# Proximity Scheme for Instruction Caches in Tiled CMP Architectures

Tareq Alawneh, Chi Ching Chi, Ahmed Elhossini, and Ben Juurlink

Embedded Systems Architectur (AES)
Technical University of Berlin
Einsteinufer 17
D-10587 Berlin, Germany
{tareq.alawneh, cchi, ahmed.elhossini, b.juurlink}@tu-berlin.de

**Abstract:** Recent research results show that there is a high degree of code sharing between cores in multi-core architectures. In this paper we propose a proximity scheme for the instruction caches, a scheme in which the shared code blocks among the neighbouring L2 caches in tiled multi-core architectures are exploited to reduce the average cache miss penalty and the on-chip network traffic. We evaluate the proposed proximity scheme for instruction caches using a full-system simulator running an n-core tiled CMP. The experimental results reveal a significant execution time improvement of up to 91.4% for microbenchmarks whose instruction footprint does not fit in the private L2 cache. For real applications from the PARSEC benchmarks suite, the proposed scheme results in speedups of up to 8%.

## 1 Introduction

Recently, Chip Multiprocessor (CMP) architectures have become very common. They became the focus of the leading CPU manufacturers (Intel, AMD, and IBM). The power wall and memory wall are the two main issues motivating the move from uniprocessor to CMP architectures to achieve high performance. With the continuous increase in the number of integrated processors (cores) on a single die and the size of the caches in CMP architectures, the high access latency of the large cache available on the chip is becoming critical part of future CMP architectures [ZA05]. Tiled CMP architectures are introduced as an efficient solution for the future scalable CMPs. These architectures are designed as arrays of identical tiles connected over a switched network on-chip (NoC). In tiled CMP architectures that employ the directory-based protocol to maintain cache coherence, when an instruction cache miss occurs in the Last Level Cache (LLC), a request is issued to the directory to obtain the code block, although the desired code block may be present in one of the neighbouring cores. Increasing the number of integrated cores in a single chip will increase the average number of network hops traversed to satisfy the request. As a result, the cache miss penalty and the on-chip network traffic will increase. In this paper, we present a proximity scheme for instruction caches in tiled CMP architectures. When an L1 instruction cache miss occurs, the desired code block is requested in parallel from

the neighbouring cores L2 caches as well as the private L2 cache. This makes it possible to resolve most L1 instruction misses in just few cycles by accessing the neighbouring L2 caches via dedicated paths instead of requesting the directory. This reduces the average cache miss penalty in instruction cache. Moreover, the available on-chip cache capacity is utilized effectively if the forwarded code blocks from the neighbouring cores are just copied in the local L1 instruction cache. This also reduces the on-chip network traffic by eliminating unnecessary message to the directory. The contributions of this paper can be summarized in the following points:

- Introducing a proximity technique for the instruction caches in the tiled CMP architectures, a scheme in which the desired code blocks, when a miss occurs in the L1-I cache, are serviced, in parallel, by the private L2 cache or the neighbouring caches before contacting the directory structure. Therefore, our proposed scheme reduces the cache latency, which in turn improves execution time.
- Furthermore, our proposed scheme reduces the on-chip network traffic as a result of reducing the number of required hops to reach the requested code block.
- Finally, we show that our scheme scales extremely well with the number of cores. In other words, it is particularly well-suited for large-scale CMP architectures.

## 2 Related Work

Several recent studies have proposed schemes to reduce the average cache miss penalty in the tiled CMP architectures. Brown et al. [BKT07] presented an algorithm which is proximity aware. Their proposed algorithm is based on the following observation: although the desired data is not present in the L2 cache of the home node, it might be still resided in other nodes. Therefore, the home node can issue a message to the closest sharer, requesting it to forward the desired data. This reduces the number of requests to the off-chip memory. Requests to the directory still introduce a major load on the on-chip network traffic to locate a node in proximity of the home node.

Hossain et al. [BKT08] introduced a scheme where a direct access is performed to the predicted remote L1 cache, which is likely to contain the desired data before requesting the directory. In their work, the desired data is requested from the close-by cache instead of neighbouring caches. Furthermore, the forwarded data are not usable before receiving an acknowledgement from the directory. Unlike our work, the provided code blocks from the neighbouring caches are usable immediately by the requested tile upon receiving them.

Williams et al. [WFM10] presented a scheme, in which a request is sent to the neighbouring caches before contacting the directory when a load miss occurs. Point-to-point links are used to transfer cache block between neighbouring nodes. Directory is contacted only when none of the neighbours have a copy of the requested data. This approach is introduced only for data caches.

Previous studies focused on improving the performance of the data caches. In contrast, our

work aims at improving the instruction caches with a proximity scheme. Investigating the proximity scheme allows us to exploit the read-only property of instruction cache blocks with a less hardware overhead.

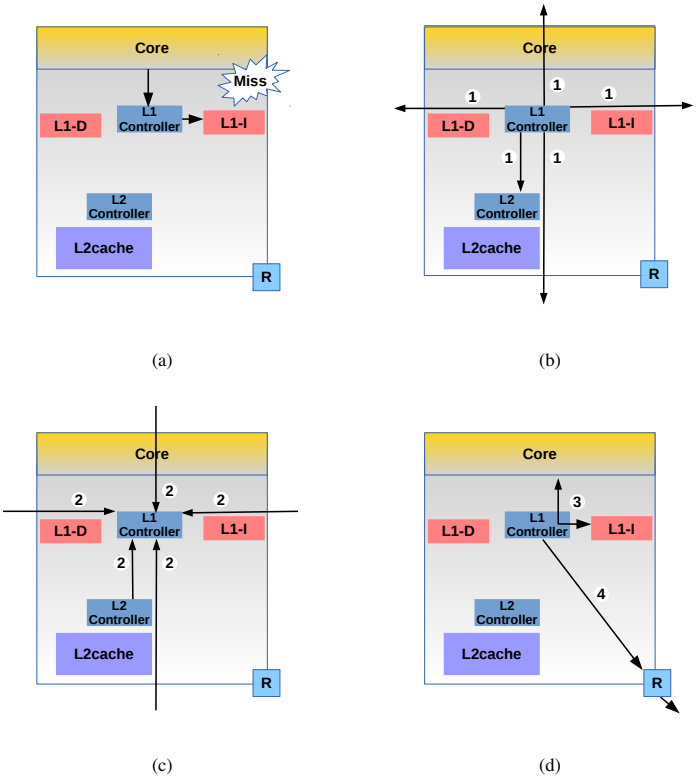



Figure 1: The detailed behaviour of proximity coherence for the instruction caches in tiled CMP architectures

## 3 The Proposed Proximity Scheme for Instruction Caches

In this paper we propose a proximity scheme for instruction cache. Dedicated links between each core and its four neighbouring cores are used to transfer the required code block from the neighbours L2 cache in the case of an instruction cache miss. The state machine of the conventional MOESI protocol is also modified to enable the use of these dedicated link. Figure 1 shows the detailed behaviour of the proximity mechanism for the instruction caches. When an instruction fetch operation results in an instruction miss (as shown in Figure 1a), the L1 cache controller sends out parallel requests to the private and neighbouring L2 caches (message 1 in Figure 1b) instead of sending a direct request to the directory.

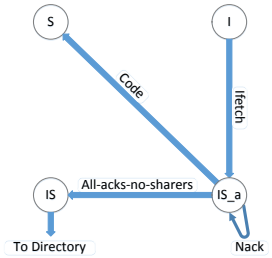



Figure 2: The additional transient states in our proximity scheme for instruction caches when an L1 instruction cache miss occurs

In the meantime, the state of the missed code block is changed to a transient state. If the private or neighbouring L2 caches have it, they reply with a hit, otherwise they return a miss (message 2 in Figure 1c). Once the L1 cache controller receives a hit from the private or the neighbouring L2 caches, the forwarded code block is copied in the private L1-I cache and forwarded to the core simultaneously (message 3 in Figure 1d) and its state is changed to the shared state. On the other hand, when the L1 controller receives all the responses and all of them replay with a miss, a request message for the directory will be issued (message 4 in Figure 1d).

Figure 2 shows the additional transient states that are added to the state machine of the conventional MOESI protocol. When an L1 instruction cache miss occurs, messages are issued to the private L2 and neighbouring caches at the same time. The code block state is moved to the *IS_a* state, which indicates that a request message is issued, waiting the responses from the private L2 cache and all neighbouring cores. If all of the them return a NACK message, which means that none of them contains a valid copy of the required code block, the code block state is moved to *IS* which indicates that a request message is issued to the directory. On the other hand, if one of the requested caches responds by sending a valid copy of the required code block, after forwarding the requested code block to the core, the code block is directly copied in the L1-I cache and its state is changed to the shared state (*S*).

## 4 Experimental Setup

By using full-system simulations based on the GEM5 [BBB+11], we evaluate the performance of our scheme against the baseline system which employs the MOESI directory-based coherence protocol. To implement a detailed simulation model for the memory subsystem, the Ruby memory model in GEM5 is used. Using a higher-level language in the GEM5 (a.k.a. SLICC), we specify our extension by modifying the state machine of the conventional MOESI protocol with all new transient states. The existing Gem5 network model is augmented with dedicated links between the neighbouring cores. Table 1 describes the values of the main parameters of the evaluated baseline system in this study. We study the tiled CMP architecture which consists of n replicated tiles interconnected

Table 1: System parameters for full-system simulation

| | |
|---|---|
| Tiled CMP size | 4, 8, 16, 32, 64 cores |
| L1-I and L1-D Cache Size | 32 KB per core |
| L1-I Cache Hit Latency | 3 cycles |
| L1-D Cache Hit Latency | 3 cycles |
| Private L2 Cache Size | 256 KB per core |
| L2 Cache Hit Latency | 15 cycles |
| L1 and L2 Block Size | 64 B |
| Network Configuration | 2D Mesh Topology |
| Memory Size | 8 GB |
| Memory Latency | 250 cycles |
| Dedicated Links Latency | 1 cycle |

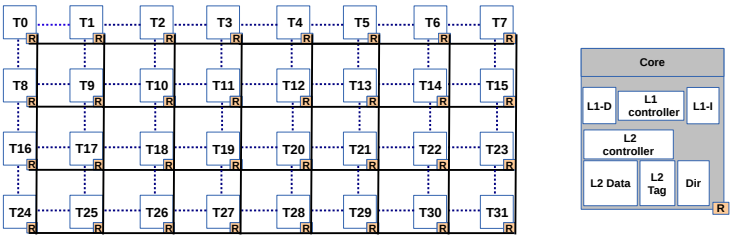
with a 2D mesh switched network as shown in Figure 3. Each tile consists of a processor core, a private split first-level instruction and data cache (L1-I/L1-D), a private second-level cache (L2), a network interface or router for on-chip data transfers and a directory to keep track of cores with copies for cached blocks.

For our evaluation, we first used microbenchmarks with the purpose of generating a high L1-I miss rate. These microbenchmarks are composed of 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 JMP instruction in every code block jumps to the JMP instruction in the next code block. In these microbenchmarks, the previous instruction pattern is replicated to create the various program sizes (i.e. its instruction footprint) from 4kB to 2MB. The entire program code is looped over by a specific count in order to have the same number of instruction cache block requests for all micro benchmarks (e.g., 8192 loops for 4 kB and 16 loops for 2MB). Each core executes the same microbenchmark. We employ this microbenchmark because it allows to precisely identify when the proposed technique is effective and when not.

Another set of experiments was performed to test the system with real applications. We evaluate some applications from PARSEC [BKSL08] parallel benchmark suite. Each run consists of n-threads of the application running on the n-core tiled CMP. Table 2 lists the applications which are simulated in this study. Due to the large number of benchmarks and the relatively long simulation time, we selected seven workloads from the PARSEC benchmark suite to cover the various application domains as shown in Table 2.

## 5 Evaluation Results and Analysis

In this section, we present the results and the analysis of the simulation results that have been obtained when running the microbenchmarks and the PARSEC suite benchmarks using our proposed proximity for the instruction caches compared to the baseline system which employs the traditional MOESI directory-based protocol.




(a) (b)

Figure 3: An example of the adopted tiled many-core processor. (a) A 4 X 8 tiled CMP (b) The architecture of a single tile. The continuous black lines show the global on-chip interconnect. The dashed black lines show the proximity links that connect L2 caches

Table 2: Simulated workloads

| Benchmark Name | Application Domain |
|---|---|
| Blackscholes | Financial Analysis |
| Swaptions | Financial Analysis |
| X264 | Media Processing |
| Dedup | Enterprise Storage |
| Canneal | Engineering |
| Fluidanimate | Animation |
| Freqmine | Data Mining |

### 5.1 Evaluation using Micro-benchmark

Figure 4 depicts the actual Average Memory Access Time (AMAT) achieved by our proximity scheme for different instruction footprints of the micro-benchmarks and for different numbers of cores. It can be seen that when the instruction footprint is smaller than or equal to 256KB, the improvements are small and in some cases even negative. The reason for this is that these instruction footprints fit in the L2 cache (Table 1). Some microbenchmarks do not provide any improvement in the AMAT as might be expected (some of them achieve a slight reduction in the AMAT due to the reduction in the cold misses). However, other microbenchmarks show an insignificant increase in the AMAT, less than 2% compared to the baseline system. The latency to access the neighbour cache is few cycles longer than that of the private L2, and combined with the fact that no local L2 copy is created in case of a hit in a neighbouring cache, the average proximity hit latency is higher than the average private L2 hit latency in the baseline approach. When the code size is 512 KB, the proposed scheme provides huge benefits, ranging from 90.3% for 4 cores to 92.1% for 64 cores. In this case the instruction footprint does not fit in the private L2 cache of a single core anymore, but a core and its neighbors together provide sufficient cache capacity to hold the entire instruction footprint. When the code size is again doubled to 1MB, interesting behavior can once more be observed. In that case significant benefits are obtained for 16, 32, and 64 cores, but for 4 and 8 cores the improvements are rather small. The reason is that when the number of cores increases, so does the average number

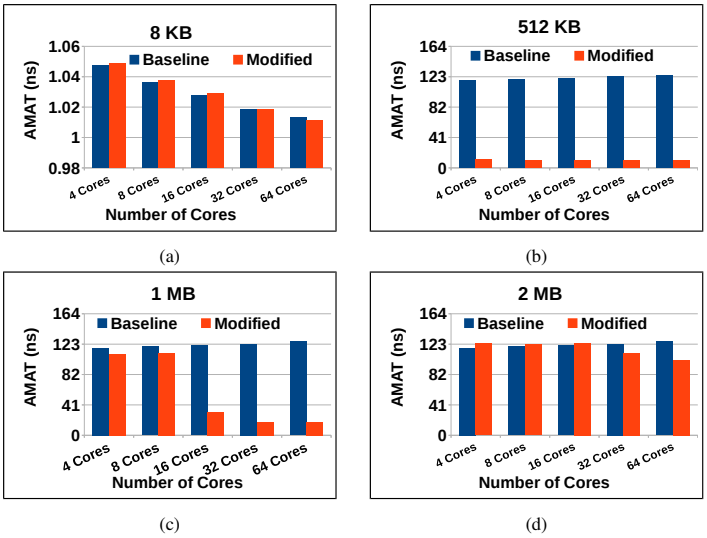



Figure 4: The actual Average Memory Access Time (AMAT) in the baseline and our proposed scheme

of neighbors. For example, in a 4-core ($2 \times 2$) CMP, each core has exactly 2 neighbors, and so the aggregate L2 cache size of a core and its neighbors is $3 \times 256 = 768$KB, which is smaller than the instruction footprint. Similarly, in a 8-core ($4 \times 2$) CMP, each core has 2 or 3 neighbors, and so the aggregate L2 cache size of a core and its neighbors is $896$KB on average, which is still smaller than the instruction footprint. On the other hand, when the number of cores is 64 ($8 \times 8$), the average number of neighbors is almost 4 (3.5), meaning that the instruction footprint fits in the aggregate L2 of a core and its immediate neighborhood. When the code size of the micro-benchmarks is again doubled, however, the instruction footprints no longer fits in the L2 caches of a core and its neighbors, which is why the improvements decrease substantially. For 32 and 64 cores, the proximity scheme still provides improvements, but for 16 and fewer cores, there is a small slowdown. In the 16 and fewer cores, the aggregate L2 cache size of a core and its neighbors is too small to hold the microbenchmark's footprint. Therefore, the proximity hit rate is low, which in turn increases the AMAT. The latency to access the neighbouring caches is few cycles higher than the access to the private L2 cache. Although, the aggregate L2 cache size of a core and its neighbors in the 32 and 64 cores is still small to hold microbenchmark's full footprint. The reduction in the L2 cache miss penalty is higher than the proximity overhead, which is why the proposed scheme still achieves improvements in the 32 and 64 cores.

Figure 5 shows the overall reduction in execution time presented by the proposed approach compared to the baseline for the same set of microbenchmarks. Microbenchmarks with small instruction footprints, as expected, do not provide any significant improvement in the execution time. On the other hand, the aforementioned improvements in the AMAT translated into reduction on the overall execution time in microbenchmarks with a footprint that does not fit in the private L2 cache (512KB and 1MB). Our proposal achieves execution

time reduction of up to 91.4% compared to the the baseline. Similar behaviour can be observed for the 2MB benchmark. The achieved reduction in execution time corresponds to the reduction in the AMAT for the same benchmark.

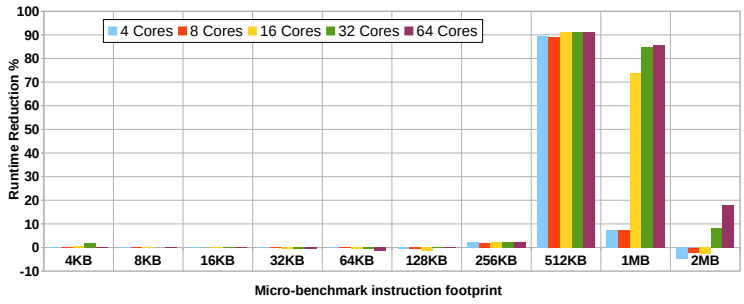



Figure 5: Runtime reduction in our proposed approach compared to the baseline system

Figure 6 shows the aggregate number of bytes transferred by the on-chip network. The proposed approach introduces a significant reduction in the on-chip network traffic for all microbenchmarks compared to the baseline system which employs the traditional MOESI directory-based protocol. Contacting the neighbouring cores via the dedicated links to obtain the required cache blocks significantly reduced the on-chip network traffic. The microbenchmarks with instruction footprints smaller than 32KB, provide on average 38.7% reduction in the on-chip network traffic. For these benchmarks most of the traffic is due to the cold misses and they are served mainly by the dedicated links from neighbouring caches. As a result, the total on-chip network traffic is reduced. In all CMP configurations based on Table 1, we can observe that the proposed scheme introduces more reduction in the on-chip network traffic compared to the baseline system as the number of core count grows. When the data and control messages travel using the on-chip network, they may take several hops to reach the destination. The average number of hops increases as the number of cores increases, which is significant for the baseline system. Larger microbenchmarks that do not fit in the L1 cache but still fit in the L2 cache present similar reduction in the on-chip network traffic for the same aforementioned reasons. However, microbenchmarks with foot-prints 512KB and 1MB benefit more because their foot-print does not fit in the L2 cache. In the baseline system, misses to the L2 cache are serviced by contacting the directory which increases the network traffic. In our approach these requests are serviced by the neighbouring cores eliminating the network traffic. On average, our proposed approach achieves a reduction in the network traffic by 45%.

### 5.2 Evaluation using Real Benchmarks

Evaluating the proposed scheme using the microbenchmarks shows that the proposed approach can be used to reduce the execution time of application by reducing the average memory access time. This approach can be used to reduce the on-chip network traffic as

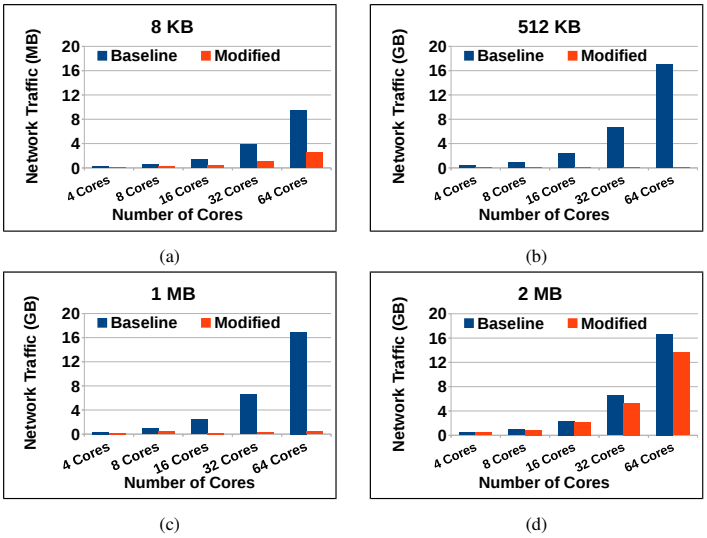
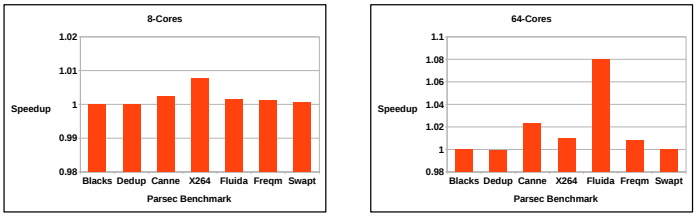



Figure 6: Total number of bytes (in GB or MB) transferred by on-chip global network in our proposed approach compared to the baseline system

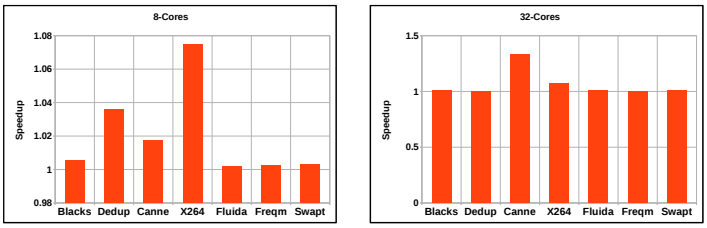
well. In this section, we present the evaluation results using real applications from PARSEC benchmark suite. All simulations are performed using GEM5 as explained in Section 4. The benefit of our proposed proximity scheme for the instruction caches is limited by the high L1-I cache hit rates which are observed in the simulated PARSEC workloads. Therefore, we reduce the L1-I cache size to 4KB to show this benefit. Figure 7 shows the execution time speedups that have been obtained for PARSEC benchmarks when shrinking the L1-I cache to 4KB for 8 and 64 cores. The proposed approach achieved speedups for all benchmarks up to 8% compared to the baseline system. The blackscholes has a small workload that fits in the L1-I cache. Therefore, it achieves a slight speedup due to the cold misses which are serviced by the neighbouring caches. On the contrary, the speedup in the fluidanimate workload increases as the number of cores grows. This comes from the increase in the average aggregated cache capacity of the neighbouring cores. The improvements in the other benchmarks are small and some cases even negative. The reason for this is that their instruction footprints fit in the private L2 cache. The slight speedups, which achieved in some benchmarks, result from the reduction in the cold misses. On the other hand, the small slow down in other benchmarks comes from the low proximity hit rate. Therefore, the AMAT increases due to the proximity overhead.

Figure 8 shows speedup for reduced L1-I cache size of 4KB and the L2 cache size of 64KB for 8 and 32 cores. In the case of the canneal and x264 benchmarks, the proposed scheme provides speedup up to 33% and 7.5% respectively. The instruction footprints of these two benchmarks do not fit in the private L2 cache of a single core anymore, but a core and its neighbors together provide sufficient cache capacity to hold the entire instruction footprint. On the contrary, other benchmarks still fit in the L2 cache and provide slight speedups due to the reduction in the cold misses.



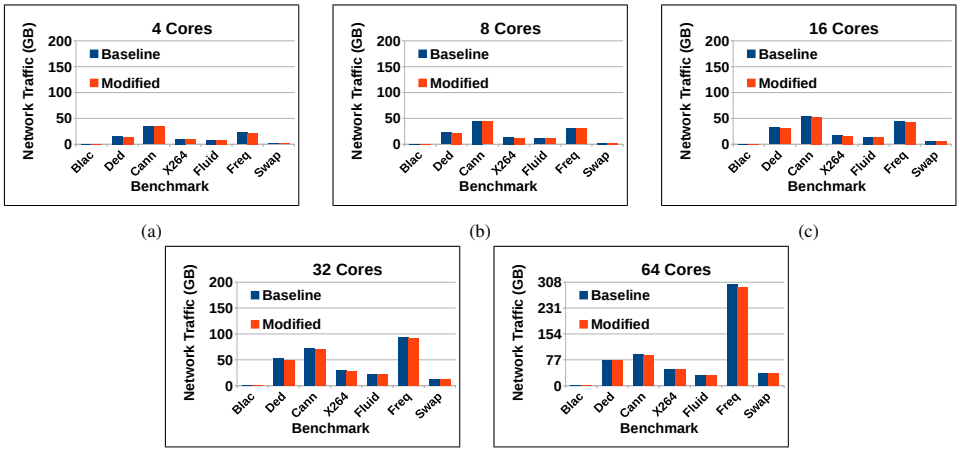

(a) (b)

Figure 7: The achieved runtime speedup in our proposed approach compared to the baseline system. When L1-I cache size is 4KB, L1-D cache size is 32KB, and L2 cache size is 256KB




(a) (b)

Figure 8: The achieved runtime speedup in our proposed approach compared to the baseline system in small and large CMP configurations. When L1-I cache size is 4KB, L1-D cache size is 32KB, and L2 cache size is 64KB




(a) (b) (c)

(d) (e)

Figure 9: Total number of bytes (in GB) transferred by on-chip global network in our proposed approach compared to the baseline system. When L1-I cache size is 4KB, L1-D cache size is 32KB, and L2 cache size is 256KB

Figure 9 shows the aggregate number of bytes transferred by the global on-chip network when the L1-I cache size is 4KB. As shown in the Figures from 9a to 9e, the total bytes transferred by the on-chip network in the blackscholes is negligible for both the traditional and proposed systems, because most of the L1-I requests are serviced by the L1-I cache. We can also observe that all the simulated PARSEC benchmarks achieve a reduction in the on-chip network traffic (9.4% on average) when our proposed mechanism is employed. This is due to requests which are serviced via the dedicated links, which in turn reduce the number of the messages transferred by the on-chip network.

## 6 Conclusions and Future work

Tiled CMP architectures are introduced as the best choice for the future scalable CMPs. As the number of the cores grows, the average cache miss penalty will have a significant impact on the overall performance. Several approaches have been proposed to reduce the average cache miss penalties in the tiled CMP architectures which employ a directory-based coherence protocol. These approaches focused on improving the performance of data caches. In this work, we propose a proximity scheme for the instruction caches to provide execution time improvements as well as reduction in the on-chip network traffic. Our results reveal a significant reduction in the overall execution time and global network traffic of up to 91.4% and 99%, respectively, for the microbenchmarks whose instruction footprint exceeding the private L2 cache size. Moreover, the proposed policy led to improvement in the PARSEC workloads execution time of up to 8% compared to the baseline system.

Applications with large footprints that do not fit in the modern L1-I cache, such as Online Transaction Processing (OLTP) workloads [LBE+98, ATAM12, HA04, KADS03, IATAM13] are most likely to benefit from the proposed approach. They exhibit a high degree of instruction reuse over multiple cores. Several prior studies observed this extensive sharing of instruction blocks among the multiple processors in different workloads [ATAM12, KADS03, CWS06, HA06]. While we do not simulate these workloads in this study, they will be considered in future work. Moreover, a detailed analysis of the power consumption for our proposed scheme is left to future work.

## Acknowledgment

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

## References


[ATAM12] I. Atta, P. Tozun, A. Ailamaki, and A. Moshovos. SLICC: Self-Assembly of Instruction Cache Collectives for OLTP Workloads. In *Proceedings of the 45th Annual IEEE/ACM International Symposium on Microarchitecture, MICRO*, pages 188–198, 2012.

[BBB+11] 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. *ACM SIGARCH Computer Architecture News*, 39:1–7, May 2011.

[BKSL08] Christian Bienia, Sanjeev Kumar, J. P. Singh, and K. Li. The PARSEC Benchmark Suite: Characterization and Architectural Implications. In *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques, PACT*, pages 72–81, 2008.

[BKT07] J. A. Brown, R. Kumar, and D. Tullsen. Proximity-Aware Directory-based Coherence for Multi-core Processor Architectures. In *Proceedings of the 19th Annual ACM Symposium on Parallel Algorithms and Architectures, SPAA*, pages 126–134, 2007.

[BKT08] J. A. Brown, R. Kumar, and D. Tullsen. 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*, pages 155–165, 2008.

[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*, pages 283–292, 2006.

[HA04] S. Harizopoulos and A. Ailamaki. Steps Towards Cache-Resident Transaction Processing. In *Proceedings of of the 30th International Conference on Very Large Data Bases, VLDB*, pages 660–671, 2004.

[HA06] S. Harizopoulos and A. Ailamaki. Improving Instruction Cache Performance in OLTP. *ACM Transactions on Database Systems*, 31:887–920, September 2006.

[IATAM13] P. Tözün I. Atta, X. Tong, A. Ailamaki, and A. Moshovos. STREX: Boosting Instruction Cache Reuse in OLTP Workloads Through Stratified Transaction Execution. In *Proceedings of the 40th Annual International Symposium on Computer Architecture, ISCA*, pages 273–284, 2013.

[KADS03] P. Kundu, M. Annavaram, T. Diep, and J. Shen. A Case for Shared Instruction Cache on Chip Multiprocessors Running OLTP. In *Proceedings of the 2003 Workshop on MEmory Performance: DEaling with Applications, Systems and Architecture, MEDEA*, pages 11–18, 2003.

[LBE+98] J. L. Lo, L. A. Barroso, S. J. Eggers, K. Gharachorloo, H. M. Levy, and S. S. Parekh. An Analysis of Database Workloads Performance on Simultaneous Multithreaded Processors. In *Proceedings of the 25th Annual International Symposium on Computer Architecture, ISCA*, pages 39–50, 1998.

[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*, pages 123–134, 2010.

[ZA05] M. Zhang and K. Asanovic. Victim replication: Maximizing capacity while hiding wire delay in tiled chip multiprocessors. In *Proceedings of the 32nd Annual International Symposium on Computer Architecture, ISCA*, pages 336–345, 2005.