Trace Filtering of Multithreaded Applications for CMP Memory Simulation

Alejandro Rico, Alex Ramirez, Mateo Valero
Barcelona Supercomputing Center - Centro Nacional de Supercomputación
Universitat Politècnica de Catalunya - BarcelonaTech
{first.last}@bsc.es

***Abstract*—Recent works have shown that modelling the performance of out-of-order superscalar cores is doable using filtered memory traces for single thread simulations. However, those techniques do not account for cache coherence actions so they cannot be used reliably in multithreaded scenarios. In this paper, we leverage the structure of parallel applications to propose a simulation methodology that enables the use of filtered memory traces for the simulation of multithreaded applications on multi-core architectures. In our experiments our proposal reduced the simulation error of state-of-the-art techniques in 39% on average, while only losing 9.5% of simulation speedup.**

I. Enabling Trace Filtering for Multithreaded Applications

Recent works [1], [2] have shown significant speed-ups by using memory trace simulation and trace filtering techniques such as trace stripping [3], [4] for the simulation of single-core/single-thread computer architectures. Using these techniques, they reduce trace size and simulation time by not including L1 hits and only simulate L1 misses. They generate a trace including all L1 misses of a benchmark running on a given microarchitecture, and then use it to feed a trace-driven simulator for exploring the design of the *non-core* components, such as cache hierarchy—L2 and above—, and off-chip memory. This methodology assumes a fixed core microarchitecture and L1 cache for all trace-driven simulations. This is a significant limitation for single-core/single-thread analysis, as the non-core design space is limited. This kind of methodology seems more attractive for multi-core architecture simulations where the non-core design space is larger and a higher simulation speed is needed.

However, filtering a trace for multithreaded application simulations is not straightforward. Trace stripping consists of filtering out all L1 hits and just store L1 misses in the trace. Thus, for a given core microarchitecture and L1 cache configuration, the non-core accesses—to the L2 cache—will be the same regardless of the non-core configuration. This holds true for single-thread applications where the accesses to L1 either hit or miss only depending on previous accesses, which are always in the same order for a given core microarchitecture. However, cache coherence actions in multi-core architectures, such as invalidations, may make an access to L1 hit or miss depending on other threads activities, which actually depend on the non-core configuration. As a result, some of the accesses that hit during the filtering process, may miss in a multi-core architecture simulation because of cache coherence actions.

In this paper we introduce the first attempt to extend trace filtering techniques to be used reliably on multi-core architecture simulations. Our proposal to solve the aforementioned problem is to take advantage of the synchronization happening between accesses from different threads to the same data. Whenever a thread wants to access shared data, it has to synchronize with other threads to avoid *race conditions*. Before a synchronization point (e.g., a lock acquire), other threads may have accessed the data, thus potentially invalidating the copies in the private cache. Our approach is then to reset (i.e., clear) the filter cache at trace generation time on every synchronization point. With this technique, every first access to a cache line after the synchronization point is recorded, thus avoiding the filtering of accesses that may potentially miss during simulation. Effectively, this methodology makes every piece of computation between synchronization points to be filtered separately, thus being independent of any operations occurring before and after its execution.

II. Methodology Implementation and Evaluation

We implemented this methodology to generate traces of multithreaded applications for TaskSim, a trace-driven multi-core architecture simulator [5]. We used applications written in the OmpSs task-based programming model [6] and generated traces that include events at the application level (for dynamic scheduling and synchronization of tasks during simulation) and the memory access level (for the cache hierarchy and memory architecture simulation) as in our prior works [5], [7]. For capturing the application-level events we instrument the OmpSs runtime system library and use Pin [8] for capturing the memory access traces. Our Pin instrumentation tool simulates a cache to filter the traced memory accesses. This filter cache is cleared using callbacks on every synchronization operation.

TaskSim implements a multi-core architecture with a two-level cache hierarchy, using an MSI coherence protocol, and a detailed model of the on-chip memory controller and the off-chip DDR-DRAM memory modules. We employ the *memory* core model of TaskSim which is based on the ROB Occupancy Analysis (ROA) model [2]. This model maintains the state of the re-order buffer (ROB) and allows issuing only those memory accesses that fit in the ROB. This model assumes that the ROB is the main performance limiting factor in the core pipeline. For our methodology, and contrarily to previous works [1], [2], we actually simulate the L1 caches in order to account for cache coherence actions.

We evaluated our implementation in terms of accuracy and simulation speedup for three different filtering approaches. The first is the naive approach, proposed in prior work [1] and labelled *naive*. It ignores cache coherence actions and filters

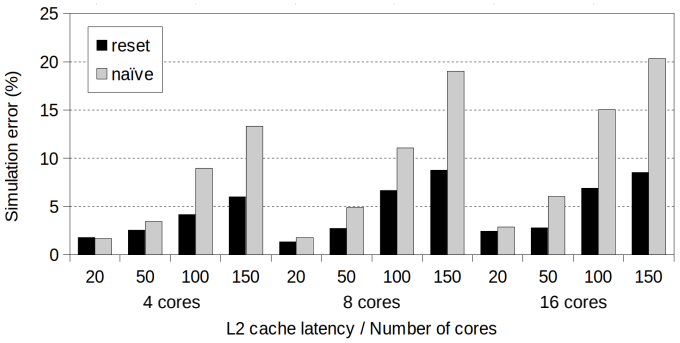

Fig. 1. Simulation accuracy. Our proposal (reset) consistently reduces the simulation error by 39% on average compared to the state of the art (naive).

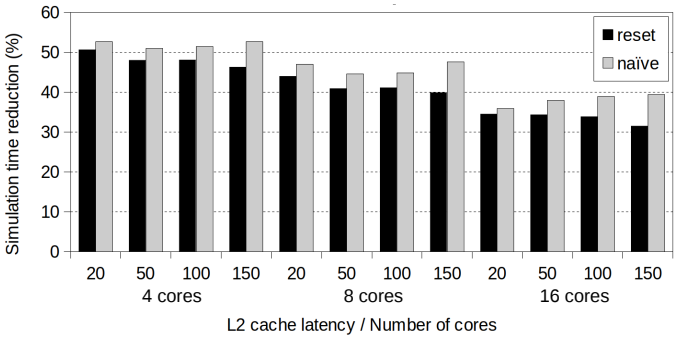

Fig. 2. Simulation speedup. Our proposal (reset) just loses a 9.6% of simulation speedup on average compared to the state of the art (naive).

the trace without clearing the filter cache at any point. The second is our proposal, labelled *reset*, which clears the filter cache contents on every synchronization point. And the third is the full/non-filtered memory access trace, which includes all memory accesses and is the baseline in our experiments.

We generated traces for a set of scientific kernels: LU decomposition, Cholesky decomposition, array reduction, vector addition and matrix multiplication. These scientific kernels are widely used in high-performance scientific applications and cover different levels of data reuse, parallelism and computation-versus-memory ratio. We used the generated traces to simulate multiple numbers of cores (to see the effects of parallelism), multiple L2 cache latencies (to see the effects of increased L1 invalidation penalties), and for multiple chip frequencies (to see the effects of an increased penalty for off-chip memory accesses).

Figure 1 shows the simulation error introduced by trace filtering in both *reset* and *naive* taking the full-trace simulation as a reference. The bars show the average of the absolute errors across all benchmarks and across two different chip frequencies: 1 GHz and 2 GHz. The X-axis shows groups of L2 cache latencies for different numbers of cores. Although the largest latencies are unrealistic, we want to show these extreme cases as an upper bound for simulation error, where L1 misses have a large penalty, even if they hit in L2.

We expected the results of *reset* to be closer to the simulation of the full trace. However, we found a lot of variability during the simulations due to dynamic scheduling decisions. Slight differences in timing due to filtering, may make tasks to be scheduled to different threads on different configurations, which leads to completely different data reuse patterns in the private caches. Further investigation is needed to achieve deterministic scheduling of tasks during simulation to get better comparable results across simulation methodologies. However, even considering these effects, our methodology consistently reduces the simulation error incurred by the incorrect filtering of the state-of-the-art *naive* methodology. The upper bound error of our methodology is 9%, while naively filtering the applications may incur a simulation error over 20%.

Figure 2 shows the simulation time reductions of *reset* and *naive* compared to using the full trace. The results show the average across all benchmarks and chip frequencies. Although our methodology includes more accesses in the trace due to having more misses during the filtering process as a result of clearing the filter cache on inter-thread synchronizations, the average loss in simulation time reduction is just 9.5% with an upper bound of 20%.

III. Conclusion

This paper presents the first attempt to enable memory trace filtering for simulating multithreaded applications running on multi-cores. We have implemented our methodology based on a task-based programming model, and used dynamic binary instrumentation to generate the full and filtered traces to feed in our trace-driven multi-core simulator. Despite the variability and non-determinism of the simulations, our methodology consistently reduces the error incurred by state-of-the-art techniques being 39% more accurate on average, while only losing 9.5% of simulation speedup over using the full trace.

Acknowledgment

This research was supported by the Consolider program (TIN2007-60625) from the Spanish Ministry of Economy and Competitiveness, the ENCORE project (ICT-248647), and the European Network of Excellence HIPEAC-3 (ICT-287759). We thank Nikola Puzovic, Paul Carpenter, Milan Pavlovic and Thomas Grass from Barcelona Supercomputing Center, and Carlos Villavieja from University of Texas at Austin for their help to improve the quality of this paper.

References

[1] H. Lee et al., "Two-phase trace-driven simulation (TPTS): a fast multi-core processor architecture simulation approach," *Software Practice and Expertise*, vol. 40, pp. 239–258, 2010.

[2] K. Lee et al., "Accurately approximating superscalar processor performance from traces," in *ISPASS'09*, 2009, pp. 238–248.

[3] T. R. Puzak, "Analysis of cache replacement-algorithms," Ph.D. dissertation, 1985.

[4] W.-H. Wang and J.-L. Baer, "Efficient trace-driven simulation method for cache performance analysis," in *SIGMETRICS'90*, 1990, pp. 27–36.

[5] A. Rico et al., "On the Simulation of Large-Scale Architectures Using Multiple Application Abstraction Levels," *ACM Trans. Archit. Code Optim.*, vol. 8, no. 4, pp. 36:1–36:20, 2012.

[6] A. Duran et al., "Ompss: a Proposal for Programming Heterogeneous Multi-Core Architectures." *Parallel Processing Letters*, vol. 21, no. 2, pp. 173–193, 2011.

[7] A. Rico et al., "Trace-driven Simulation of Multithreaded Applications," in *ISPASS'11*, 2011, pp. 87–96.

[8] C.-K. Luk et al., "Pin: Building customized program analysis tools with dynamic instrumentation," in *PLDI'05*, 2005, pp. 190–200.