On the Performance Consequences of Cache Configurations in Physics Simulation

Nathan Beckmann

Abstract

Chip multiprocessors (CMPs) have allowed for high levels of thread-level parallelism in applications. As the number of cores increase, the demand on the memory structure has increased as well; both for data needs and communication between cores. Many different configurations of the on-chip storage are possible by changing the size, number of ports, and/or sharing policy of the cache.

In this paper we explore the performance impact of various cache configurations for multi-core architectures. In particular, we use PhysicsBench, a benchmark suite of physics simulations designed to emulate the workload of an interactive entertainment application [*reference*]. This workload is a good choice because it exploits parallelism in many key parts of the code and translates well to a multi-core environment.

This workload has threads working on small segments of data independently. As such, there should be little communication between threads. I expect the best performance from a configuration using smaller, faster L2's backed by a larger L3. If this is the case, then it remains to be seen at what point the trade off deteriorates, and if this configuration translates well to other applications.

Introduction and Motivation

Chip multiprocessors are a method of using transistor real estate by placing several cores on a single chip. These chips differ from having multiple independent processors because the extra-core components can be shared amongst cores in many different configurations. Different applications have led to vastly different successful designs, such as IBM's Cell [*reference*], Intel's Core [*reference*], Intel's upcoming Tulsa [*reference*], Sun's Niagara [*reference*] and graphics processor architectures [*reference*].

As cores are added to CMPs, the memory structure becomes increasingly important to the performance of the processor. Among other reasons, this is because the additional cores require proportionally larger memory bandwidth. Also, the cache can be used as a mechanism for communication among cores, increasing traffic and contention within the chip itself. This paper focuses on the possibilities for dividing the cache above the L1 among cores—it is generally assumed that the L1 caches are private to each core. The trade off involves several factors: the size of the cache available to a single core, the latency of the cache, and the overhead in maintaining consistency between caches. A design that uses a private L2 cache for each core has the advantage of a smaller, simpler cache with lower latency. These designs can be fast, but usually result in poor aggregate cache usage and a large number of misses to memory. Designs that use a single shared L2 solve this problem, but at the cost of higher cache latency.

Alternatively, hybrid designs exist that use multiple L2's local to each core, but with some storage from each cache shared amongst all cores in an attempt to improve aggregate cache usage. This gives low latency for locally stored data, but requires requests to be sent across the chip for distant data. These designs complicate the routing network on the chip and make wire delay and dominating factor in cache latency [*6, 8, and 35 in CC*]. Improvements upon this premise, such as Cooperative Caching [*reference*], selectively share data that is needed among cores and simplify the connection framework between caches. In this study we only investigate a static partitioning of the cache, but more complicated schemes exist that dynamically partition the cache [*reference*]. In particular, PDAS provides cache space to cores based on demand and performance thresholds [*reference*]. While these schemes are more complicated, they can have significantly improved performance over statically partitioned methods; for example, PDAS showed a single-thread performance improvement of 26% and 27% over private and shared L2 schemes.

In this study, we investigate the performance of several different cache configurations: (1) private L2's for each core, (2) a single shared L2, (3) L2's shared between pairs of cores, (4) small L2's shared between pairs of cores backed by a large L3. Our application for these simulations is PhysicsBench – a set of physics simulations that emulate the workload of an interactive entertainment application [*reference*]. The contribution of this study is the performance impact of various cache configurations for physics simulations running on a typical CMP architecture. Specialized architectures exist for physics, such Cell and ParallAX [*reference*], but these architectures are highly adapted to physics simulation. For example, the Cell replaces the cache with a fully programmable memory. Furthermore, physics simulations have unusual memory access patterns that could produce much different results than a standard benchmark such as SPEC.

I expect the best performance from the private L2's with a backing L3 cache. This is because each core works on a small, independent set of data during its computation. By keeping the L2's private to only a few cores, we can reduce the hit latency while keeping a low miss rate. Furthermore, the L3 will provide a relatively fast way of retrieving any missing data. I expect that in this application, private L2's will perform admirably for the same reason, but a shared L2 will suffer.

Approach

We model the system using the SuperESCalar simulator (SESC) [*reference*]. SESC models the MIPS ISA, using MINT as a functional simulator with a library that handles most OS calls. This eliminates the need for a full OS running in the simulator and speeds up simulation significantly, at the cost of some OS functionality.

As mentioned, we use the PhysicsBench suite to test performance. These applications use the Open Dynamics Engine (ODE) to model physics [*reference*]. Tests within the suite use different features of ODE in different scenarios to test the full range of ODE's functionality. The physics simulation has two key phases: collision detection and world-step. These are divided among four helper cores

while the remainder of the simulation is run on a 'master' core. It is the performance of the helper cores that we are interested in, as these cores show the interaction of the cores and cache hierarchy.

The cache configuration is specified in the SESC configuration file. Each configuration is identical except for the parameters of the cache. The cache is kept at roughly 4MB total for the chip, although slight variations are necessary due to limitations in SESC. For each test, the following parameters were used:

Private L2: Four 1MB caches with a latency of 8 cycles.

Shared L2: One 4MB cache with a latency of 15 cycles.

Shared-Pairwise L2: Two 2MB caches (shared between pairs of cores) with a latency of 12 cycles.

Shared-Pairwise L2 and L3: Two 512KB L2 caches with a latency of 6 cycles backed by a 4MB L3 cache with a latency of 20 cycles.

Then a simulation is run with several tests for each configuration. SESC outputs a plethora of statistics for the execution. We compare performance using the overall execution time of the four helper cores and the miss rates and access times to the cache.

Results

This will be completed as tests are run. Initial results indicate that including an L3 cache improves performance.

Conclusion

We will conclude based on the complete results.

References

- Tom's benchmark paper
- Tom's PDAS “Fast and Fair” paper
- Cooperative Caching?
- http://en.wikipedia.org/wiki/List_of_Intel_Xeon_microprocessors