroneous results [7, 9, 10, 11, 12].

For each instruction executed by a processor, the processor must make one or more accesses to its memory. This might lead one to believe that a trace of a processor's memory accesses is even more difficult to obtain than an instruction trace. However, it is not. Most processors are equipped with caches, intermediate storage devices that act as liaisons between the processor and its main memory. A cache serves as a “short term memory,” quickly responding to memory requests that are repetitive or localized in nature. Since most of a processor’s memory requests have this property [13], a cache effectively shields main memory from most of the processor’s activity. On occasion, however, when the processor requests data that isn't in the cache, the cache must accommodate the request by retrieving the data directly from main memory, passing it on to the processor, and then retaining a copy for itself. Afterwards, if the processor requests the same data in the near future, the cache can respond more quickly.

Most systems have more than one cache, arranged in a parallel and/or serial hierarchy. Caches closer to the processor are typically smaller and faster, while caches closer to main memory are larger and slower. A **cache-filtered address trace** is a record of processor memory requests that were not immediately satisfiable by the caches. Unlike *instruction* traces, long, accurate cache-filtered *address* traces are relatively easy to collect [14].

Under certain conditions, cache-filtered address traces may contain enough information to generate instruction traces [15]. To pursue this idea, I will develop CITCAT, a procedure capable of performing the conversion. CITCAT will leverage the best features of traditional trace collection techniques, such as instruction inlining [1, 7, 8, 16], hardware monitoring [2, 14], and processor simulation [17, 18], to produce long, accurate instruction traces. The beauty of CITCAT is that the effects of the tracing process on the system being traced can be negligible.

CITCAT stands for "Constructing Instruction Traces from Cache-filtered Address Traces." The premise of CITCAT is the fact that processors are **deterministic finite state machines**. This means that if a processor's present state and external stimuli are known, the next state of the processor can be accurately predicted. By determining the state of the processor at the beginning of a trace, and then recording all the external events that affect it afterwards, one can use this information to reenact, in a simulation, the processor's exact behavior. The simulated environment, in turn, makes it easy to probe any aspect of the processor's behavior without altering it [17], and without sacrificing valuable data. Thus, if both the *present state* image required to initialize the simulation and the *external stimuli* schedule required to keep the simulation on track can be obtained from a cache-filtered address trace, the address trace can be converted into an equivalent instruction trace [15]. This is the main objective of CITCAT.

My thesis will describe the procedures followed to implement CITCAT on an R4000-based system, explore the procedure's strengths and weaknesses, and quantify the results. If CITCAT proves successful, it may have a dramatic impact on the way traces are collected (and distributed) in the future.

Thesis Statement

A cache-filtered address trace, collected during an interval of processor execution, can be converted into a corresponding instruction trace, provided that the machine state at the beginning of the interval, and external events that affect the processor during the interval, can be extracted from the trace.

Methods

CITCAT combines software-based processor simulation, instruction inlining, and hardware-based bus monitoring to build accurate instruction traces, or traces that represent *real* machines operating under normal, *unperturbed* conditions.

The first step in CITCAT will be to patch the operating system with a driver that outputs the machine's state to the bus, so that it can be recorded by a hardware monitor in the cache-filtered address trace. This driver will be invoked only once, at the beginning of the trace, and then the machine will be allowed to operate as normal. Meanwhile, the hardware monitor will also record processor-external events, such as cache-filtered memory accesses and external interrupts [19]. After the trace is complete, the data recorded by the hardware monitor will be processed to produce an initial machine state record and an external event schedule, which will be used by a simulator to reproduce the processor's behavior. The simulator will then output the instruction trace.

In particular, the following three types of information will be required to run the simulation:

1. The **initial processor state**, including registers, flags, TLB entries, etc. must be known so that the processor can be properly initialized at the beginning of the simulation.
2. The **initial memory state**, including the states of the first- and second-level caches, must be known so that the memory subsystem can be properly initialized.
3. A schedule of **processor-external events**, including interrupts, DMA, and timing information, is required in order to simulate external devices connected to the system.

Some changes must be made to the operating system in order to guarantee that this information can be extracted from the trace. Hopefully, these changes will have minimal impact on the overall behavior of the system. Whether or not this is a realizable goal will be one of the subjects of my research.

The simulator will run as follows:

1. **Initialize the processor** with the initial processor state, including the program counter, registers, flags, TLB entries, etc.
2. **Initialize main memory** with the initial memory image.
3. **Execute the instruction** at the program counter.
4. **Output trace information** for the instruction just executed.
5. **Check the events schedule** to see if it is time for an external event.

6. **If it's time for an external event, simulate it** by signaling an interrupt, modifying main memory, or performing any other appropriate action.
7. **Repeat from step 3.**

Simulation is the final step in the CITCAT instruction trace generation procedure. Because this algorithm will allow the simulator to execute the same instruction sequence as a real, unperturbed processor, the simulator will (hopefully) produce a nearly perfect instruction trace corresponding to the original address trace.

Contribution to Computer Science

This thesis will demonstrate to the computer science community that it is possible to convert cache-filtered address traces into long, accurate instruction traces. The CITCAT procedure will be documented and published so that others can apply the techniques to additional computer systems. Finally, a shallow exploration of the trace compression potential of CITCAT will be conducted, in the hopes that it will inspire others to further develop the technique.

Delimitations of the Thesis

The main goal of this thesis is to prove that address traces can be converted into instruction traces. Therefore, this thesis will focus more on theory and application results, rather than hardware-specific details related to bus (address) trace collection, processor simulator design, etc. Only uniprocessor environments will be considered.

Because this research will be performed in collaboration with Tandem Computers, Inc., many specific aspects of the research, such as trace collection techniques, address traces, simulation software, generated instruction traces, etc. will be considered “proprietary” and therefore cannot be made available to the public.

Although I will also report briefly on the lossless instruction trace compression potential of CITCAT, I will not explore trace compression in detail.

Thesis Outline

I will report the results of my work using the alternative "Paper Thesis" option, as described in the graduate handbook. It will begin with the traditional required elements, such as a title page, abstract, acceptance page, etc. Thereafter, however, it will follow the format of a publishable paper. The chapters proposed for this paper are as follows:

1. **"Introduction"** (2 pages). Instruction traces will be defined, and the need for good instruction traces will be argued. Reasons why good instruction traces are hard to obtain will be explained. An overview of the rest of the paper will be given.
2. **"Legacy approaches"** (6 pages). A definition for the term "perfect instruction trace" will be proposed. Traditional techniques for collecting instruction traces, such as stepping, code modification, hardware monitoring, and processor simulation, will be reviewed. Reasons why these techniques fail to produce perfect instruction traces will be argued, but the potential for hardware monitoring and processor simulation techniques to approach this goal will be admitted.
3. **"CITCAT, a hybrid technique"** (7 pages). The general CITCAT procedure will be described. The precise nature of cache-filtered address traces for which CITCAT is possible will be discussed, including requirements for obtaining an initial machine state and asynchronous event schedules.
4. **"From theory to practice: CITCAT on a MIPS R4000 microprocessor-based system"** (5 pages). The hardware-specific challenges faced while implementing

CITCAT on an actual R4000-based system will be narrated. Many of the techniques described here will apply to other architectures as well.

5. **"Analysis of CITCAT on R4000-based systems"** (4 pages). A quantitative analysis of CITCAT-generated R4000 instruction traces will be given. The address trace/instruction trace ratio, overall system perturbation, and trace compression potential will be measured.
6. **"Conclusion"** (1 page). The main points of the paper will be summarized.
7. **"References"** (2 pages). Works cited in the thesis or related to the thesis will be listed.

Thesis Schedule

Most of this work has already been completed. The first draft of the proposed thesis has already been submitted to Dr. Flanagan, my advisor, for review. After the proposal defense, which I hope to complete by Jan. 29, 1999, I will submit the thesis to Dr. Morse, my second advisor. As soon as he has finished reviewing it, hopefully by Feb. 5, 1999, I will schedule a defense. If all goes well, I will defended my thesis by Mar. 5, 1999.

Bibliography

1. Chris Stephens, Bryce Cogswell, John Heinlein, and Gregory Palmer: "Instruction Level Profiling and Evaluation of the IBM RS/6000." *Proceedings of the Eighteenth International Symposium on Computer Architecture*, pp. 180-189. ACM 1990. Evaluates the RS/6000 architecture using instruction profiling data obtained by instrumenting benchmark software with tracing code.

2. Jeffrey Dean, James E Hicks, Carl A. Waldspurger, William E. Weihl, George Chrysos: "*ProfileMe*: Hardware support for instruction-level profiling on out-of-order processors" in *Proceedings of the thirtieth annual IEEE/ACM international symposium on microarchitecture*, 1997. pp. 292-302. Introduces *ProfileMe*, a system that samples instructions and collects detailed information about the instruction's progress through the execution pipeline, and about the interactions between pairs of instructions.
3. Chih-Po Wen: "Improving instruction supply efficiency in superscalar architectures using instruction trace buffers" in *Proceedings of the 1992 ACM/SIGAPP Symposium on Applied computing (vol. I): technological challenges of the 1990's*. pp. 28-36. A branch prediction algorithm that relies on an "instruction trace buffer" to achieve unprecedented prediction accuracy, resulting in fewer pipeline stalls. Results are based on instruction trace-driven simulation.
4. Douglas W. Clark, Joel S. Emer: "Performance of the VAX-11/780 translation buffer: simulation and measurement" in *ACM Transactions on Computer Systems*, February 1985, vol. 3 no. 1, pp. 31-62. Uses direct hardware monitoring and trace-driven simulation to characterize the performance of the VAX-11/780 TLB.
5. C. Chekuri, R. Johnson, R. Motwani, B. Natarajan, B. R. Rau, M. Schlansker: "Profile-driven instruction level parallel scheduling with application to super blocks" in *Proceedings of the 29th annual IEEE/ACM international symposium on microarchitecture*, 1996. pp. 58-67. Introduces methods for factoring a program's instruction profile into traditional heuristics for instruction scheduling that exploit instruction-level parallelism.

6. J. Kelly Flanagan: “A National Trace Collection and Distribution Resource.” 1998. Proposal submitted to and accepted by the NSF describing the need for a universal trace data distribution center.
7. Anita Borg, R. E. Kessler, Georgia Lazana, David W. Wall: “Long address traces from RISC machines: generation and analysis” (*WRL Research Report 89/14*). Western Research Laboratory, Sept. 1989. Argues that imperfect trace data can have a significant impact on the conclusions drawn from trace-driven simulation studies. Introduces a trace generation and analysis system that addresses the limitations of other methods.
8. Susan J. Eggers, David R. Keppel, Eric J. Koldinger, Henry M. Levy: “Techniques for efficient inline tracing on a shared-memory multiprocessor” in *Proceedings of the 1990 conference on measuring and modeling of computer systems.* Performance Evaluation Review, Special Issue, Vol. 18 No. 1, May 1990. Describes a tool for collecting instruction traces from multithreaded parallel execution environments, using JIT code modification. Reviews limitations of trap-based tracing techniques, and demonstrates decreased execution dilation for their code modification technique.
9. J. Kelly Flanagan, Brent E. Nelson, Greg Thompson: “The inaccuracy of trace-driven simulation using incomplete multiprogramming trace data” in *IEEE International Workshop on Modeling, Analysis and Simulation of Computer and Telecommunications Systems* (MASCOTS). Feb. 1996. Presents evidence that incomplete trace data can result in significantly erroneous estimates for certain types of proposed architectures.
10. J. Kelly Flanagan, Brent E. Nelson, James K Archibald, and Knut Grimsrud: "Incomplete Trace Data and Trace Driven Simulation" in *Proceedings of the International Workshop on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*

MASCOTS, pp. 203-209. SCS 1993. Demonstrates the sensitivity of trace-driven simulation studies to trace completeness, showing that more accurate tracing techniques are needed.

11. Richard M. Fujimoto, William C. Hare: “On the accuracy of multiprocessor tracing techniques.” Georgia Institute of Technology report GIT-CC-92-53, June 1993. Demonstrates the dramatic effect that slight system perturbation, due to inline tracing, can have on the resulting address trace, possibly leading to significant errors in trace-driven simulation experiments.
12. Knut Grimsrud, James Archibald, Richard Frost, Brent Nelson, Kelly Flanagan: “Estimation of simulation error due to trace inaccuracies” in *IEEE Asilomar Conference*. Oct. 1992. Examines the effect of trace inaccuracies on the evaluation of cache performance using trace driven simulation. Demonstrates that omitting operating system references can produce inaccurate results.
13. David A. Patterson, John L. Hennessy: *Computer Organization & Design: The Hardware/Software Interface*. 1998 Morgan Kaufmann Publishers, Inc. A well-known, authoritative textbook on many aspects of computer organization, with a focus on the hardware/software interface.
14. K. Grimsrud, J. Archibald, M. Ripley, K. Flanagan, and B. Nelson: "BACH: A Hardware Monitor for Tracing Microprocessor-Based Systems." *Microprocessors and Microsystems*, October 1993, vol. 17 no. 6. Describes the design and operation of address trace collection hardware capable of collecting long, accurate, and contiguous traces from a variety of processors before cache-filtering.
15. Charlton D. Rose, J. Kelly Flanagan: “Constructing Instruction Traces from Cache-filtered Address Traces (CITCAT)” in ACM’s *Computer Architecture News*, Dec. 1996. Describes

the failures of legacy instruction tracing approaches, and proposes the CITCAT concept in general terms.

16. Thomas Ball, James R. Larus: "Optimally profiling and tracing programs" in *ACM Transactions on Programming Languages and Systems*. Vol. 16, No. 4 (July 1994), pp. 1319-1360. Describes an efficient inline instruction tracing technique that does not require inserting code into each basic block.
17. Emmett Witchel, Mendell Rosenblum: "Embra: fast and flexible machine simulation" in *Proceedings of the 1996 international conference on measurement and modeling of computer systems*. Performance Evaluation Review, Special Issue, Vol. 24 No. 1, May 1996. Describes techniques and strategies employed in implementing Embra, the worlds "fastest reported complete machine simulator." Simulation statistics reported can be dynamically configured, making it ideal simulation-based instruction tracing. References many papers describing simulator implementation techniques.
18. Bob Cmelik, David Keppel: “Shade: a fast instruction-set simulator for execution profiling” in *Proceedings of the 1994 conference on measurement and modeling of computer systems*. pp. 128-137. Describes *Shade*, an efficient instruction-set simulator with extensible trace generation capability.
19. Lily Jow: “Tandem Memorandum 5730-G001-96” (Subject: “BYU Instruction Tracer Interface Signals (Release 1)”). Describes technical details of the joint-effort CITCAT implementation on a Tandem R4400-based system (proprietary information).
20. Joe Heinrich: *MIPS R4000 microprocessor user’s manual*. 1994 MIPS Technologies, Inc. A must-have for any researcher performing low-level work on an R4000-based computer system; describes the operation of the MIPS R4000 microprocessor in detail.

Artifacts

It is not intended that there will be any printed appendices to submit with the thesis, other than the various tables and illustrations that will be incorporated into the main body of the report. However, additional items to be submitted with the thesis will include:

1. Operating system patches and drivers to enable CITCAT on an R4400-based machine.
2. Cache-filtered address trace data recorded as the first step in the CITCAT procedure. Includes (a) SysCmd data, the cache-filtered address trace data, and (b) Status data recorded concurrently from the processor status pins for the purpose of event scheduling.
3. Software for merging SysCmd and Status traces into a unified composite trace suitable for use with CITCAT trace analysis tools.
4. Trace filtering programs to generate various schedules, such as (a) initial processor images, (b) initial memory images, (c) external interrupt schedules, (d) I/O schedules, (e) DMA event schedules, and (f) Count register schedules.
5. Patches for *Sable* (an R4000 simulator) that provide CITCAT support.

This thesis proposal by Charlton D. Rose is accepted in its present form by the Department of Computer Science of Brigham Young University as satisfying the thesis proposal requirement for the degree of Master of Science.

J. Kelly Flanagan, Committee Chair	Date
Brian S. Morse, Committee Member	Date
Robert Preece Burton, Committee Member	Date
Scott N. Woodfield, Graduate Coordinator	Date