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 [1]. 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. It should, therefore, respond well to small, fast cache configurations that provide local storage to each core. Results indicate that this is highly dependent on the memory footprint of the particular application, and more importantly, that memory system accounts for a small percentage of application performance.

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 [4], Intel's Core [5], Intel's Tulsa [5], Sun's Niagara [6] and graphics processor architectures [7].

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. Improvements upon this premise, such as Cooperative Caching [3], 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. In particular, PDAS provides cache space to cores based on demand and performance thresholds [2]. 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 [1]. 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 [8], 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.

Approach

We model the system using the SuperESCalar simulator (SESC) [9]. 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 [10]. 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 'worker' 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 for the four 'worker' cores. The 'master' core is identical for all tests; it is a 5 GHz “super-processor” that feeds jobs to the workers. The cache for the 'worker' cores 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.
Perfect Caching: In this test, we use a L2 that never misses with a latency of 1 cycle. This test gives us context to see what performance could be achieved with an idealized cache.

Due to simulator problems that could not be resolved in our time budget, only two tests from the PhysicsBench suite were run to produce the results. The first test (Test A) is a simpler test with low resource requirements, whereas the second test (Test B) involves several objects as well as cloth simulation, requiring higher resources.

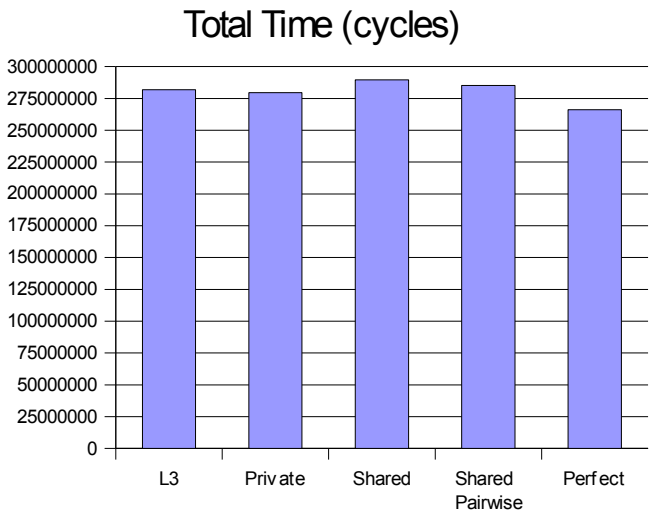
Results

After each simulation, SESC produces a file containing a plethora of statistics. We are going to consider the overall performance of the application, as well as some cache-specific performance metrics.

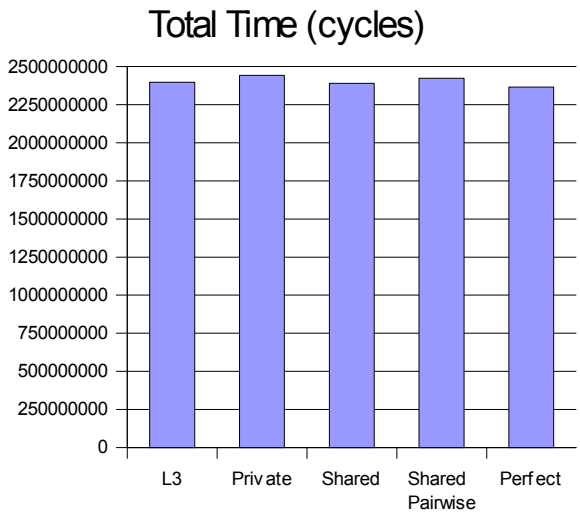
When discussing the results, we only focus on the results for the worker cores, as the simulation statistics showed that the performance for the master core was essentially identical across configurations.

Overall Performance: These graphs show the total execution time by number of cycles. The number of cycles is the average number of cycles executed by each worker thread.

Test A



Test B



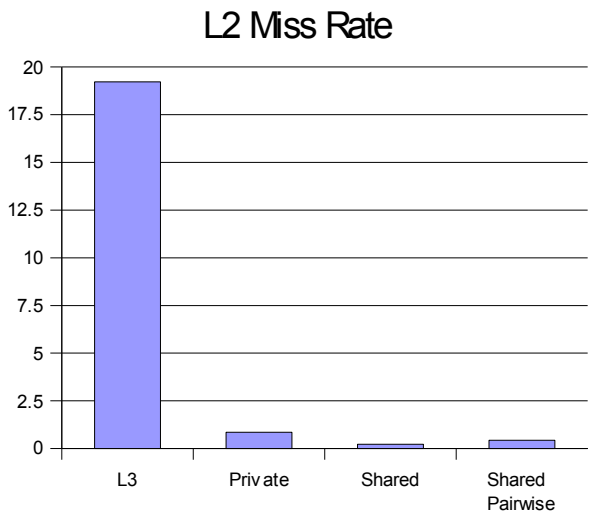
These tests show surprisingly little variation between configurations. No measurement varies more than 2% from the mean. The perfect-caching configuration outperforms the best realistic configuration by only 6.3%. This indicates that the memory system is not the bottleneck for these applications, and the amount of performance improvement possible is limited. In order to see how the different caches performed, we now consider their performance in isolation.

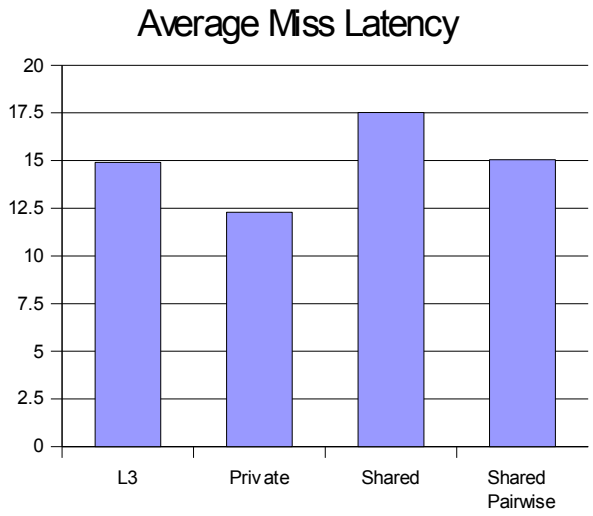
Cache Performance: We judge the performance of the cache based on the average miss latency of the L2 cache and the miss rates of the caches through the hierarchy. These values are the average of the rates for each worker core. As a point of interest, we also show the overall number of misses to memory, as the sum over all worker cores.

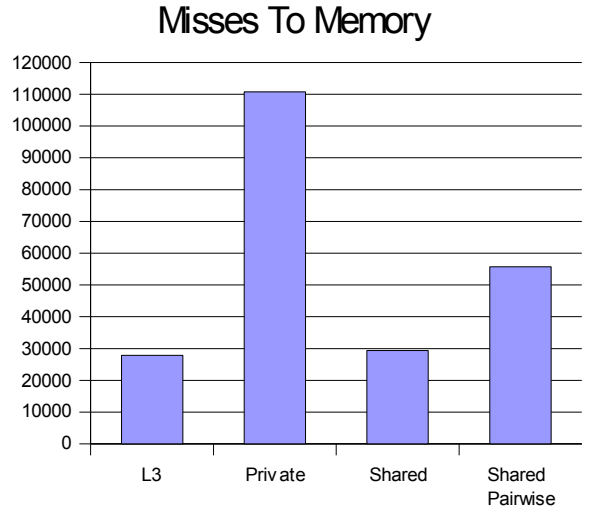
Some results are not shown below. The master cores for each test performed essentially identically, as did the L1 caches on the worker cores. The miss rates for the L3 cache in the L3 configuration are:

Test A – 2.22%
Test B – 22.47%

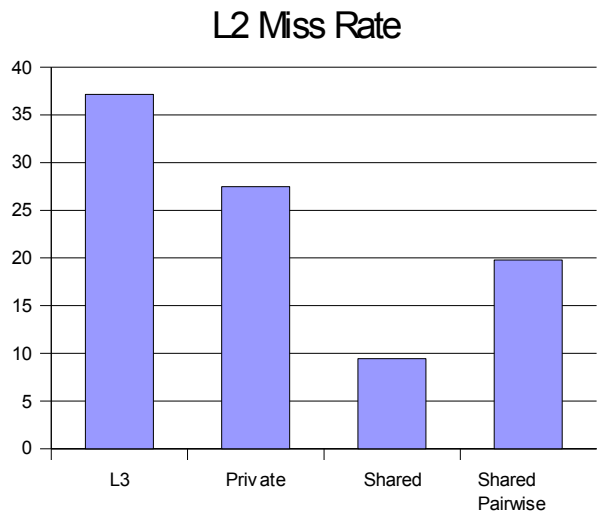
Test A

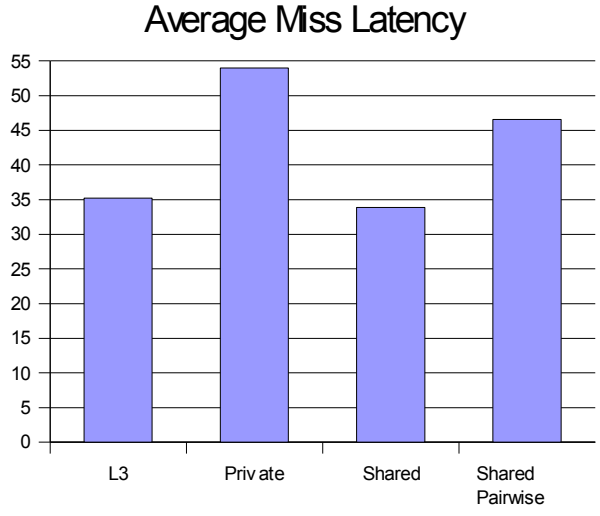

L2 Miss Rate
20
17.5
15
12.5
10
7.5
5
2.5
0
L3
Private
Shared
Shared Pairwise

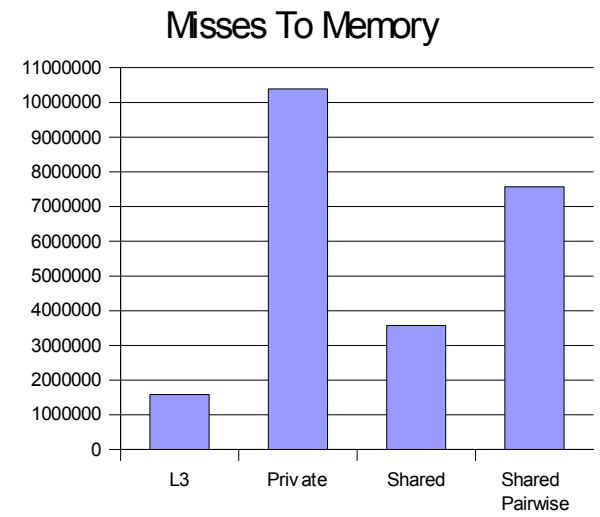

Average Miss Latency
20
17.5
15
12.5
10
7.5
5
2.5
0
L3
Private
Shared
Shared Pairwise


Misses To Memory
120000
110000
100000
90000
80000
70000
60000
50000
40000
30000
20000
10000
0
L3
Private
Shared
Shared Pairwise

Test B


L2 Miss Rate
40
35
30
25
20
15
10
5
0
L3
Private
Shared
Shared Pairwise

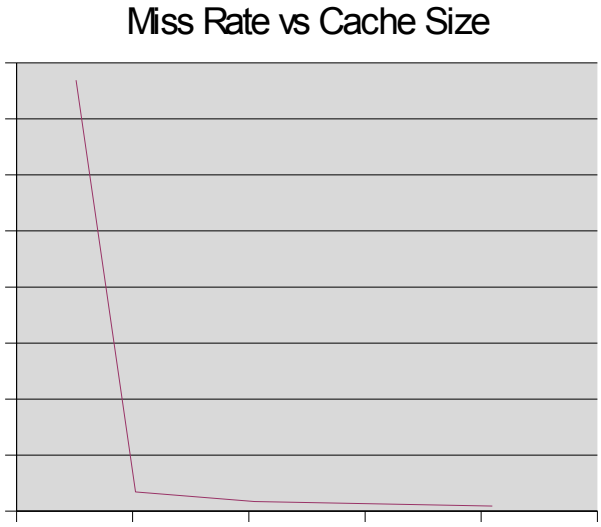

Average Miss Latency
55
50
45
40
35
30
25
20
15
10
5
0
L3
Private
Shared
Shared Pairwise


Misses To Memory
11000000
10000000
9000000
8000000
7000000
6000000
5000000
4000000
3000000
2000000
1000000
0
L3
Private
Shared
Shared Pairwise

Many of these results match our expectations. The misses to memory are highest for private L2 caches and lowest for shared L2 caches. The shared-pairwise configuration falls somewhere in between, which is not surprising because the size of the L2 in this scheme lies between the private and shared schemes. The L3 scheme shows the lowest number of misses to memory, also unsurprising because the L3 cache and the shared L2 cache are both 4MB, but the L3 scheme has the highest aggregate cache size (see *Approach*).

The most directly relevant statistic to the applications overall performance is the average access time of the cache, which is shown by the Average Latency graphs above. A good metric for the performance of the cache itself is the miss rate, or what percentage of accesses are unable to be served and drop to a higher level in the hierarchy. The variance in cache performance is much more exaggerated than the variance overall, and shows the merit of each scheme.

In Test A, the private configuration performs the best, and the shared configuration the worst. This is because Test A requires few resources, and the memory footprint fits comfortably in the 1MB L2 caches of the private scheme. Conversely, the shared scheme suffers extra latency with no performance benefit. As expected, the shared-pairwise scheme performs decently, somewhere between the private and shared. The performance of the L3 scheme is surprising; it fairs poorly, shown dramatically in the miss rate of its L2 cache. For the other three configurations, the miss rate is roughly inversely proportional to the size of the cache, for the L3 configuration the miss rate sky-rockets:



This is because the L2 cache of the L3 scheme is too small to hold the footprint of even this simple application. This makes the performance of the scheme as a whole suffer. Its miss rate is comparable to that of the private scheme for Test B, which suffered from the same problem.

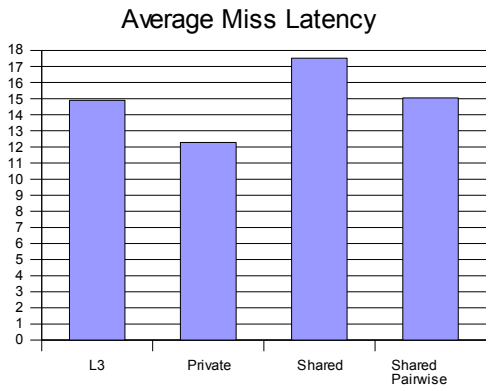
In Test B, the resource demands are higher, and the miss rates of all caches increase. In particular, the private configuration performs poorly and the shared

scheme performs well. This is because the memory footprint of the test no longer fits into the 1MB privately-shared cache. Once again, the shared-pairwise scheme performs decently.

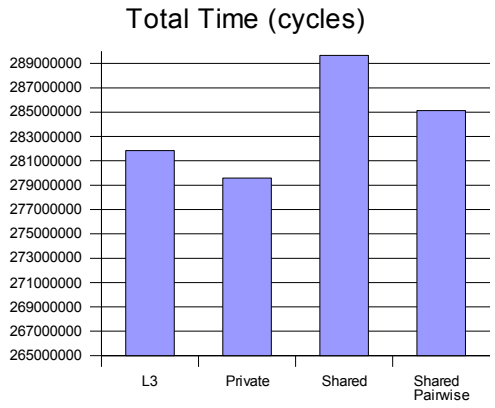
In this test, the L3 configuration gains ground and is very close to the performance of the shared scheme, despite its over 35% L2 miss rate. This is because with the miss rate of all other caches increasing, the L3 comes into play and provides a distinct advantage – only 20 cycles are required to access the L3, which catches most misses from the L2, as opposed to 100 cycles to access main memory.

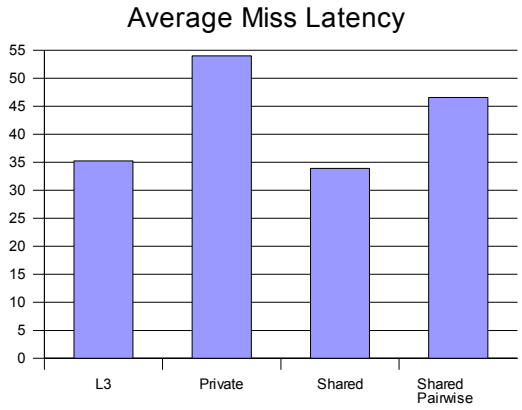
Cache Performance vs. Overall Performance:

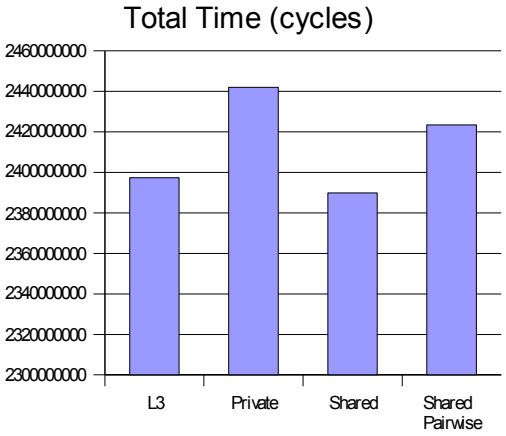
Test A



Test B







By changing the scale of the total time graph, we observe definite correlation between the average miss latency and the overall performance. But as noted above, an 18% variance in cache performance accounts for less than 2% variance in overall application performance.

Conclusion

These different configurations show that, even under perfect circumstances, memory accounts for a small portion of the overall performance of these physics simulations. Effort might be better spent in speeding up the execution of the code on the cores, or increasing the number of cores, rather than focusing on the cache configuration.

Regardless, we learned important properties of these application's interactions with memory. Depending on the demand of the application, the private and shared L2 schemes perform very differently. The shared-pairwise scheme, however, is consistently decent.

The L3 scheme did not perform up to expectation, but this is possibly the fault of the parameters chosen, and not a flaw in the scheme. By making the L2 cache smaller than 1MB, we could not hold the memory footprint of even a simple application. In Test B, the L3 configuration performed quite well, as other configurations were unable to hold the application's memory footprint. A direction for further exploration is the performance of the L3 configuration with the L2 increased to hold the memory footprint of these tests. Furthermore, these results suggest that the memory footprint of the two tests are quite different, and they might respond to well to a dynamic-sharing scheme, such as PDAS.

References

1. Thomas Yeh. PhysicsBench. http://www.cs.ucla.edu/~tomyeh/Papers.htm.
2. Thomas Yeh, Glenn Reinman. Fast and Fair: Data-stream Quality of Service. International Conference on Compilers, Architecture, and Synthesis for Embedded Systems (CASES), Sep 2005.
3. Jichuan Chang, Gurindar S. Sohi. Cooperative Caching for Chip Multiprocessors.
4. IBM Cell, http://www.research.ibm.com/cell/
5. Intel Processors, http://www.intel.com and http://en.wikipedia.org/wiki/List_of_Intel_Xeon_microprocessors
6. Sun Niagara, http://en.wikipedia.org/wiki/UltraSPARC_T1
7. GPUs, http://en.wikipedia.org/wiki/Graphics_processing_unit
8. Thomas Yeh, Petros Faloutsos, Sanjay Patel, Glenn Reinman. The 3th Internation Symposium on Computer Architecture (ISCA-34), June 2007.
9. SESC, http://iacoma.cs.uiuc.edu/~paulsack/sescdoc/
10. ODE, http://www.ode.org