Kernel-based Memory Simulation (Extended Abstract)

Richard Uhlig, David Nagle, Trevor Mudge & Stuart Sechrest

Department of Electrical Engineering and Computer Science
University of Michigan
e-mail: uhlig@eecs.umich.edu, bassoon@eecs.umich.edu

1 INTRODUCTION

Trace-driven simulation is a widely-accepted technique for studying the components of computer memory systems such as caches and translation look-aside buffers (TLBs). However, trace-driven methods are time consuming, requiring 20 to 50 times as long to run as actual hardware, and often cannot accurately take process interaction and operating system effects into account.

To overcome the limitations of trace-driven simulation, we have developed an alternative approach in which memory simulators run in an active operating system kernel. This method, called kernel-based memory simulation, allows us to account for all system activity, including multiple process and kernel interactions. Further, by using privileged machine operations to cause traps into the simulator only when a miss in a simulated memory structure occurs, a kernel-based simulator is able to process hits at the full speed of the underlying host hardware.

Our implementation of a kernel-based simulator, called *Tapeworm*, uses a three-step algorithm:

(1) On a miss, trap to the kernel-resident simulator, count the miss and clear the trap on the missing memory location so that future misses can proceed uninterrupted.

(2) Invoke a replacement policy that selects an entry to be displaced from a simulated TLB or cache.

(3) Set a trap on the displaced memory location so that future references will miss and re-invoke the simulator.

Several different techniques can be used to force traps into the kernel after misses in a simulated memory structure. For TLB simulation, where page-size granularity is needed, traps can be set or cleared using page valid bits, or by installing and removing page-table entries in a software-managed TLB [1]. For cache simulation, where cache-line granularity is needed, instruction breakpoints, data breakpoints, or memory parity traps can be set [2].

Two essential features of this approach enable Tapeworm to overcome the limitations of trace-driven simulation. First, because Tapeworm processes only the infrequent case of misses, it is much faster than comparable trace-driven simulators, which consider all memory references. Second, because Tapeworm resides in the kernel of a running operating system, it is in an ideal position to take multiple process interaction and operating system effects into account. A disadvantage of kernel-based simulation is that it is less flexible than traditional trace-driven simulation with respect to the TLB and cache configurations that it can simulate. Further, this method does not work for other forms of architectural modeling, such as instruction-pipeline simulation.

2 EXPERIENCES

We have implemented Tapeworm TLB and instruction cache simulators to run on a DECstation 5000/200 under Mach 3.0 with a user-level BSD UNIX server. This extended abstract reports primarily TLB simulation results. For a more detailed discussion of the design and performance of the Tapeworm I-cache simulator, see [4].

Figure 1 shows that typical slowdowns for Tapeworm TLB simulation are 0.25 to 1.25. Because miss ratios tend to be higher for caches than for TLBs, Tapeworm I-cache simulations exhibit larger slowdowns, from close to 0 to as high as 7. These results compare favorably with trace-driven simulations where slowdowns are usually in the range of 20 to 50.

This work is supported by Defense Advanced Research Projects Agency under DARPA/ARO Contract Number DAAL03-90-C-0028 and a National Science Foundation Graduate Fellowship.

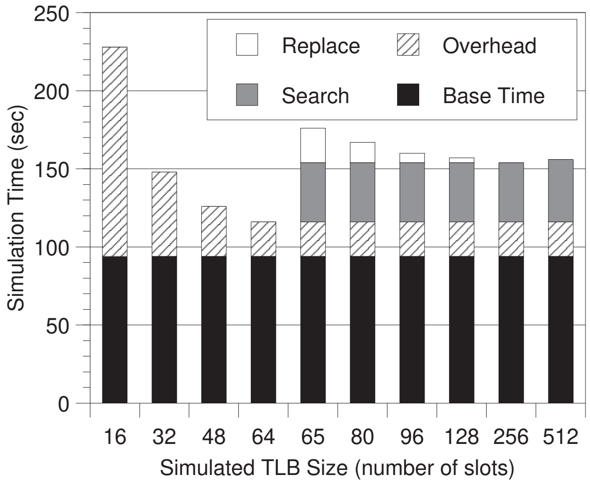

Figure 1: Tapeworm Simulation Times

This figure shows typical Tapeworm simulation times for different sizes of TLBs. *Base Time* is how long the workload would run without Tapeworm in the kernel, while *Overhead* is the time added by the Tapeworm code that intercepts TLB misses. *Search* and *Replace* are the amount of time used by Tapeworm to maintain a data structure for simulated TLBs that are larger than that of the underlying host's TLB (64 slots in this case).

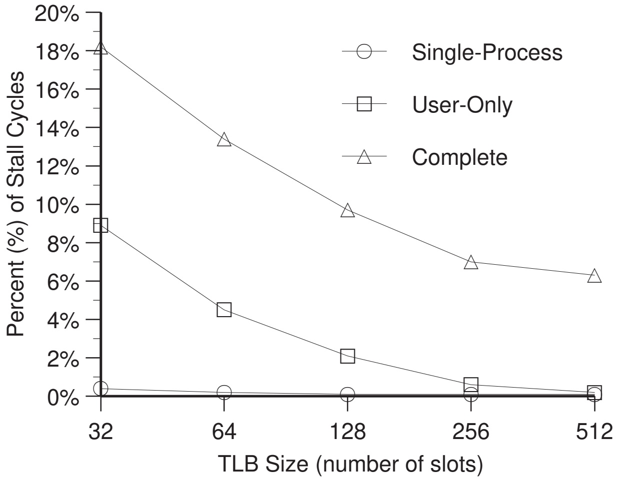

Figure 2: Effects of Multi-process and OS References

This plot shows TLB performance (given as a percentage of total memory stall cycles) for a workload that decompresses an MPEG video. The different lines correspond to the amount of workload activity taken into account. *Single-process* means that only references from the MPEG decoder process were considered, while *User-only* refers to all user processes (include X display and UNIX servers) and *Complete* refers to all references, including those made by the kernel.

Figure 2 illustrates the importance of including multi-process and operating system references in TLB simulations. For this workload (an MPEG decoder), the time is divided between four address spaces, the MPEG process (40%), an X display server (5%), the BSD UNIX server (30%) and the kernel (25%), each of which compete for slots in the TLB. In the plot, the large difference between the *Single-Process* data and the *Complete* data show that a simulation methodology that only takes the MPEG process activity into account would dramatically underestimate interference between the four address spaces, and thus the true impact of the TLB on overall memory performance. Because it resides in the kernel and can field a trap from any address space, including that of the kernel itself, Tapeworm easily takes multiple process and kernel references into account (see the upper two lines of Figure 2).

Although Tapeworm can simulate TLB and cache configurations of varying sizes, associativities and line sizes, it does suffer some problems of flexibility. In particular, architectural structures which require accurate accounting of time, such as write buffers can not be simulated using this approach.

3 FUTURE WORK

We are extending this work in several ways. We are using Tapeworm to study architectural support for next-generation operating system technology like Mach 3.0, in work follows our previous studies in this area [1, 3]. Additionally, we are working on ways to improve the kernel-based simulation method itself. To increase Tapeworm's speed, we are exploring set-sampling techniques and investigating architectural support to lower the cost of handling misses. To enhance flexibility, we are looking at low-cost ways to design hardware to better accommodate a kernel-based simulator. Finally, we are studying the use of traditional statistical methods for isolating and characterizing sources of variation between simulation results.

4 REFERENCES

[1] D. Nagle, R. Uhlig, T. Stanley, S. Sechrest, T. Mudge, and R. Brown, "Design tradeoffs for software-managed TLBs," in *The 20th Annual International Symposium on Computer Architecture,* San Diego, California, IEEE, May 1993.

[2] S. Reinhardt, M. Hill, J. Larus, A. Lebeck, J. Lewis, and
Prototyping of Parallel Computers," in *SIGMETRICS 93 (Special Issue of Performance Evaluation Review),* Santa Clara, CA, ACM, May 1993.

[3] D. Nagle, R. Uhlig, T. Mudge, and S. Sechrest, "Optimal Allocation of On-chip Memory for Multiple-API Operating Systems," in *The 21st International Symposium on Computer Architecture,* Chicago, IL, April 1994.

[4] R. Uhlig, D. Nagle, T. Mudge, and S. Sechrest, "Tapeworm II: Ultra-Fast Memory System Simulation," University of Michigan Technical Report (in preparation).