Moola: Multicore Cache Simulator

Charles F. Shelor and Krishna M. Kavi
Department of Computer Science and Engineering, University of North Texas
Denton, TX, 76203, USA
(cfs0042, kavi)@unt.edu

Abstract

Chip multiprocessors have become the normative architecture for medium and high performance processors. These devices introduce new questions and research topics. One such topic is exploring the design space of a cache-memory hierarchy that prevents the memory accesses from being a limiting factor on system performance. Simulation of system workloads is a widely accepted method for evaluating proposed cache organizations. Cycle accurate simulation of multicore devices requires a significant amount of time, limiting the number of configurations that can be analyzed. The generation of a memory access trace file from a cycle accurate simulation can be used to analyze multiple cache configurations in much less time.

This paper introduces Moola, a multicore, trace-based cache simulator with cycle accurate timing within the cache-memory subsystem. Moola is suitable for experimenting with different cache configurations, including different types of last level cache (LLC) implementations and demonstrating to researchers and students how access congestion at the shared LLC can adversely impact the system performance. Moola is highly configurable at run-time through configuration files and command line arguments. An analysis of congestion effects in the LLC is provided as an example of how Moola can be used to analyze current cache constructs.

keywords: Computer architecture, multicore processors, cache simulation.

1 Introduction

The continuing trend to higher throughput processors with limitations on total power dissipation has resulted in the proliferation of multicore processors rather than pushing the clock rates of uniprocessors [1][6]. Having 8 to 32 processing cores on a single chip requires an efficient cache-memory subsystem to prevent memory system bottlenecks from being the limiting performance factor. Characterization of the multicore memory workload and the performance of various cache-memory organizations is essential to developing multicore processors that meet their performance expectations.

Simulation is a common methodology used for identifying performance bottlenecks and for exploration of alternative solutions. Cache simulations are typically trace-driven or execution-driven [13]. Cycle accurate simulations are very slow while binary-instrumented code runs close to real time. Either of these options can generate the trace files needed for cache simulations. The principle disadvantage of trace files is the file size can be 10's to 100's of Gigabytes even after compression. Execution driven simulators do not have large file sizes, but reproducibility of a multicore binary-instrumented code is difficult with variations in operating system response to establishing the multiple processes. Trace files maintain the same ordering providing repeatable stimulus for multiple cache system organizations.

While DineroIV [3] is a widely used cache simulator that accepts memory traces as inputs, permitting various cache configurations, it is not suitable for exploring the design space of multicore systems with both private and shared caches. There have been many extensions to Dinero, but in most cases they are ad hoc and do not permit a systematic evaluation of different cache organizations in a multicore environment. Moola addresses the limitations of previous trace driven cache simulators. Moola is an open source tool and can be easily configured or extended to meet most needs for simulating multicore cache systems. The tool can be particularly useful for students in computer architecture course to understand the impact of cache designs on the performance of multicore processors.

The remainder of this paper provides details of the Moola cache simulator in section 2. An example application of Moola analyzing the L3 access congestion for 21 benchmarks is given in section 3. A short assessment of Moola's accuracy is given in section 4. A summary discussion of related work in multicore cache simulation is presented in section 5. The conclusions for the paper appear in section 6.

2 Moola Multicore Cache Simulator

Moola is a multicore cache simulator developed for use in a university environment to illustrate the complexities of multicore cache systems. It is highly configurable to provide for simulation of a variety of cache structures. It includes a built-in timing model to show the performance impacts of different cache parameters in a multilevel cache-memory subsystem. It is a modular, open-source tool to allow customization and extension into a variety research projects. Moola is designed as a trace based simulator allowing a single time-consuming cycle-accurate

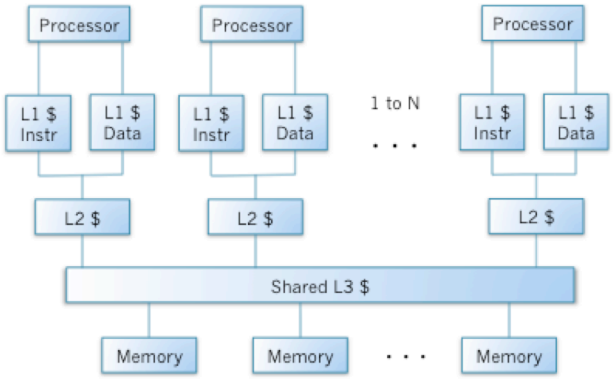

Figure 1: Moola Example System Architecture

simulation to generate a trace file, which is then processed rapidly through dozens or even hundreds of cache configurations. Three key features of Moola are detailed in sections 2.1 Moola Configuration, 2.2 Moola Last Level Cache Types, and 2.3 Moola Timing Model. Figure 1 is a block diagram illustrating an example Moola architecture showing multiple processors, multiple split L1 private caches, multiple unified L2 private caches, a shared L3 unified cache with multiple ports to main memory. A two-level cache with a shared L2 can be configured also. The interactions of private and shared caches increase in complexity when shared data, locks, and barriers must be modeled. The simulation must not change the order of access of any of these structures to ensure the simulation accurately reflects the system being analyzed. Coherency of shared data must be modeled in the cache simulator and coherency statistics should be reported. False sharing statistics should be reported also.

2.1 Moola Configuration

Moola is very flexible in the amount and the manner in which it can be configured. There are two main configuration classes for a Moola run: physical system and run management. The physical system being modeled must be described sufficiently to allow Moola to simulate the characteristics of the system being evaluated. The inputs to be applied during the simulation must be identified and coordinated. A combination of configuration files and command line arguments provide a modular configuration environment. A base configuration can be described in a configuration file and then individual items can be given modified values in the command line. A script to analyze the effects of L2 associativity is easily constructed as seen in Figure 2. The "*-cfg file*" command line option reads the specified file and processes the configuration data in the file. The "*-l2_assoc*" option then overrides the configuration value from the file with the value on the command line.

The configuration files can reference other configuration files. This allows a configuration file for each of several

```
# script to analyze associativity in L2 cache
moola -cfg base.cfg -l2_assoc 1
moola -cfg base.cfg -l2_assoc 2
moola -cfg base.cfg -l2_assoc 4
moola -cfg base.cfg -l2_assoc 8
```

Figure 2 Configuration Script Example

L1 cache configurations, each of several L2 cache configurations, each of several L3 cache configurations and then system configuration files that mix and match the different cache configuration files to obtain the different system structures to analyze. Table 1 lists the cache parameters that can be configured. The parameter option is prefixed with a cache identifier such as *l1d* for the level 1 data cache or *l2* for the level 2 unified cache.

Table 1: Configurable Cache Parameters

Option	Value
_access	Integer: cycles access time
_assoc	Integer: associativity
_bsize	Integer: block size in bytes
_coherent	String: coherency protocol
_distrib	Integer: block size for distributed cache interleave factor
_org	String: cache organization {private | shared | nonblock | distributed}
_pref	String: prefech policy
_replace	String: replacement policy
_sbsize	Integer: subblock size in bytes
_size	Integer: cache size in bytes
_write	String: write policy

Table 2 shows the general configuration options for setting the number of processor cores, assigning input files to processors, configuring the main memory parameters, controlling configuration file reading and writing and some statistics output controls. The actual configuration for each run can be written to a configuration output file in a format that is readable as a configuration input file for a repeated run and to document the configuration of the run.

2.2 Moola Last Level Cache Types

The last level cache (LLC) of a multicore system accepts all of the memory accesses that will terminate in the LLC or that must be resolved with an access to main memory. Applications that have relatively high LLC miss rates can force a bottleneck at the LLC, especially when multiple instances of the application are running at the same time. The implementation of the LLC will have a significant impact on the overall performance of the system. Moola has 3 models of LLC that can be evaluated. The simplest model is that of a blocking cache. The first access to an

Table 2: General Configuration Parameters

Option	Value
cfg	String: configuration file name
cfg_out	String: configuration output file name
comb_i_d	Combine instr & data stats
cores	Integer: number of cores to sim
csvfile	String: name of CSV out file
h, help	Print short help message
h, help	String: detail help on command
informat	String: trace file format
mem_access	Integer: memory access in cycles
mem_adrs	Integer: memory on each port
mem_nleave	Integer: interleave size for mem
mem_ports	Integer: number memory ports
multicore	String: multicore trace file name
output_sets	Output statistics for sets
preset	String: select a preset config
snapshot	Integer: snap shot integer instr
unicore	String: trace file for proc N
nicore_sh	String: trace file for proc N, instructions shared

idle LLC will be accepted and the access will be processed. All subsequent accesses to the LLC will be blocked and queued. These LLC requests will be processed in the order that they are received after the initial access completes. The maximum wait time for the last arriving request is the number of pending requests times twice the sum of the LLC access time plus the main memory access time. This accounts for the case in which all of the pending accesses are LLC misses and must perform a cache line write-back before replacing the cache line from the main memory. The average wait time at the LLC will be dependent on the average LLC miss rate for the currently running applications. The maximum wait time for the last arriving request for the case of all LLC hits is the number of pending requests times the LLC access time.

The second LLC model implemented by Moola is a non-blocking hit-under-miss cache. After the first access to an idle LLC, subsequent access requests while the cache is being accessed will be blocked on a first-come-first-served queue. If the current access is a miss and a memory reference is initiated, the first queued accessed will be processed. If this access is a hit, the access is completed and the next queued access is processed. If the access is a miss, the miss is queued for memory access and the next queued LLC access is started. If multiple memory ports have been configured and the second miss uses an idle memory port, then the miss resolution can be immediately initiated. The maximum wait time for this model is the LLC access time plus twice the main memory access time for a sequence of all misses with write-backs. Note that only one LLC access time is needed as the other LLC accesses are hidden by the main memory accesses. The maximum wait time for the last arriving request for the case of all LLC hits is the number of pending requests times the LLC access time.

The third LLC model implemented in Moola is a non-blocking distributed cache. The LLC implementation is partitioned into the number of blocks controlled by a configuration parameter to model the number of physical memory segments in the LLC. When an LLC access arrives its address is examined and routed to the appropriate segment. Each segment operates as a non-blocking cache. If the arriving requests do not access the same memory segment, the accesses proceed in parallel. If all of the accesses are to the same memory segment in the LLC, then this model performs exactly like the second model. However, if the addresses do not result in conflicts, this model provides an LLC access speedup limited to the minimum of the number of cores or the number of LLC segments. Running appropriate benchmarks through Moola will quantify the speedup possible for different applications.

2.3 Moola Timing Model

The Moola built-in timing model will not perform cycle accurate timing of multiple-issue, out-of-order processors with the memory subsystem. However, it will provide a good estimate of memory system contributions to total execution time. All of the timing parameters are entered as integer multiples of the processor clock. A single CPI model is constructed by indicating a split L1 instruction cache having a 1 clock access time. The L1 data cache is given a 0 clock access time. This results in 1 clock per instruction if all cache accesses are hits. The trace files must contain instruction fetches if this timing model is to be used.

Each processor model has a buffer of memory traces. At simulation start-up, the selected trace files are read and the buffers are filled. The time-of-issue of the first instruction in each buffer is set to the current simulation time. The first memory reference is pulled from the buffer and processed. The number of cycles to complete the access is returned, added to the current simulation time, and stored as the issue time of the next trace transaction. Instruction trace records should appear in the trace files prior to the data trace record for that instruction. This ensures that no data access is simulated before the instruction access completes. Each trace buffer is searched to find the buffer with the smallest issue time for the next trace record.

If a miss is detected during the processing of a trace record, the cache with the miss calls the next level cache to resolve the miss. This may include a write-back of the current cache as well as a read of the missing data. Timing of the operation is based on the number of cycles per access configured for the cache, the number of accesses required, and the amount of time an access might be blocked waiting on the lower cache level.

The trace file is read to refill the buffer whenever it becomes empty. If the end of the trace file is reached, the simulated processor is marked idle. There is also an option to command the processor to loop back to the start of the trace file and repeat it. This allows shorter benchmarks to be simulated repeatedly while a longer benchmark continues to run. The simulation terminates when all of the processor trace files have reached their ends with no more repetitions pending the trace buffers are empty.

Performing a cold start of 8 cores simultaneously generates 100% miss rates until the instruction and data caches are warmed up. This situation is unlikely to occur in a real system as the operating system takes some finite amount of time to create a process and initiate execution. This can be modeled in Moola by configuring a time delay by processor for when they will start processing the given trace file.

3 Moola Example

An example application was used to debug and test Moola. This application was to analyze the congestion at the LLC of the Intel Ivy Bridge processor in a 1 to 8 core configuration. There were 7 SPEC integer benchmarks [5] and 21 MiBench benchmarks [4] analyzed. The cache model was configured to match the Intel Ivy Bridge [A, I2] and main memory was configured to fast PC memory [10] The 7 SPEC integer benchmarks were: bzip2, gcc_166, gcc_200, gcc_typechk, gobmk, hmmer, and mcf. The 21 MiBench benchmarks were: adpcm_c, adpcm_d, basicmath, blowfish, charcnt, CRC32, dijkstra, fft, ffti, gsm_toast, gsm_untoast, jpeg_c, jpeg_d, patricia, qsort_int, qsort_text, sha, stringsearch, susan_c, susan_e, and susan_s. In addition to running each benchmark individually, a mixed benchmark consisting of gcc_typeck, gcc_200, basicmath, blowfish, dijkstra, ffti, jpeg_c, and susan_s in an 8 processor configuration. Some of the shorter benchmarks were reloaded when they completed to attempt to keep all 8 processors busy through most of the simulation.

Table 3 shows the results of the simulations as the average of the speedups achieved by each benchmark. This table also shows the difference between shared instruction memory and non-shared instruction memory. Sharing the instruction memory among the benchmark instances increased the cache hit rates which increased the speedup by 8%-17% for 3 or more instances.

Table 3: Average Speedups

Instances	Non-shared instr	Shared instr
2	1.76	1.81
3	2.21	2.49
4	2.60	3.05
5	3.17	3.64
6	3.60	4.17
7	4.19	4.51
8	4.46	5.00

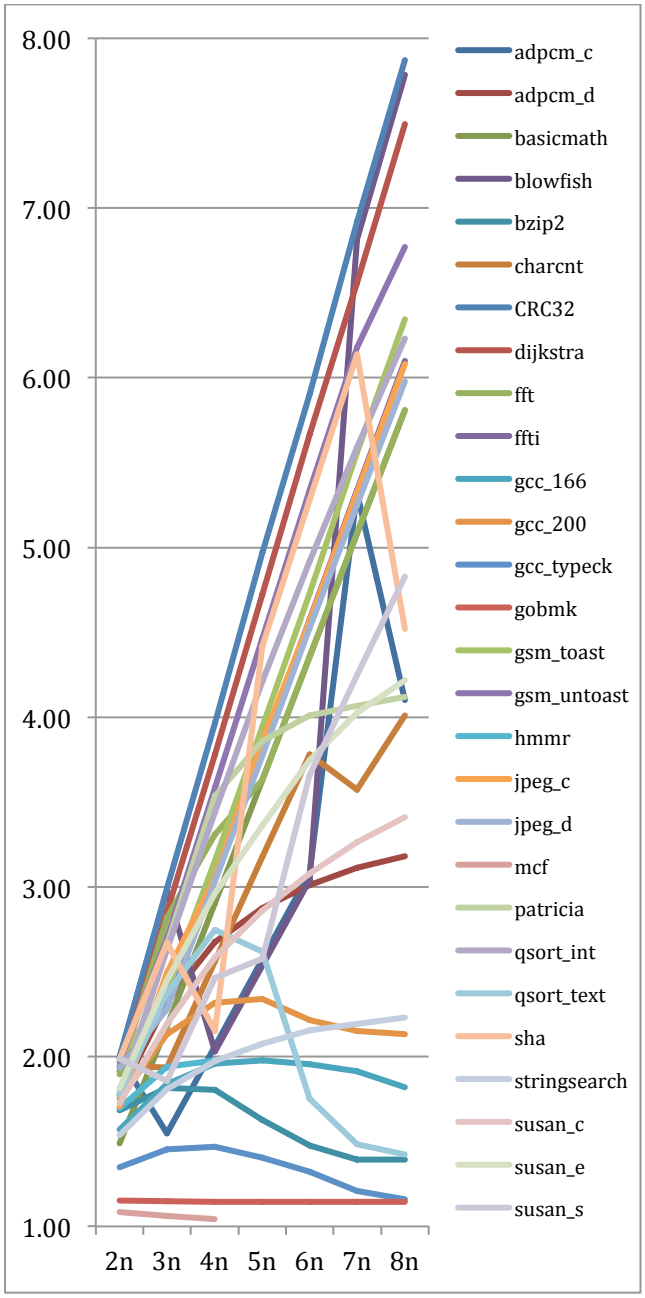

Figure 3. Speedup by Benchmark

The smaller benchmarks that were mostly contained within the L3 cache showed nearly linear speedup with increasing instance count and processor count. CRC32 and dijkstra are the primary examples of this. A couple of the "cache buster" applications such as hmmer and mcf showed no speedup improvement as more instances were added, indicating the performance was limited by accesses to memory even with only a single instance. Others, such as susan_e, patricia, and charcnt, started with a linear speedup with 2-4 processors and then leveled off with speedup values of 3 to 4. Eleven of the twenty-one benchmarks had speedups less than 4.0 when running on 8 processors. This can be seen in Figure 3: speedup by benchmark.

4 Accuracy Assessment

The accuracy of Moola was assessed by having a program spawn 1 to 14 processes and having each spawned process run the gobmk benchmark executable from the SPEC2006 suite. A mid-level MacPro system with 8 physical cores and 8 virtual cores (via hyperthreading) running at 3.0 GHz was used to collect this data. The runtimes of the 14 test cases were use to compute actual speedup values. Moola was then configured to match the cache size and timing parameters of the Intel(R) Xeon(R) CPU E5-1680 v2 used in the MacPro system. Moola then provided estimated speedups for the gobmk trace files with both blocking and hit-under-miss L3 cache configurations. These results are shown in Figure 4.

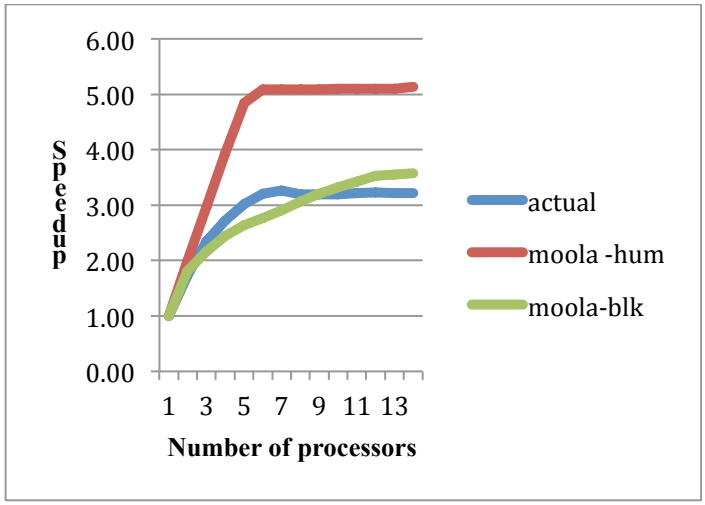

Figure 4: Moola estimated and actual speedups

The Moola estimated speedups for a blocking level 3 cache are fairly similar to those obtained with actual measured timing. The Moola estimated speedups with a hit-under-miss L3 cache showed significantly better performance from 3 to 5 processors and then flattened when limited by the available memory bandwidth. We will be running more benchmarks through the accuracy analysis and will be adjusting the timing estimates to improve the correlation between the estimates and actual results.

5 Related Work

There are a variety of cache simulators available for research use. DineroIV [3] has seen significant use with over 600 citations in Google Scholar. This venerable cache simulator is limited to uniprocessor systems and is therefore not suitable for use in this application. CMP$im [9] is a good multicore cache simulator tool for the Pin binary instrumentation package from Intel. This tool provides miss rates and other statistics, but does not provide built-in timing. Since it is not trace-based, it requires running the benchmark for each configuration analyzed. It is a proprietary product that cannot be modified to provide different styles of LLC operation. Another Pin package from Ratanaworabhan [11] is an open-source project that would allow development of different LLC access architectures, however, it does not provide a built-in timing model and requires rerunning the benchmarks for each configuration analyzed. MCSMC [7] does provide a timing model with similar capability to Moola. However, the memory accesses used as input are from synthetic trace generators and the cache structure appears to be limited to a hierarchical binary sharing that does not reflect the interconnection methodology of current, common multicore processors. The work done by Tao [12] uses the Valgrind binary instrumentation tool set as the driver for the cache simulator. This tool does not include a built-in timing model nor does it provide the ability to vary the type of LLC access.

6 Conclusions

This paper introduces Moola, an extensive cache simulator for multicore systems. Moola is an open source trace-based cached simulator providing a built-in timing model for cache-memory performance estimation. Moola also allows different LLC cache organizations and is highly configurable for modeling different cache architectures. An example analysis using Moola to demonstrate congestion effects at the LLC is provided. Moola models cache coherencies in a multicore system. The coherency protocol is selected during configuration at run-time and new protocols can be easily added. This analysis shows that Moola is a useful tool for students and researchers to explore multicore cache concepts and designs. More information on the status of Moola and the source code for Moola can be found in the "projects panel" at http://www.csrl.unt.edu/.

Acknowledgements

This research is supported in part by NSF award #1237417 and by industrial memberships of the NSF Net-Centric IUCRC. The authors wish to acknowledge the support given by Mike Ignatowski and Dave Mayhew of AMD for their suggestions and insights into cache-memory subsystems of current and future computer systems.

References

[1] AMD MultiCore Technology: http: //multicore.amd.com

[2] http://www.anandtech.com/show/4830/intels-ivy-bridge-architecture-exposed, September 17, 2011.

[3] Jan. Edler and Mark. D. Hill. "Dinero IV Trace-Driven Uniprocessor Cache Simulator, University of Wisconsion, http://www.cs.wisc.edu/~markhill/DineroIV

[4] Matthew R. Guthaus, Jeffrey S. Ringenberg, Dan Ernst, Todd M. Austin, Trevor Mudge, Richard B. Brown, IEEE 4th Annual Workshop on Workload Characterization, Austin, TX, December 2001.

[5] John. L. Henning, "Performance Counters and Development of SPEC CPU2006," Computer Architecture News, vol. 35, pages 118-121, March 2007

[6] IBM Cell Processor: http://www-306.ibm.com/chips /techlib/techlib.nsf/products/Cell

[7] Muhammad Ismail, Talat Altaf, and Shahid Mirza, MCSMC: A New Parallel Multi-level Cache Simulator For Multi-core Processors. In *Proceedings of Electronics, Communications and Photonics Conference (SIECPC)*, pages 264-269, King Abdul Aziz City, Saudi Arabia, April 27-30, 2013.

[8] InfoWorld.com, "Intel brings Haswell micro-architecture to servers with Xeon E3 chip" http://www.infoworld.com/d/computer-hardware/intel-brings-haswell-microarchitecture-servers-xeon-e3-chip-216138, April 10, 2013

[9] Aamer Jaleel, Robert S. Cohn, Chi-Keung Luk, and Bruce Jacob, CMP$im: A Pin-Based On-The-Fly Multi-Core Cache Simulator, Fourth Annual Workshop on Modeling, Benchmarking and Simulation (MoBS), 2008

[10] NewEgg.com, G.SKILL Sniper Gaming Series 32 GB DDR3 SDRAM memory, 9-9-9-24 timing, http://www.newegg.com/Product/Product.aspx?Item=N82E16820231610, fastest available memory on NewEgg website, April 25, 2013.

[11] Paruj Ratanaworabhan, Functional Cache Simulator for Multicore. In *Proceedings of International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON),* pages 661-664, Hua Hin, Thailand, May 16-18, 2012

[12] Jie Tao, Marcel Kunze, Wolfgang Karl, Evaluating the Cache Architecture of Multicore Processors, In Proceedings of 16th Euromicro Conference on Parallel, Distributed and Network-Based Processing, pages 12-19, Toulouse, France, February 13-15, 2008

[13] Richard A. Uhlig. And Trevor N. Mudge. "Trace-driven Memory Simulation: A Survey", In ACM Computing Surveys, pages 128-170, Vol. 29, June 1997