Accelerating Multiprocessor Simulation
6.895 Project Proposal

Kenneth C. Barr

October 8, 2003

1 Introduction

To understand and improve a multiprocessor system, one must be able to simulate changes to its behavior (e.g., coherency scheme) and its various configuration parameters (e.g., number of processors, cache sizes, etc). The iterative debug loop must be reasonably quick and full-scale simulations should be as fast as possible if the simulator is to be a desireable tool. There are countless ways to attack slow simulators from coding optimizations (e.g., the macro-based instruction execution of SimpleScalar) to parallel hosting of simulation in which each simulated processor runs on a different physical processor. [6] provides a good survey of research used to speed simulation of large systems with components that operate in parallel; the paper also discusses problems specific to such large-scale simulation including the slowdown imposed by synchronization of the blocks that operate in parallel (or by replay when this synchronization is relaxed). Speed boost typically comes at the expense of accuracy and/or complexity[1] so techniques which can minimize this expense are prefered. The RSIM group examines such tradeoffs in the context of shared-memory multiprocessors [2] and [5].

My project is motivated by two interesting techniques which strive for speed with minimal impact on accuracy and a not un-reasonable amount of complexity. In the first, "Rigorous Statistical Sampling," short periods of slow, detailed simulation are alternated with long periods of fast, reasonably accurate simulation. If carefully performed, the technique yields accurate results with dramatic speedup [8]. Multi-configuration simulation allows a single simulation run to report the effects of many different processor configurations with a single run. For example, the properties of associative caches allow a n-way associative cache to answer questions about an m-way associative cache where $n > m$ [7].

We are studying Theory of Parallel Systems, so I will investigate speeding the simulation of directory-based cache-coherence protocols. I propose to implement a preliminary "meta-directory" structure in software that can be quickly filled with load and store requests during periods of fast "functional warmup." When it comes time for detailed simulation, this structure must support the "replay" of requests into an accurate memory model so that the machine is in a sequentially consistent state. I will report on the hardware and application configurations that allow such a structure, and I will discuss cases in which preliminary structure falters.

2 The Meta-Directory

The design of the meta directory has not been completed. This section includes a transcription of a conversation with Professor Krste Asanovic.

For each cache line, store a cicular queue of n last memory accesses. The queue must maintain a notion of time-ordering, but the structure should allow for out-of-order inserts by any processor. The oldest entries can be overwritten. n must be small to save memory and give us fastforward-like speeds, but large enough

[1]this simulation triangle idea may be attributable to Todd Austin

for correctness. Determining this size is one challenge of the project. Intuition suggests it may be close to $associativity * processors$.

An *access* is comprised of the ID of the processor which generated it; a timestamp; and whether it was a read or write. The resulting structure looks something like:

```
addr1:  {node #, time, r/w}, {node #, time, r/w}, ... , {node #, time, r/w}
addr2:
...
addrm:
```

We can construct the directory state from this structure as follows. Scan backwards in time to build the directory. Once you see a write, previous accesses don't matter (and can be removed to save space) and all the readers you've encountered are readers. For example the access pattern `R R R W R R R R R` becomes `W R R R R R`.

The question is: when is such a structure going to give different results from sequential simulation (and when does it actually matter that the results are different)? Spin locks might be a problem, generating lots of useless reads that roll thru the queue: a write that would have changed the flow of computation is never seen, so a different (but hopefully correct) execution is seen.

We also talked about an easier way to seperate each node's set of accesses. Maybe each address in the table stores a pointer to each node's list:

```
         __>  {,,}->{,,}->{,,}->
        /
addr2:  --->  {,,}->{,,}->{,,}->
        \
         -->  {,,}->{,,}->{,,}->
```

Maybe "time" in the above structure should be a time -range- so that we can coelesce a single node's accesses:

```
struct read{
   time of first read (may not even be necessary)
   time of latest read
}

struct line{
    enum initial access (one of "null," "write by x," "read by x");
    reads * proc;  //for each processor, note time of first read and latest read
}
```

3 Methodology

I will begin by choosing an SMP simulator such as Simics [4], M5 [1], or Bochs [3]. Each has its advantages and disadvantages. Simics is product of commerical quality, but is closed-source and thus precludes certain modifications. M5 provides accurate models for bus contention and already contains both fast and detailed components, but runs the (relatively) obscure Tru64 flavor of Unix. Bochs is a open-source and x86-based, but has recently lost reliable support for SMP simulation. It is my hope that a working SMP Bochs will be brought up shortly.

Once a simulator has been chosen, I will implement a simple, yet detailed directory model. The next step will be the implementation of the meta-directory structure that can be replayed into the detailed model yielding a sequentially consistent state. Microbenchmarks in Cilk should be sufficient to study the correctness of my models. If unforseen difficulties arise (or, if time does not permit completion of a working system), my results will be an analysis of the challenges in creating such a structure. I hope that discussions with the course staff will help me present a theoretic argument in addition to concrete examples.

My project may be influenced by the directions of Sean Lie's transactional memory project which is in the ISCA pipeline. If the infrastructure work needed for my project can benefit Sean, we have discussed the possibility of joining forces to meet his deadline.

4 Acknowledgements

I would like to thank Professor Krste Asanovic for suggesting the idea and Professor Steve Reinhard for offering me a preview copy of M5. Bradley Kuczmaul and Emmett Witchel have jump-started my Bochs proficiency.

References

[1] N. L. Binkert, E. G. Hallnor, and S. K. Reinhardt. Network-oriented full-system simulation using m5. In *Sixth Workshop on Computer Architecture Evaluation using Commercial Workloads*, February 2003.

[2] M. Durbhakula, V. S. Pai, and S. V. Adve. Improving the accuracy vs. speed tradeoff for simulating shared-memory multiprocessors with ILP processors. In *3rd International Symposium on High-Performance Computer Architecture*.

[3] K. Lawton et al. *Bochs*. http://bochs.sourceforge.net.

[4] P. S. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Hllberg, J. Hgberg, F. L. A. Moestedt, and B. Werne. Simics: A full system simulation platform. *IEEE Computer*, Feb. 2002.

[5] V. S. Pai, P. Ranganathan, H. Abdel-Shafi, , and S. V. Adve. The impact of exploiting instruction-level parallelism on shared-memory multiprocessors. *IEEE Transactions on Computers*, 1999.

[6] D. M. Rao and P. A. Wilsey. An ultra-large scale simulation framework. *Journal of Parallel and Distributed Computing*, 2002.

[7] R. A. Sugummar. *Multi-Configuration Simulation Algorithms for the Evaluation of Computer Architecute Desig ns*. PhD thesis, University of Michigan, 1993.

[8] R. Wunderlich, T. Wenisch, B. Falsafi, and J. Hoe. SMARTS: Accelerating microarchitecture simulation via rigorous statistical sampling. In *30th International Symposium on Computer Architecture*.