Natalie Enright
Dana Vantrease
CS 838
Final Report

To Include or Not To Include:

The CMP Cache Coherency Question

Introduction

Chip Multiprocessors (CMPs) have several characteristics that raise interesting questions pertaining to chip design, especially design related to the memory hierarchy. With several processing nodes and their respective caches on-chip, the division and structure of off-chip and on-chip memory has more flexibility and variety than previously studied processor systems. In addition, an increased number of processing nodes on-chip independently generating a number of memory requests implies that off-chip pin bandwidth becomes an expensive commodity. Cache coherence protocols are directly affected by the described circumstances and are the focus of this paper. In particular, the tradeoffs between inclusive, exclusive and non-inclusive policies in a single CMP with a shared L2 are studied in detail. Throughout the study, a focus is given to minimizing off-chip bandwidth while sustaining or improving performance on-chip

Three intuitively obvious coherence protocol schemes were assessed in this study: Inclusion, Non-inclusion, and Exclusion. Multilevel inclusion is maintained when the contents of the L1 caches are a subset of the L2 caches. Therefore, if a block is replaced in the L2 cache due to a conflict or capacity miss, that same block must be evicted in all of the L1's in which it is present. In the case of a single processor chip, this can only be the instruction or the data cache but as the number of cores increases, the evicted block could be present in multiple data and/or instruction caches. Exclusion is the other extreme. From a processing node's perspective, the data that is present in its L1 cache cannot be present in the L2 cache. Non-Inclusion lies in between the two. A block can be present in both the L1 and L2 or one or the other.

The most commonly implemented policy in processors today is inclusion. IBM carried this tradition into the realm of CMPs by implementing it in the IBM Power4 CMP (Barosso et al). Conversely, however, Compaq's CMP design, the Piranha (Tendler et al), chose a policy of exclusion. It is worth noting, that with the sizes set by the L1 and shared L2, if an inclusion was specified, the L2 would be entirely taken up with L1-replicated data. To utilize cache space, an exclusion policy was specified and shadow tags were added to the L2, effectively doubling the on-chip cache size. The shadow tags contained information about what data blocks were contained the L1's. The differing opinions within industry are evident, and thus deserve scientific evaluation.

The goal of this project is to evaluate a range of cache inclusion policies to see which performed best in terms of miss statistics for a variety of commercial workloads. As a basis, a protocol that maintained strict multilevel inclusion, provided by the MultiFacet team, was used. From there exclusion and non-inclusion protocols were developed for evaluation. The non-inclusion and exclusion protocols were attempted, and though successful under a variety of configurations in the protocol tester running upward to 10 million memory transactions, they did not work in SIMICs. Shortly before the project deadline, a newly developed non-inclusion protocol by Mike Marty was used. Although this protocol is still buggy, it ran long enough to collect some data for 4 processor configurations. Our hypothesis is that as the ratio of L1 to L2 cache sizes increases, an inclusive protocol will begin to hurt performance making it worthwhile to consider a non-inclusive or exclusive protocol.

Cache Coherency Protocols

Inclusion:
The baseline protocol used maintained strict multilevel inclusion. This protocol was provided with the initial release of the simulator. Multilevel inclusion has been the standard in cache inclusion policies in processors to date with the exception of the aforementioned Piranha. The problem of conflict/capacity misses negatively affecting performance by evicting blocks from the L1 cache is much less significant in uniprocessors since there are only two L1 caches competing for space in the L2 cache. This limited competition in a uniprocessor does not warrant the increased complexity that comes with not enforcing inclusion. In this protocol, the L1 cache design is relatively simple and straightforward and more complexity exists at the L2 cache. The L1 cache has the following states: modified, shared and invalid. The L2 cache has the following states: modified, owned, owned with L1 sharers, shared, shared with L1 sharers and modified in the L1 but stale in the L2. The number of states required for the L2 cache in this protocol is slightly more than that of a traditional shared memory multiprocessor protocol.

Some of the benefits of this protocol are that it is simpler than the other two protocols presented below. Inter-chip communication is easier since all of the on-chip data is automatically contained with the L2 cache. A strict inclusive protocol does not require that remote chips snoop the L1 tags. The main drawback, which was described above, is the fact that multilevel inclusion does not effectively exploit the total available cache capacity on chip due to the conflict of data at the second level of cache.

Non-Inclusion:
The second protocol considered was a non-inclusion protocol. We spent a significant amount of time trying to develop our own working protocol, but Mike Marty was able to provide us with one before we had ours working. Learning SLICC presented a number of challenges and we were not able to overcome. This protocol added owned and exclusive states to the L1 cache. A set of L1 shadow tags were added at the L2 cache to provide coherency information to the other on-chip L1 caches as well as to external requests.

These shadow tags increased the complexity at the L2 level and required a significant number of additional states. The benefits of a non-inclusive protocol are that it increases the effective on chip cache storage and provides low latency cache transfers between L1 caches. Transfers between the L1 caches of different on-chip cores need to go through the L2 cache but this latency is still significantly less than having to travel out to memory.

This protocol also reduces the off-chip cache bandwidth requirements as more of the requested data is likely to be found on chip. However, this policy is not without its drawbacks. The additional communication from the L2 cache to the L1 cache for data consumes additional bandwidth and causes additional contention for the L1 cache ports. Non-inclusion also complicates write-back over an inclusion protocol. When a dirty block needs to be written back from an L1 cache, it is quite possible that the block is no longer present in the L2 cache. A writeback requires that a block be allocated for the data when it comes from the L1 cache. As mentioned before, the final drawback is the increased complexity in both the level one states and the addition of the shadow tags at the second level of cache.

Exclusion:
The third and final protocol explored was an exclusion based protocol. As briefly described earlier, from a processing node's perspective, the data exists in either the L1 or the L2, but never both simultaneously. Because blocks only exist in the L2 when they are expelled via capacity and conflict misses, the L2 effectively acts as a large victim cache with data in Modified, Exclusive, or Invalid State. Besides being a large victim cache the L2 behaves as a central point for synchronizing the L1s communication by having all requests and responses pass through it. Such a protocol gives the L2 the opportunity to observe all of the L1s requests for use in off-chip coherency communication and effectively directing on-chip requests and responses. When a request is received from an L1 at the L2, the data may exist in the L2, in the shadow tag cache of L1 tags, or somewhere beyond the on-chip memory hierarchy. In each case, the request is directed to the appropriate place and a data response is sent back. Depending on what information the L2 and shadow tags have, data sent to back to the requesting L1 arrives in Shared with another L1 Modified/Clean, Modified, or Exclusive In the transient period while requests are being sent and responses collected, the L2's data block remains held as a lock.

The main perk of the exclusion protocol, and why the authors suspect the Piranha used it, is its utilization of space on chip. Very little data is replicated, allowing a larger quantity of unique data to be present compared to the two alternative protocols presented in this study. In the scenario when a processing node is accessing data in another processing node's L1, 2 hops must be made to arrive at the data (L1a→ L2 → L1b), versus inclusion's 1 hop (L1a → L2). Though this is more expensive for the Exclusion protocol in this scenario, having the extra data on-chip saves at other times when the exclusion protocol may perform 2 hops to obtain the data while the Inclusion must go off-chip for data expelled from the L2 for capacity reasons.

The exclusion protocol, while very attractive for space utilization reasons, does have a handful of drawbacks. The position of a block, whether it is in the L1 or the L2, may be independent of how a single processing node is accessing it. That is, if 2 processing nodes are sharing a block, and one expels it from their L1 for capacity reasons, the other one, regardless of how heavily they may be using it, must also expel it, only to, perhaps, request it back into the L1 immediately after writing it back. In addition to this theoretical drawback, the implementation of the exclusion protocol is quite complex. The L2 may direct a request to get data from an L1, but when the request arrives there, that L1 may be in the processing of writing it back. Several solutions to this exist, such as requiring an L1 who wishes a writeback to put the L2 block in a transient 'lock' state before writing back, or immediately satisfying the data request, and then in some manner canceling the writeback that will arrive at the L2 (if the writeback were permitted to take place, and L1 and L2 could simultaneously hold the data).

Experiments

The experiments were designed to determine at what configuration each coherency protocol was preferable. For all of experiments, oltp ran for 50 transactions and jbb ran for 5,000 transactions. We varied the size of the L1 caches to change the ratio of the total L1 cache size to the total L2 cache size. All data is presented for a single chip, 4 processor CMP system. Given more time, it would be interesting to also vary the number of processors in the configuration but for this study our protocol implementation limited the number of processors ran and how long they ran. The goal was to find the design point at which having an inclusive protocol might hurt performance or where the performance of a non-inclusive protocol is significantly improved over the base inclusion case such that the complexity is warranted. Even for relatively small L1 caches, when there is a large number of cores that aggregate, L1 size can add up. In a 4-processor configuration, a 2 MB L2 cache was simulated. This cache size allowed use of reasonable L1 cache sizes that amounted to a significant percentage of the L2 cache size.

The second experiment ran was designed to approximate the performance of an exclusive protocol. A set of experiments was run that added the total aggregate L1 cache size to the L2 cache size to determine if the extra capacity given to the system by an exclusion policy would result in any improvement in the cache performance for a 4-processor and 8-processor configuration. The baseline simulation with 4 processors had 2 MB of L2 cache and was 4-way set associative. The pseudo-exclusion experiment added 512KB of cache to the second level. The 8 processor configuration was run with 4 and 5 MB of L2 cache.

Results

The first graph shows the misses per thousand instructions for the inclusive and non-inclusive protocol run for oltp and jbb. Each cluster of bars represents an increase in L1 cache size. Our L1 cache sizes range from 8KB to 256KB. Although 256KB is large for

a L1 cache the intention was to show the extreme data point where the aggregate L1 cache size was equal to the L2 cache size. There is a jump in miss rate for the inclusive protocol when the L1 to L2 ratio goes from 25 to 50 % for both benchmarks. These data points could be valid design points or just artifacts of the simulation. These simulations are non-deterministic and at the time of the presentation, time permitted a limited run of data, thus one cannot be completely convinced that this is reliable data. Subsequently simulations were rerun for one of the data points in questions, specifically the configuration when the L1 cache size is 50% of the L2 cache size for oltp. Here is a graph presenting the miss rates since adding the extra capacity should improve those; however the trend for the ruby cycles for each data point is similar to the trend presented in the miss rate graphs.

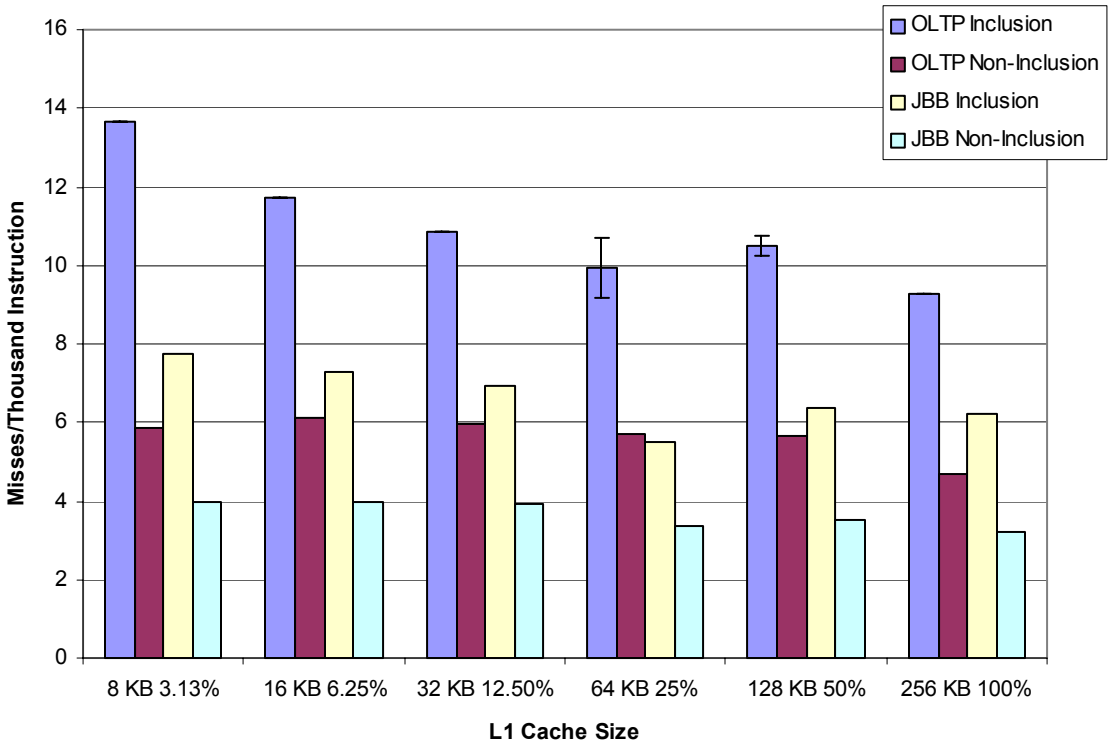

Figure 1: Misses per Thousand Instruction for Inclusion and Non-Inclusion protocols

One thing to note about the non-inclusive data points is that the miss rate presented only accounts for misses that need to go off chip to memory. Since the latency of these misses is the same as those misses presented in the inclusive case we felt that it was a fair comparison. The data does not account for local requests that miss in the L2 but can be satisfied by another on-chip L1. From the runs done, one can see from the inclusion protocol data that after the L1 cache size reaches 25% of the L2 cache size, the miss rate

starts to increase. We believe that at this point, and possibly sooner is when a non-inclusion protocol would be a better design choice. Also worth noting is that at all points, the off-chip miss rate of the non-inclusive protocol is better than that of the inclusion protocol so the question of implementing a non-inclusive protocol seems to be more one of complexity than performance. Is the benefit worth the additional design time and complexity? Figure 2 presents the same data as Figure 1 but the top part of the stacked bar has been added. This stacked bar represents the L1 misses that are satisfied locally by another L1. In some cases the total misses in the non-inclusion case are higher than in the inclusive case but this is believed to just be an artifact of simulation and more runs would produce data where the total misses in the inclusive case are closer to that of the base case. This data is presented because we felt it interesting to note that some of the stacked bars are significantly lower than the total miss rate in the base case. This indicates that by not enforcing inclusion, many misses are satisfied by the requesting processor's L1. Initially, the belief was this would often be the case. But as the results show, most of the stacked bars come very close to the base case indicating that a significant portion of the misses are satisfied by other on-chip processing node's L1s and not satisfied by the requesting processor's L1. This is a very interesting result because it indicates that keeping data on-chip longer will benefit more cores by giving them lower latency miss times.

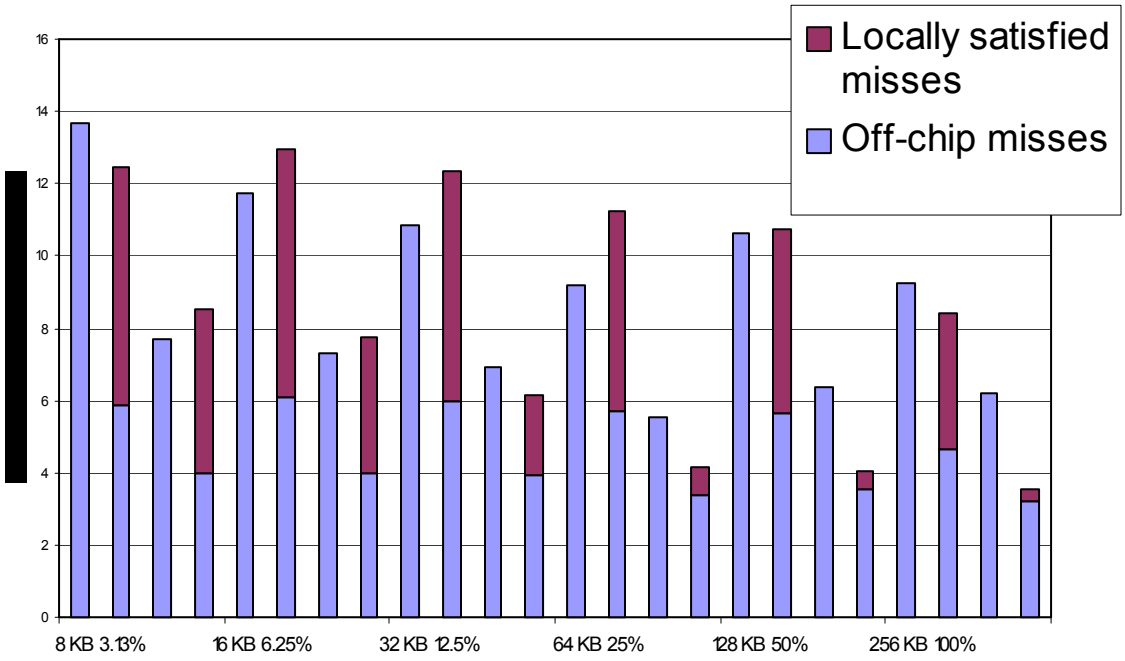

Figure 2: Miss rates including locally satisfied misses for Non-Inclusion

Figure 3 presents data from our pseudo-exclusion experiment described above. The question arises, why does OLTP see so little improvement from the extra capacity when it sees a more significant improvement from non-inclusion? Intuitively, the numbers should be more similar. There are two possibilities for this result. First is the previously mentioned non-deterministic nature of simulation and the data presented could be an artifact of simulation. The second explanation relates to the associatively of the caches

simulated. For the non-inclusive simulations since each of the L1 caches is 2 way associative and there are 8 L1 caches, we effectively add 16 ways of associatively to the on-chip cache storage. However, in order to get an odd cache size from SIMICs, we simply added on additional way of associativity to the L2 cache increasing its associativity from 4 to 5. This large discrepancy in the associativity makes it very difficult to compare these results and reach an accurate conclusion. Unfortunately, from the way in which ruby creates its caches, another way to generate an odd sized cache was not discovered, so this experiment best served the approximation without a working exclusion protocol.

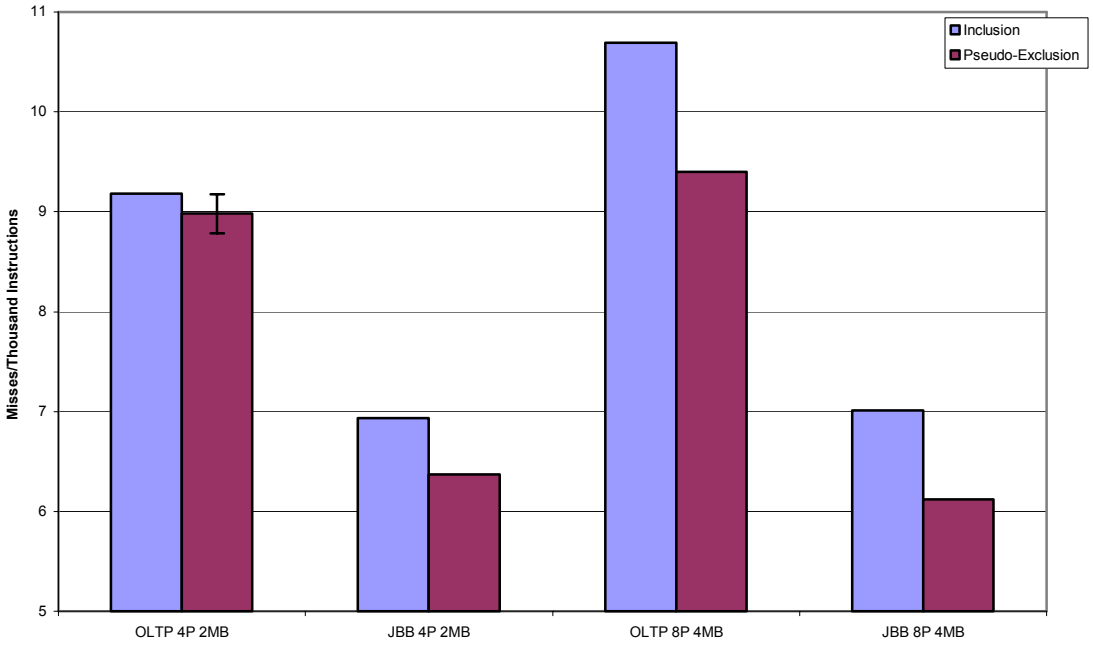

Figure 3: Pseudo Exclusion Experiment Results

Discussion

Evaluating inclusion policies for the caches is not as simple as improving miss rates and performance. There are several other issues that need to be considered when deciding to switch from a strict inclusion protocol to a non-inclusive protocol or an exclusive protocol. Two issues that might limit performance of a non-inclusive or exclusive protocol that were not modeled in our study are the L2 to L1 bandwidth and the ports in to L1. Sending additional requests from the L2 to the L1 because they cannot be satisfied by the L2 requires additional bandwidth and may limit the effectiveness of non-inclusion or exclusion. Also, the number of ports into the L1 could be a point of contention if the L2 is forwarded a significant number of requests to the L1 while that L1's processor is also trying to access the cache. Adding additional ports to the L1 might not be a desirable solution since adding ports increases the area of the cache and negatively impacts its access time.

As the protocols get further from inclusion and closer to exclusion, complexities arise. This is due to a combination of factors, the most basic is that the protocols were adapted from a provided inclusion protocol, and as more changes were made, the fragility of the protocol increased. In addition, the protocols theoretically become more complex the further from inclusion they are. Exclusion, for example, requires the L1's to synchronize with the L2's to enforce exclusion at all times. For example, an exclusion implementation must ensure that a writeback generated by an L1 does not violate exclusion when it finally arrives at the L2.

The presented study focused on single chip systems. This assumption was made for a few reasons. One, a clear target system was never specified, and it was assumed that CMPs have a viable market in single chip configuration. Secondly, and more importantly, the time was not available to explore all of the issues and code complexity associated with making our protocols work for multiple chips. Running in multiple chip configurations would have also made it difficult to study in isolation the on-chip effects of the protocols. Another interesting question arises when you add multiple chips. Does extending the life of a block on chip hurt the performance of remote chips? Seeking the answer to this interesting question has been left for future work.

Two of the three protocols studied use shadow tags. For the configuration studied, the shadow tags consumed from 1.4% of the total cache area (including tags, line information, and data) for the smallest L1s modeled to 22% of the total cache area for the largest L1s modeled. The results for the largest L1s modeled are comparable to Piranha's cache configuration and its published estimate of 25% of the cache area consumed by shadow tags (Barroso et al). Clearly, the larger the L1 and the more L1s present, the larger the tag cache. However, since the L2 behaves as a victim cache, perhaps there is a limit to its usability and the space is best spent in large L1s, despite the increased tag cache that goes along with it.

Limitations

We acknowledge that this study suffers from a few limitations. First, data is presented for only 2 workloads. While more workloads were attempted, not all of these runs completed successfully for a variety of reasons. A complete set of data points for both oltp and jbb were obtained and presented above. Secondly, the runs were shorter than anticipated due to time constraints of the presentation and final report. Thirdly, significant amount of time was spent coding in SLICC, which limited the time left to run simulations. As mentioned above, these runs are non-deterministic which means that a single data point is not necessarily representative. Due to time constraints most data points are only single runs and we feel confident that they represent an accurate trend for the data. However, a few data point were re-run multiple times to determine if the initial run was accurate or if it was incorrect due to an artifact of the simulation infrastructure. All of the above lead to mildly inconclusive results; however, we feel that the trends are

accurate and that this project was a great learning experience because it considered many of the issues involved in maintaining or not maintaining inclusion in the cache hierarchy.

Conclusions

The studies and analysis presented above have brought forth interesting observations and many avenues for future research. Notably, alternatives to a strict inclusive protocol are worthwhile avenues of research in CMP systems. A single-chip CMP system, similar to the one studied here, indicates that extending the life of a data block on chip and satisfying requests on-chip whenever possible in a seemingly efficient manner is worthwhile for the workloads evaluated. However, the complexity in the protocol's implementation is still a serious drawback and would benefit greatly from further research into a simple non-inclusive or exclusive protocol.

Although inclusion protocols in a single chip environment were studied, the authors speculate that the benefits seen in non-inclusion and exclusion will persist in a multi-chip environment at the cost of additional complexity and drawbacks. Weighing the performance benefit of such protocols versus the additional development complexity will be interesting further research.

References

L. Barroso, K. Gharachorloo, R. McNamara, A. Nowatzyk, S. Qadeer, B. Sano, S. Smith, R. Stets, and B. Verghese. Piranha: A Scalable Architecture Based on Single-Chip Multiprocessing. In Proceedings of the 27th Annual International Symposium on Computer Architecture (ISCA'00), pages 282--293, June 2000.

J. M. Tendler, J. S. Dodson, J. J. S. Fields, H. Le, and B. Sinharoy. POWER4 system microarchitecture. IBM Journal of Research and Development, 26(1):5--26, January 2001.