Transparent, Low-Overhead Profiling on Modern Processors

Jennifer Anderson Lance Berc George Chrysos
Jeffrey Dean Sanjay Ghemawat Jamey Hicks Shun-Tak Leung
Mitch Lichtenberg Mark Vandevoorde Carl A. Waldspurger William E. Weihl
Compaq Computer Corporation

Over the past two years, the Digital Continuous Profiling Infrastructure (DCPI) research project at Compaq's Systems Research Center and Western Research Lab has been exploring new ways of profiling computer systems. We have developed the DCPI tools, a suite of software profiling tools that provide transparent, low-overhead (0.5%-3.0% slowdown) profiling of complete systems [1]. The DCPI tools run on Alpha microprocessors under Digital UNIX and Microsoft Windows/NT and are freely available for downloading from `http://www.research.digital.com/SRC/dcpi/`.

The DCPI tools provide profile information at varying levels of granularity, from whole images, down to individual procedures and basic blocks on down to detailed information about individual instructions, including information about dynamic behavior such as cache misses, branch mispredicts and other forms of dynamic stalls. Instruction-level stall information is attributed to the instructions that actually incur such stalls, in contrast with some systems that attribute the information to a nearby instruction. This precise attribution is extremely useful when tuning code.

On in-order processors such as the Alpha 21064 and 21164, the tools rely on periodic cycle counter interrupts and static analysis of an executable image to provide instruction-level information. On out-of-order processors, this approach is not feasible and we have designed a new form of hardware support for instruction-level information called *ProfileMe*, which can provide significant insight into the behavior of programs running on complex microprocessors (especially out-of-order processors) [2]. *ProfileMe* requires only modest hardware modifications and can be used by our DCPI tools in a way that collects detailed profile information without substantial profiling overhead.

The DCPI profiling tools have several characteristics that distinguish them from other profiling tools:

- **Profiling is efficient, transparent and easy to use.** The system has an overhead of 0.5%-3.0% slowdown for most workloads, which is low enough that many users leave the profiling software turned on all the time. Unlike many other profiling systems, there are no separate steps to prepare a program for profiling. When the system is turned on, profiling happens transparently, for all activity on the entire system. The fact that profiling is transparent and can be left on all the time is important, since the omnipresence of profiling information encourages programmers to use profiling data as a matter of course, rather than doing so only in exceptional circumstances. DCPI also stores all profiles in a profile repository and automatically maintains the association of executable images to profiles. This means that profiles remain valid and can be located even if executables are moved or copied, freeing programmers from having to explicitly manage profile files.

- **The profiles provide instruction-level information.** Most profiling tools in common use today measure instruction execution frequencies (e.g. `pixie`). In today's modern microprocessors, with deep memory hierarchies, sophisticated branch predictors, and dynamic out-of-order execution pipelines, instructions are not all the same. Execution counts hide the distinction between a single-cycle add operation and a load operation that takes a hundred cycles or more to complete. Profiling tools such as `prof` that measure or estimate execution time instead of instruction counts fix this problem. However, most time-based tools can only show where time is being spent and are unable to provide much insight into why particular regions of code are consuming time. Often the reasons for a stall are not immediately obvious even to those knowledgeable about a CPU's microarchitecture, and to those programmers that are not intimately familiar with the hardware details of the processor, the reasons for a stall are often quite mysterious. To address this, the DCPI tools identify instructions that stall and also identify the potential cause(s) for the stall (*e.g.* data cache miss, branch mispredict, etc.). Many users of DCPI tools have relied on this instruction-level insight to significantly improve the performance of their programs, and we are working on various automatic compiler optimizations that take advantage of this kind of information.

Chrysos is at Digital Semiconductor (chrysos@vssad.hlo.dec.com), Hicks is with the Cambridge Research Lab (jamey@crl.dec.com), Anderson and Dean are with the Western Research Lab ({jennifer,jdean}@pa.dec.com), and the remaining authors are with the Systems Research Center ({berc,sanjay,sleung,mitch,mtv,caw,weihl}@pa.dec.com). More information about profiling research at Compaq can be found on the Web at `http://www.research.digital.com/SRC/dcpi/`.

- **The profiles cover the entire system.** DCPI profiles the entire system, including executable programs, shared libraries, device-drivers, and the operating system kernel. Bottlenecks can show up in any of these places, and the trend in modern applications is to separate programs into a growing number of shared libraries and often into multiple communicating processes. Therefore, it is increasingly important to be able to profile the entire computer system, rather than just a single application.

Perhaps the most unusual aspect of the DCPI tools and *ProfileMe* hardware is their ability to deliver instruction-level information about cache misses, branch mispredicts and other dynamic stalls, with the information attributed to the exact instruction(s) that experience these events.

The technique for gathering instruction-level information differ depending on whether or not the profiling is being done on an in-order or out-of-order processor. We begin by discussing how we collect instruction-level information for these two classes of processors, and then discuss some applications of instruction-level profiling information in the context of program optimizations. Further details about the profiling software and hardware can be found in two other papers [1] [2].

Gathering Instruction-Level Information on In-Order Processors: PC Sampling

On in-order processors, the DCPI tools rely on gathering samples of the program counter value (PC) randomly using a periodic cycle counter interrupt, producing PC sample counts whose expected value for each instruction is proportional to the total time spent executing that instruction. On the in-order Alpha 21064 and 21164 processors (and on many other in-order processors), the "total time" measured by the PC samples for an instruction is the number of cycles for which that instruction was the next instruction to be issued. For example, a load instruction that takes 10 cycles on average will accumulate 10 times as many PC samples as will an add instruction in the same basic block that takes only one cycle on average to execute. By analyzing the sample data for groups of frequency-equivalent instructions (instructions that are guaranteed to execute the same number of times), it is possible to estimate an execution frequency and an average cycles-per-instruction-execution value for each instruction in the program. Once this is done, a variety of heuristics are used to help explain why particular instructions take longer to execute than their ideal case execution behavior. Explanations considered include data cache and instruction cache misses, data and instruction TLB misses, branch mispredicts, and various other forms of dynamic stalls. The approach taken is to consider all the considered reasons as possible and to have the heuristics rule out cases that cannot happen or that are extremely unlikely. For example, the heuristic for an instruction cache miss considers this as a possible stall reason for instructions that start a basic block or that are at the beginning of an instruction-cache line. Heuristics for other stalls have a similar flavor. These heuristics are able to narrow down the set of possible causes for a stall to only one or two possible reasons 80% of the time, and often (56%) can narrow the cause down to a single reason. These analyses depend on the fact that the Alpha 21064 and 21164 processors are in-order processors, and are also helped by the fact that the pipelines of these processors are relatively simple.

Gathering Instruction-Level Information on Out-of-Order Processors: ProfileMe Hardware

Although the techniques described above work effectively for in-order processors, they break down for out-of-order processors, because the number of samples for a given PC is no longer proportional to the amount of time spent executing the instruction. To gather instruction-level information on such processors, we have developed *ProfileMe*, a new form of hardware support for performance measurement. *ProfileMe* differs from the traditional hardware event counters provided by most processors. While event counters provide useful aggregate information, such as the total number of branch mispredicts during a program run, they do not accurately attribute these events to individual instructions. The reason is that the instruction that caused an 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 handling the event counter overflow. Thus, sampling of events such as data cache misses or branch mispredicts using event counters gives samples that are "in the neighborhood" of where the event has occurred, but it is often difficult or impossible to identify the exact instruction that caused the event. Out-of-order and speculative execution amplify this problem, but it is present even on in-order machines.

The approach used in *ProfileMe* is quite different. Rather than counting *events* and sampling the program counter when the event counters overflow, we sample *instructions*. We introduce a software-loadable counter that counts fetched instructions. As each instruction is fetched by the processor, the counter value is incremented. When it overflows, hardware hardware in the fetch unit tags the next instruction to be fetched as a profiled instruction. As a profiled instruction executes, other logic in the processor pipeline records information about its execution in internal registers. Information recorded includes the instruction's PC, the number of cycles spent in each pipeline stage, whether it suffered I-cache or D-cache misses, the effective address of a memory operand or branch target, and whether it retired or why it aborted. After the instruction completes, we generate an interrupt and deliver the recorded information to software. This approach has several benefits. First, all the recorded information is directly attributed to the PC value that was recorded for the instruction. Second, a single sample

delivers an entire record of what happened to an instruction during its execution. This is in contrast with an event counter, which delivers only unattributed information about a single event, without any correlation to other events in the system. By aggregating the *ProfileMe* samples for multiple executions of the same instruction, metrics such as branch mispredict rates or data cache miss rates *for individual instructions* can easily be estimated. By using the per-pipeline-stage latency information, information about which instructions are stalled in which parts of the processor can be obtained.

The *ProfileMe* hardware gives a detailed record of what happened to a single instruction during its execution. However, on out-of-order processors, even knowing which individual instructions have stalled in various pipeline stages is not sufficient to understand the performance impact. Since out-of-order execution is explicitly designed to mask stalls of individual instructions, it is important to be able to understand what other activity is going on in the machine at the time that the instruction was executing. For example, consider an instruction that incurs a 20 cycle stall sitting in the issue queue waiting for its data operands to becomes available. Is such an instruction a problem? On an in-order processor, the answer is a definitive yes: progress for all subsequent instructions is block by this stall. On an out-of-order processor, the answer is maybe: if there is sufficient concurrency available in the code stream surrounding the instruction that the stall can be masked by performing other useful work, then the stall is not a problem. To understand performance bottlenecks on out-of-order processors, it is important to also be able to measure instruction-level concurrency. Through the use of *paired sampling*, which replicates the ProfileMe hardware to permit the simultaneous sampling of two potentially concurrent instructions, we can examine the overlap of pairs of samples, and by aggregating many such pairs, we can use statistical analyses to estimate a variety of interesting concurrency metrics for individual instructions, such as number of retired instructions while the instruction was in flight, or number of wasted issue slots while the instruction was in flight [2].

Uses of Instruction-Level Information

The DCPI tools have been widely used both within Compaq and externally by a number of other companies and universities. Much of their use arises in examining the performance of existing programs and systems to manually tune performance. For example, they have been used extensively to improve the code generated by Compaq's production compiler for UNIX and Windows/NT. Use of our tools led to improvements in the code generation of the compiler that produced a 20% speedup in several SPEC95 benchmarks. Similarly, our tools identified D-cache stalls as the major problem in the inner loop of a major Windows/NT benchmark; a combination of prefetching and hoisting loop-invariant loads out of the loop yielded a total speedup of about 20%. Use of our tools pinpointed a performance problem in running one query of a widely-used decision-support database benchmark, and fixing this problem resulted in a factor of 20 speedup for this piece of the benchmark. Similar results have been obtained on a wide range of programs.

In addition to manual tuning, we are investigating using detailed instruction-level information about stalls and their causes to drive automated optimizations. For example, we are looking at prefetching based on measured latencies for loads, and at reorganizing data structures with poor cache performance to reduce pollution of the cache by data that is never used.

Conclusions

The DCPI tools demonstrate that low-overhead transparent instruction-level profiling of complete systems is possible, and our experiences and those of our users have shown that instruction-level information is extremely helpful in understanding how programs perform on modern machines. On in-order processors instruction-level information can be obtained simply with periodic interrupts. The development of *ProfileMe* will enable the same sort of low-overhead transparent instruction-level profiling on out-of-order processors. To date, the instruction-level information has proved invaluable in doing manual tuning of computer programs and systems. We are continuing to perform research on using this kind of instruction-level information to drive automatic optimizations.

Acknowledgments

We would like to thank all of the people involved with the DCPI and ProfileMe projects, including Monika Henzinger, Scot Hildebrandt, Jim Keller, Rick Kessler, Dan Liebholz, Ed McLellan, Dick Sites, Gerard Vernes, and Jon White, for their significant contributions to these projects. In addition, many users of DCPI have contributed valuable feedback that have greatly improved the usability of the DCPI tools.

References

[1] J. Anderson, L. M. Berc, J. Dean, S. Ghemawat, M. R. Henzinger, S.-T. Leung, R. L. Sites, M. T. Vandevoorde, C. A. Waldspurger, and W. E. Weihl. Continuous profiling: Where have all the cycles gone? In *ACM Transactions on Computer Systems*, pages 357–390, Nov. 1997. An earlier version appears in the *Proc. of the 16th Symp. on Operating System Principles*, St. Malo, France, Oct. 1997.

[2] J. Dean, J. E. Hicks, C. A. Waldspurger, W. E. Weihl, and G. Chrysos. *ProfileMe*: Hardware support for instruction-level profiling on out-of-order processors. In *Proc. 30th Annual Intl. Symp. on Microarchitecture*, Dec. 1997.