Proximity Coherence for Instruction Caches in Tiled CMP Architectures

Tareq Alawneh*[1], Chi Ching Chi*[1], Ben Juurlink*[1]

* *Embedded Systems Architecture (AES), Technische Universitat Berlin, Germany*

ABSTRACT

Directory coherence protocols have been employed in large-scale many-core architectures to maintain cache coherence as an alternative solution to snoopy coherence protocols. However, these protocols introduce an indirection through the directory to obtain the cache coherence information, thus increasing the cache miss latencies. Recent research results have shown that a high degree of code sharing exists among the cores in multicore architectures . We propose a proximity coherence protocol for the instruction memory, a scheme in which neighboring L2 caches on tiled multicore architectures are used to reduce the requests to the directory. By extending the MOESI-directory protocol with a proximity coherence for instruction cache, initial performance improvements of up to 65.6% are observed for a micro-benchmark whose instruction footprint exceeds the L2 cache size.

KEYWORDS: Proximity Coherence; Tiled CMP; Instruction Memory

1 Introduction

An important challenge for many-core architectures is maintaining cache coherence in an efficient manner. There are two dominant classes of protocols to maintain cache coherence in parallel computing systems: snoopy-based and directory-based protocols. Directory-based protocols are more scalable than snoopy-based protocols. Unfortunately, directory protocols introduce an indirection to obtain the coherence information from the directory which increases the cache miss penalty.

In this work, we present a proximity coherence protocol for instruction caches in tiled chip multiprocessor (CMP) architectures. In this protocol L1 instruction cache miss penalties are reduced by requesting the cache block from neighboring L2 caches in addition to the private L2 cache, before contacting the directory. This protocol improves performance in case multiple neighboring cores execute the same program code. Some workloads, such as online transaction processing (OLTP), exhibit a high degree of instruction reuse over multiple cores. Several prior studies observed this extensive sharing of instruction blocks among the multiple processor cores in different workloads [CWS06, LBE+].

[1]E-mail: {tareq.alawneh,cchi,juurlink}@aes.tu-berlin.de

2 RELATED WORK

Hossain et al. [HDH08] introduced a scheme where a direct access is performed to a predicted remote L1 cache, which likely contained the desired data, before requesting the directory. Williams et al. [WFM10] presented a protocol where an L1-D cache sends a request to the neighboring caches by using dedicated paths before contacting the directory.

Previous studies focused on improving the performance of the data caches. In contrast, our work aims at improving the instruction caches with a proximity coherence scheme. Specializing the coherence protocol for instruction caches allows us to exploit the read-only property of instruction memory, which allows for a less complex protocol and reduced hardware overheads.

3 PROXIMITY COHERENCE FOR INSTRUCTION CACHES

When an instruction miss occurs, the L1 cache controller sends out, in parallel, requests to the private L2 cache as well as the L2 caches of neighboring cores. In the meantime, the state of the missed code block is changed to a transient state to indicate that it is waiting for the responses from the private L2 cache and neighboring cores. If the private L2 or neigh boring cores 's caches have the required block they will reply with a hit, otherwise they will reply with a miss. Once the L1 controller receives a hit, the forwarded code block is copied in the private L1-I cache simultaneous by forwarding it to the core, and its state is changed to another transient state to indicate that the code block is now valid, but some responses are still not received. When the L1 controller has received all responses and at least one of them is a hit, then the state of the code block becomes shared. Otherwise a request to the directory for the required code block is issued.

4 EVALUATION

To validate and preliminary evaluate the proposed proximity cache coherence protocol, we integrated it into the Gem5 simulator [BBB$^+$] and executed a micro-benchmark using the simulator. This micro-benchmark has been designed with a purpose of generating a high L1-I miss rate. It is a mixture of jump (JMP) and no operation (NOP) instructions. Each jump instruction is padded with NOP operations to fill a complete cache block of 64 bytes. The offset of the JMP instructions is set to 64 in order to access each cache block with the minimum number of instructions. In the micro-benchmarks this instruction pattern is replicated to create the various program sizes from 4 to 512 kB. The entire program code is looped over by a specific count in order to have the same number of instruction cache block request for all micro benchmark program sizes, e.g., 8192 loops for 4 kB and 64 loops for 512kB. Each core executed the same micro-benchmark. We varied the size of the micro-benchmark (i.e. its instruction footprint) from 4KB up to 512KB.

Figure 1 shows the overall execution time improvements our proposed cache protocol achieves using the global on-chip interconnect to carry the proximity messages. The micro-benchmarks with instruction footprints smaller than 32KB, which fit in the L1-I cache size, as expected do not provide any improvement in the overall execution time (some of them achieve a slight speedup due to the reduction in the cold misses). However, the micro-

benchmarks with 64KB, 128KB, and 256KB instruction footprints, which fit in the L2 cache size (256KB), suffer from a slow down of up to 30% compared to the traditional MOESI directory-based coherence protocol. The latency to the neighbor cache is 15 cycles longer than the private L2, and combined with the the fact that no local L2 copy is created in case of a hit in a neighboring cache, the average proximity hit latency is higher than the average private L2 hit rate in the baseline approach. On the other hand, our proposed cache protocol reduces the overall execution time of the 512KB instruction footprint micro-benchmark by up to 65.5%. The proximity hits reduce significantly the requests to the directory for this program size.

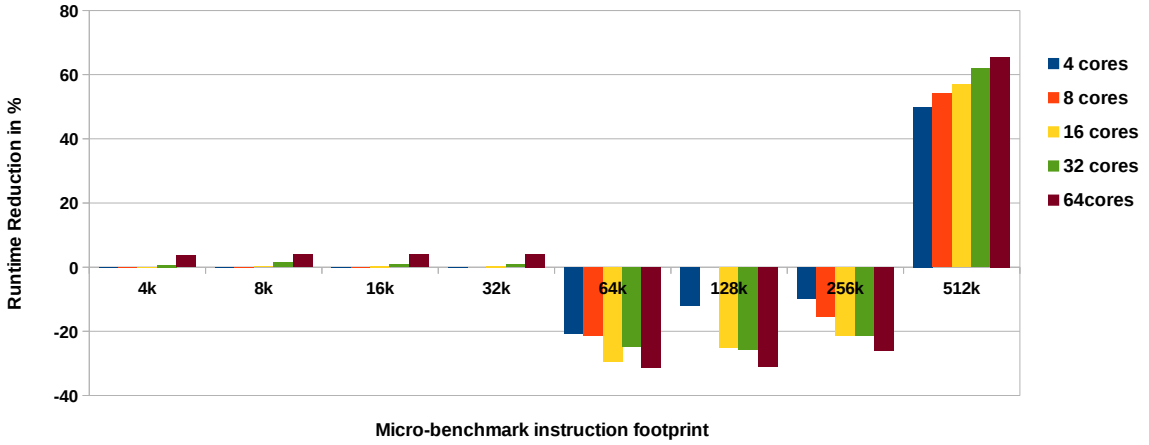
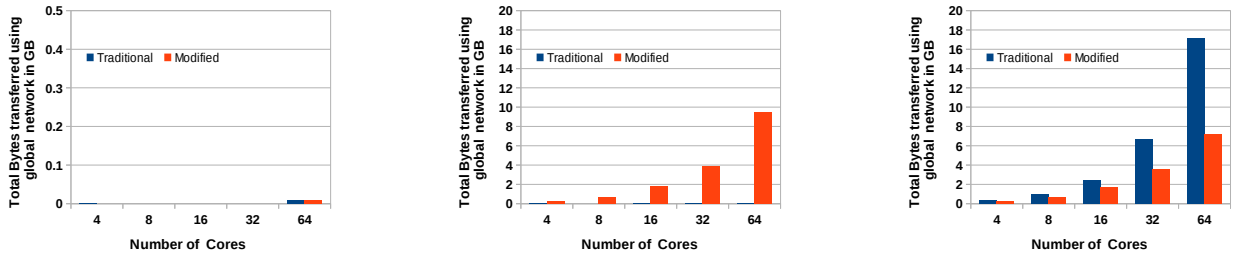


Figure 1: Runtime reduction in our proposed cache protocol compared to the traditional MOESI directory-based baseline protocol using the global on-chip interconnect to carry the proximity messages.



(a) 4 KB (b) 64 KB (c) 512 KB

Figure 2: Total number of bytes (in Gigabyte) transferred by on-chip global network in our proposed cache protocol compared to a system using the traditional MOESI directory-based baseline protocol.

Figure 2 shows the aggregate number of bytes transferred by on-chip network. Figure 2a shows that the on-chip network traffic is not affected for the 4KB instruction footprint micro-benchmark, because most of the L1-I requests are present in the private L1-I. However, as shown in Figure 2b that our extension generates more traffic than the baseline system for the benchmarks that fit in the L2 cache. In the baseline and the proposed approach for this program size no L2 misses occur. Our approach, however, also requests the neighboring caches. Figure 2c shows that our proposed cache protocol succeeds in reducing the bytes

transferred on the global on-chip interconnect for the 512kB instruction footprint micro-benchmark by up to 58%, due to a large percentage of the generated L1-I misses are serviced from neighboring caches, instead of the contacting the directory to obtain the required codes blocks.

5 CONCLUSIONS

Previous research which studied the indirection problem of the CMP directory-based coherence protocols, focused on improving the performance of the data caches. In this work, we propose a proximity coherence for the instruction caches in the tiled CMP architectures. Our results reveal a significant reduction in the overall execution time and global network traffic of up to 65.5% and 58%, respectively, for the micro-benchmarks whose instruction footprint exceeding the private L2 cache size. The degradation on the performance experienced by some micro-benchmarks result from the additional latency of the global on-chip interconnect to carry the proximity messages. In future work we will evaluate dedicated low latency links to improve the performance.

6 ACKNOWLEDGMENT

This work is funded by the Erasmus Mundus program of the European Union, Erasmus Mundus EPIC project.

References

[BBB+] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood. The gem5 simulator. *SIGARCH Comput. Archit. News*.

[CWS06] K. Chakraborty, P. M. Wells, and G. S. Sohi. Computation spreading: Employing hardware migration to specialize cmp cores on-the-fly. In *Proceedings of the 12th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS XII, 2006.

[HDH08] H. Hossain, S. Dwarkadas, and M. C. Huang. Improving support for locality and fine-grain sharing in chip multiprocessors. In *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques*, PACT '08, 2008.

[LBE+] J. L. Lo, L. A. Barroso, S. J. Eggers, K. Gharachorloo, H. M. Levy, and S. S. Parekh. An analysis of database workload performance on simultaneous multithreaded processors. In *Proceedings of the 25th Annual International Symposium on Computer Architecture*, ISCA '98.

[WFM10] N. B. Williams, C. Fensch, and S. Moore. Proximity coherence for chip multiprocessors. In *Proceedings of the 19th International Conference on Parallel Architectures and Compilation Techniques*, PACT '10, 2010.