ELSEVIER

Microprocessors and Microsystems 24 (2000) 429–436

MICROPROCESSORS AND MICROSYSTEMS

www.elsevier.nl/locate/micpro

Utilization of cache area in on-chip multiprocessor

Hitoshi Oi[a,*], N. Ranganathan[b]

[a]HAL Computer Systems, Inc., Campbell, CA 95008, USA
[b]Department of Computer Science and Engineering, University of South Florida, Tampa, FL 33620, USA

Received 18 January 2000; revised 14 August 2000; accepted 5 September 2000

Abstract

On-chip multiprocessor can be an alternative to the wide-issue superscalar processor approach which is currently the mainstream to exploit the increasing number of transistors on a silicon chip. Utilization of the cache, especially for the remote data is important in the system using such on-chip multiprocessors since the ratio of the off-chip and the on-chip memory access latencies is higher than traditional board-level implementation of the cache coherent non-uniform memory access (CC-NUMA) multiprocessors. We examine two options to utilize the cache resource of the on-chip multiprocessors whose size is restrained by the die area: (1) the instruction and/or private data are only cached at the L1 cache to leave more space on the L2 cache for the shared data; (2) divide cache area into the L2 and the remote victim caches or use all the area for the L2 cache. Results of execution-driven simulations show that the first option improved the performance up to 15%. For the second option, a remote victim cache with 1/8 of the L2 cache size improved three out of four benchmark programs by 4–8%. However, the combination of L2 and victim caches that divide the cache area into two halves of the same size was outperformed by the L2 cache occupying the entire cache area in three out of four benchmark programs.

Keywords: Cache area; On-chip multiprocessor; Memory latency; Performance evaluation

1. Introduction

The progress of the semiconductor device technologies has increased the number of transistors on a silicon chip. Until now, this increasing number of transistors on a chip has been used to implement superscalar processors which can issue multiple instructions simultaneously [1]. However, this approach is limited by the following. The circuit that implements a higher degree of instruction level parallelism (ILP) than the current level, (two to three instructions can be issued simultaneously) occupies much more die area while most of the applications cannot exploit such a high degree of ILP [2]. Also, the circuit complexity of the processor with a higher degree of ILP lowers the clock speed. It is reported that the performance of an eight-issue processor is only 20% better than that of a four-issue processor when the clock speed is taken into consideration [3].

An alternative approach is to implement a multiprocessor on a single chip [2,4–7]. In this approach, each processor is moderately superscalar (for example, there were four two-issue processors in the estimation of [4]) and operates at a very high clock rate. In a multiprogramming environment, each processor executes a process from a different application, and benefits from the moderate superscalar architecture and the fast clock speed. When an application has a coarse grain parallelism, it is divided into multiple threads either by the programmer or by the compiler. Each processor executes a different thread from the application, and threads benefit from the fast on-chip communication as well as from the fast clock speed.

Yet another approach to exploit a large number transistors is to integrate the main memory and the processor on the same chip [8]. With this approach, the memory latency is significantly reduced due to the wider on-chip datapath and the lower signal propagation delay. In this paper, the combination of the second and the third approach is considered: implementing a small number of processors and the main memory on a single chip. When the processors are running in a multiprogramming environment, they benefit from the fast access of the on-chip memory modules. When processors are executing a parallel program, they benefit from the fast interprocess communication.

When the number of processors on a single chip is not sufficient to provide the computational power required by the application, it is possible to build a larger system by connecting several such on-chip multiprocessors by an interconnection network. The resulting system is a CC-NUMA with each on-chip multiprocessor being a cluster

* Corresponding author.
E-mail address: oi@hal.com (H. Oi).

0141-9331/00/$ - see front matter
PII: S0141-9331(00)00094-6

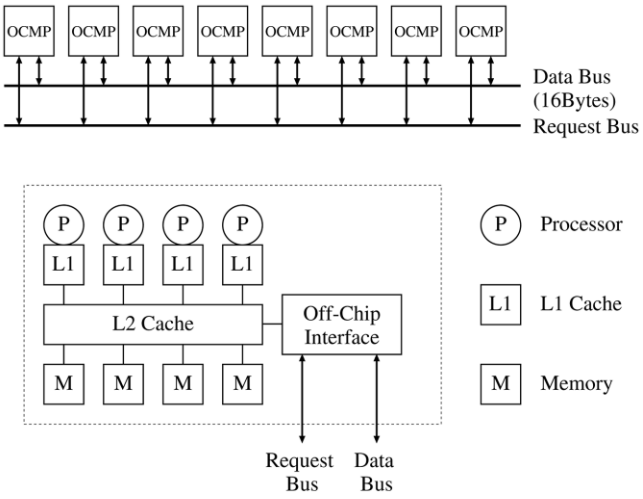

Fig. 1. On-chip multiprocessor architecture and a bus-based 32 processor system.

[9]. However, the access time ratio between the local and the remote memory is expected to be higher than that of current CC-NUMA systems that are built with traditional technologies, such as printed circuit board (PCB). To reduce the higher latency of the remote access, the utilization of a cache would become important.

In this paper, two design options are examined to utilize the cache resource of the on-chip multiprocessor whose size is restrained by the die area. Due to the on-chip integration, the access speed of the local memory becomes close to that of the L2 cache. The first option is to give more room on the L2 cache for the shared data by single level caching (i.e. using only L1 cache) for the private data and/or instructions. Higher locality of the private data and the instruction access than the shared data and the faster on-chip memory make this option possible. Another option is to incorporate a remote victim cache, a cache that only stores remote blocks replaced from the L2 cache, when the total cache area is limited by the die size.

This paper is organized as follows. In the next section, the architecture of the on-chip multiprocessor is described. In Section 3, the methodology for the performance evaluation including the simulation model and benchmark programs are presented. The performance evaluation of the on-chip multiprocessor and the effectiveness of the design options are given in Section 4. A summary of this paper is provided in Section 6.

2. On-chip multiprocessor architecture

Olukotun et al. investigated a floor-plan of their on-chip multiprocessor architecture and estimated the area consumption in detail [4]. Their architecture is used as a starting point. The semiconductor technology they assumed was that of 1997. When manufactured in the near future, it is expected that more transistors will be available on a silicon chip than Olukotun assumed. It is assumed that this extra die area is to be used for accommodating the main memory on the same chip. This on-chip main memory reduces the memory access time significantly. Off-chip memory accesses involve I/O circuits and connecting wires that have much higher inductive and capacitive load. The on-chip memory can avoid these delays. Secondly, the width of the on-chip memory datapath can be wider than that of the off-chip memory. This is because the datapath width of the on-chip memory is not restricted by the number of pins of the processor chip.

This on-chip multiprocessor can be used as a building block of a larger scale multiprocessor by connecting several chips with some interconnection network. The resulting system is a CC-NUMA with each chip being a cluster [9]. However, the ratio of access speed between the local and the remote memory is significantly higher in this architecture due to the faster on-chip memory access. In other words, the remote miss rate has a greater impact on the performance of the on-chip multiprocessor than on CC-NUMA systems built with traditional technologies, such as PCB.

Fig. 1 shows the architecture of the on-chip multiprocessor. There are a small number of processors that are moderately superscalar. Each processor has its own instruction and data L1 caches. The size of the L1 caches is small and it has a low degree of set-associativity (probably direct-mapped or two-way as seen in current processors) so that its access speed can catch up the fast processor clock cycle. Processors on the same chip share an L2 cache that has a larger size, a higher degree of set-associativity and hence a slower access speed. The on-chip main memory is divided into several modules to avoid access contention. Each memory module is associated with directory entries representing copies of memory blocks. The off-chip communication interface is to connect several on-chip multiprocessor chips via an interconnection network and/or to external memory modules.

Table 1
System parameters. Timing parameters are represented by processor clock cycles

L1 cache	Direct mapped; Size: 8 KB/processor	Latency: 1 Block size: 32 bytes
L2 cache	Four-way set-associativity; Size: 64 KB/chip	Latency/cycle time: 8/4 Block size: 64 bytes
Memory	Four-way interleaved; Latency/cycle time:	 25/30
Off-chip I/F	Latency/cycle time:	10/5
Bus	Cycle: 5	Data bus width: 16 bytes
Configuration	4 Processors per chip	8 chips (32 processors)

3. Simulation environment

In this section, the simulation environment to evaluate the performance of the proposed architecture in Section 4 is described.

3.1. Architecture model

For the performance evaluation, an execution-driven simulator was developed using the ABSS multiprocessor toolkit [10]. ABSS takes a parallel application written with p4-macro [11] and augments it with instrumentation codes.

Each chip has four processors as in [2]. A configuration of a bus-connected 32 processors (8 chips) system is considered as a case study in this paper. Each on-chip multiprocessor has the configuration in Fig. 1 with an assumption that the size of the on-chip memory is large enough to store all the instructions and the data.

The L1 cache is direct-mapped and separated for the instruction and the data, while the L2 cache is four-way set-associative and unified. A split transaction bus is used to connect eight chips that are separated for the request (read request, invalidate request and acknowledgment) and the data. An invalidation request is broadcast to all the chips via the request bus, and each chip which has a copy of the block will send an acknowledgment when it has finished the invalidation of the corresponding cache block in its L2 cache. The on-chip memory module is four-way interleaved. Contention at the shared resources (L2 cache, memory module, off-chip interface, request/data buses) are modeled.

The system parameters that will be used in the performance evaluation in Section 4 are shown in Table 1. A relatively small L2 cache (64 KB per chip) is chosen, since the default problem sizes of SPLASH2 suites are used which were determined to run simulations in reasonable time [12]. However, the data sizes of practical applications are so large that they are likely to split out from the cache [16].

The clock speed of the bus is five times slower than that of the processor. Since the width of the data bus is 16 bytes, it takes four bus cycles (equivalent to 20 processor clock cycles) to transfer a L2 cache block (64 bytes) between chips via the data bus.

Table 2
Benchmark programs and their problem sizes

Application	Description	Problem size
FFT	1-D FFT	64k complex points
Ocean	Multigrid solver of ocean movement	258×258 grid
Radix	Radix sort	256k keys, radix = 1024
Volrend	Rendering of 3-D volume	Head

3.2. Benchmark programs

A set of parallel applications in Table 2, that are from Stanford SPLASH2 suites [12], are used as benchmark programs for the performance evaluation. FFT is a one-dimensional Fast Fourier Transform of n complex points. Ocean simulates large-scale ocean movements by multigrid equation solver (contiguous allocation version). Radix performs a radix sort of integers. Volrend renders a three-dimensional volume using a ray casting technique. In FFT and Volrend, the initialization was excluded from the instrumentation. The default problem size of each benchmark program in [12] is used.

The allocation of the shared data to memory modules is as follows. When a memory location (in the unit of the L2 block size) is accessed by a processor for the first time, the memory location is assumed to be allocated in one of the memory modules in the same chip as the processor. Within a chip, one of memory modules is chosen in a round-robin manner.

4. Performance evaluation

In this section, the performance of the on-chip multiprocessor is evaluated with the proposed design options for performance improvement.

4.1. Single-level caching of private data and/or instruction access

For each benchmark program, code and data sizes, number of accesses, and L1 and L2 cache hit ratios are summarized in Tables 3–6.

In general, the L1 hit ratios of the instruction and the private data access are very high ($\geq$96%) and are much higher than that of the shared data access. This fact leads to the idea of leaving more room on the L2 cache for the shared data by using only the L1 cache for the instruction and/or private data. It is likely that the instructions and the private data are allocated on the memory modules which are on the same chip as the processor accessing them. Thus, even if the accesses to the instruction/private data are missed at the L1 cache, they can be fetched from the on-chip

Table 3
FFT: hit ratio and data/instruction sizes

Category	# Access (M)	Size	L1 hit ratio (%)	L2 hit ratio (%)
Overall	67	3.5 MB	95.70	71.15
Instruction	54	8 KB	96.51	99.82
Private	6.4	160 KB	97.35	71.15
Shared	6.5	3.1 MB	87.32	66.66

memory quickly. On the other hand, in the case of the shared data access, its miss penalty will be expensive due to the off-chip communication that is significantly slower than the on-chip communication. Also, the hit ratio of the shared data is lower than that of the instruction/private data due to invalidations.

There are several characteristics that an application and/or a system configuration should have to benefit from the option of leaving more room on the L2 cache for the shared data. The L1 hit ratio of the instruction/private data should be high, while their L2 hit ratio should be low. The access latencies of the L2 cache and the on-chip memory should be close (which is likely for the on-chip memory). These characteristics suppress the increase of the L1 miss penalty when the L2 is not used for the instruction/private data access. Capacity and conflict misses should be dominant within all the L2 misses of the shared access so that not using the L2 for the instruction/private data can improve the L2 hit ratio of the shared access. The cost of the remote access should be expensive so that the improvement of the L2 hit ratio for the shared access will be more effective. This can be further divided into a slower interconnection network, higher contention at interconnection network, etc.

Four different cases are examined: In *base* case, all the instructions, private data, and shared data occupy the L2 cache. With *priv* option, private data does not occupy the L2 cache (it is only stored at the L1 cache). Similarly, with *inst* option, the instruction is only cached at the L1 cache. $P + I$ is a combination of the *priv* and the *inst* options. The execution speed (inverse of the execution time) normalized to that of the *base* case of each benchmark program is presented in Fig. 2. The number of replacements at the L2 normalized to the *base* case is shown in Fig. 3.

The L1 hit ratio of the instruction fetch for FFT is lower than those of other benchmark programs while the L2 hit ratio is quite high (99.82%). This means that the increase of the L1 miss penalty for the instruction fetch is higher than other benchmark programs when the L2 cache is not used for the instruction access. The size of the FFT instruction code is small (8 KB) since it is a kernel program [12]. Thus, not using the L2 cache for the instruction did not improve the L2 hit ratio of the shared access significantly (from 66.66 to 66.70%). Consequently, *inst* and $P + I$ degraded FFT's performance to 67 and 62% of the *base* case (not appeared in Fig. 2 since they are out of range), respectively. Therefore, this option should not be used for FFT. On the other hand, the L1 hit ratio of the private access is higher than that of the instruction access (97.35%) while the L2 hit ratio is lower than the instruction access (71.15%). Also, the size of the private data (160 KB) is much larger than the instruction size (8 KB). As a result, the number of replacements at the L2 cache was reduced by 21%, the L2 hit ratio of shared data was increased from 66.66 to 68.38%, and the performance of FFT was improved by 4% with the *priv* option.

In Ocean, most of the L2 misses are accesses to the local memory blocks. It is known that Ocean has a strong nearest-neighbor communication pattern [13]. The cluster structure of the on-chip multiprocessor makes most of misses satisfied locally (within the same chip). Consequently, the fraction of the remote accesses is quite small in all the access modes. Thus, the effects of two design options examined in this section are also small. *Priv* and *Inst* options improved Ocean's performance by 3 and 1%, respectively. $P + I$ option was slightly worse than *priv* alone. Actually, the number of replacements at the L2 cache was smaller for $P + I$ than for *Priv* or *Inst* alone. However, the increased L1 miss penalty for the instruction and private data access killed the benefit of the increased L2 hit ratio for the shared data access.

Neither *priv* nor *inst* option alone improved the performance of Radix significantly. With *priv* option, the average access latency was slightly decreased, but the number of stalls due to synchronization was increased. Thus, it resulted

Table 4
Ocean: hit ratio and data/instruction sizes

Category	# Access (M)	Size	L1 hit ratio (%)	L2 hit ratio (%)
Overall	765	17.8 MB	95.80	57.83
Instruction	567	68 KB	99.55	87.01
Private	56	109 KB	96.17	61.27
Shared	142	15.5 MB	80.70	54.85

Table 5
Radix: hit ratio and data/instruction sizes

Category	# Access (M)	Size	L1 hit ratio (%)	L2 hit ratio (%)
Overall	117	2.48 MB	97.17	80.72
Instruction	90	4 KB	99.99	73.99
Private	19	168 KB	99.25	54.35
Shared	8	2.3 MB	60.98	81.92

in a slight performance degrade (less than 1%). However, when two options were used together, ($P + I$), the performance of Radix was improved by 3%.

In Volrend, the benefit of the single-level caching for the private data and/or instruction access was most effective. With the *priv* option alone, its performance was improved by 14%. In Volrend, there were frequent private data accesses. This increased the busy rate of the L2 cache which was shared by four processors on the same chip. By not using the L2 cache for the private data access, the average wait time per L2 cache access was reduced from 0.61 to 0.18 processor clock cycle. Since the L2 hit ratio of the private access was quite low (35%), the increase of the L1 miss penalty for the private access caused by the *priv* option was small. The number of replacements at the L2 cache was reduced to 14% of the *base* case by the *priv* option (Fig. 3). Although it was not as good as the *priv* option, the *inst* option reduced the number of replacements at the L2 cache by 26%, and it improved the performance of Volrend by 5%. When two options were combined ($P + I$), the performance of Volrend was improved by 15%.

4.2. Effectiveness of remote victim cache

In CC-NUMA multiprocessors, the latency of the remote data access is significantly higher than that of the local data. One way to alleviate the latency of the remote access is to add a remote cache to the memory hierarchy [16]. Moga and Dubois proposed an augmentation to the remote cache, the remote victim cache: a cache memory that only stores remote blocks replaced from the L2 cache [17]. Their study showed that relaxing the inclusion property better utilizes the remote cache.

In the past studies including the above, it was assumed that nodes and clusters of CC-NUMA multiprocessors were build with traditional technologies such as printed circuit board (PCB). Thus, the addition of the remote caches was mainly restrained by the cost. However, in the case of the on-chip implementation, the total size of the L2 cache and the remote victim cache is restricted by the die area. A question that arises in this situation is, should the available chip area be divided into L2 and remote victim caches, or just use it for a large L2 cache.

Six cache configurations are examined: the base configuration in Table 1 (an L2 cache of the size 64 KB/chip only), an L2 cache of the size 64 KB/chip plus a remote victim cache of sizes 8, 16, 32, and 64 KB/chip, and an L2 cache of 128 KB/chip only. A notation of X:Y is used, where X and Y are the sizes of the L2 and the remote victim caches, respectively. The total chip area used for the L2 and remote victim caches should be in the order of $64{:}0 < 64{:}8 < 64{:}16 < 64{:}32 < 128{:}0 < 64{:}64$. This is because 64:64 consumes more area for the cache tag and the datapath than 128:0. If 64:32 or a smaller victim cache outperforms 128:0, it can be said that the remote victim cache better utilizes the limited cache area, and vice versa. It is assumed that the remote victim cache is four-way set-associative and its access hit takes four clock cycles in addition to the L2 latency.

A remote victim cache of the smallest size (8 KB) improved the performance of FFT by 4%. In FFT, the remote victim cache was effective only in the write accesses (i.e. there were no hits for the read access at the remote victim cache). The number of hits at the remote victim cache itself was small (only 0.12% of the L2 write misses were saved by the remote victim cache). However, subsequent write accesses to the same address resulted in either L1 or L2 hits, and they lowered the average access latency. Although it was negligible, there was a negative effect of the remote victim cache. The remote victim cache prevents a modified L2 cache block from being written back to its home memory location. This reduces the number of write-backs and leads to fewer bus transactions, but it also increases the number of remote accesses. Since the remote victim cache keeps modified data in a remote location (owner node), read accesses by processors in other chips

Table 6
Volrend: hit ratio and data/instruction sizes

Category	# Access (M)	Size	L1 hit ratio (%)	L2 hit ratio (%)
Overall	1815	10.6 MB	97.21	62.60
Instruction	1326	19 KB	98.39	84.28
Private	452	122 KB	95.87	35.00
Shared	36	10.5 MB	70.69	67.43

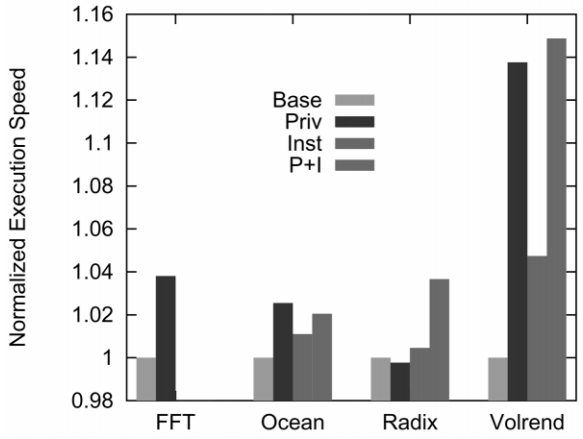

Fig. 2. Single-level caching of instruction/private data.

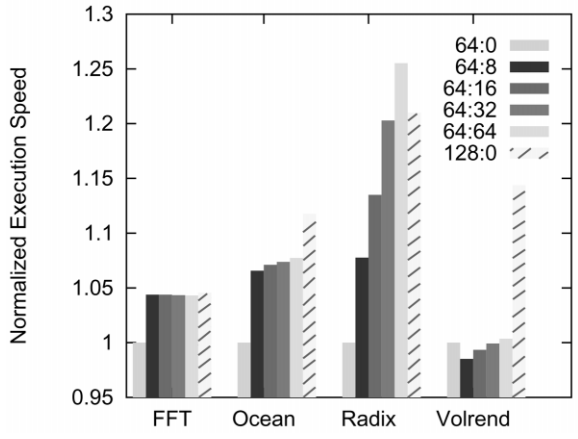

Fig. 4. Effect of L2 cache and remote victim cache sizes.

involve extra transactions between the home node and the owner node. In FFT, increasing the size of the remote victim cache beyond 8 KB did not prove effective. The performance of the configurations with a larger remote victim cache (64:8–64:64) were almost the same, despite the size of the remote victim cache being increased.

The effectiveness of the remote victim cache in Ocean is similar to that in FFT. A remote victim cache of the size 8 KB improved the performance of Ocean by 6%. However, more improvement cannot be expected when the size of the remote victim cache is increased beyond 8 KB. For example, in 64:64, the size of the remote victim cache size was eight times larger than in 64:8, but the performance of Ocean was only 1% better than 64:8. The performance improvements of FFT and Ocean showed a difference when a larger L2 cache was used: 128:0 improved Ocean's performance by 12%, which was about 4% better than 64:64. There were also some similarities and differences between FFT and Ocean. For example, the number of victim cache hits was small (0.03%), but they made subsequent accesses to the same address as either L1 or L2 hits. Also, all the remote victim cache hits were in the write mode in both benchmark programs, but in the case of Ocean,

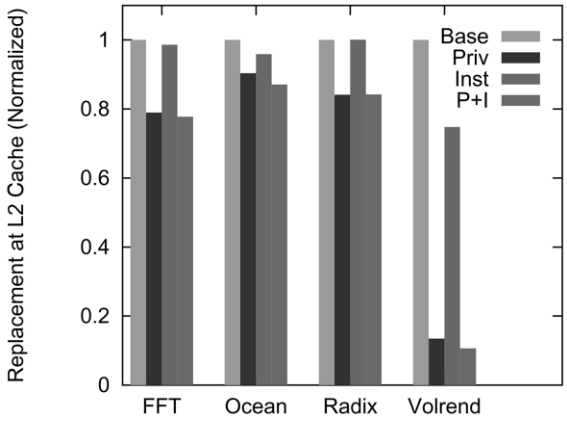

Fig. 3. Normalized number of replacement at L2 cache.

addition of the remote victim cache did not increase the number of remote read accesses.

The use of a remote victim cache was most effective in Radix. Even 64:8 improved the performance of Radix by 8%, and its effectiveness was increased up to 26% with 64:64. Again, all the remote victim cache hits were in write modes. With the 64:64 configuration, 18% of the L2 write misses resulted in remote victim cache hits. A larger L2 cache (128:0) also improved the performance of Radix by 21%, but it was slightly less than 64:64. The hit ratio of the L2 cache in 128:0 (89.7%) was better than that of 64:64 (80.5%). However, 18% of the L2 write misses resulted in remote victim cache hits, and the remote victim cache reduced the number of remote write backs of modified cache blocks (in 128:0 there were 44,802 write backs while there were 24,116 in 64:64).

A small fraction (0.02%) of the L2 write misses were saved by the remote victim cache of 8 KB size in Volrend. However, the number of the L2 cache hits was not increased. This means that a cache block that was hit at the remote victim cache was not re-used by any processor in the same chip later. In addition, the number of remote read accesses due to the victim cache was increased by 7%. Thus, the performance of 64:8 was slightly worse than that of 64:0. Actually, the benefit of the remote victim cache was negligible, even with 64:64. On the contrary, 128:0 improved the performance of Volrend by 14%. As it has been shown in the previous subsection, a large fraction of accesses in Volrend was to private data in which the remote victim cache was not effective. A larger L2 cache (128:0), however, worked for both the private and the shared data, and hence it improved the performance of Volrend significantly.

5. Related work

Olukoton et al. studied the on-chip multiprocessor as an alternative to wide-issue superscalar processors, which is currently the mainstream for using transistors on a silicon

chip [4]. They compared a six-issue superscalar processor and a four-way on-chip multiprocessor, for parallel applications and multiprogramming workload. They assumed the semiconductor technology of 1997 and estimated that four processors and L1 and L2 caches could be implemented on a single chip. It is expected that technology of near future can provide more die area than they estimated. Thus, we assumed the main memory can also be included on the same chip. Another difference was that they evaluated the performance of single chip configurations, while we studied multiple chip configurations where the inter-chip communication would have a significant effect on the performance.

We assumed that the L2 cache is shared by the processors on the same chip. The effectiveness of the shared L2 caches was studied by Nayfeh et al. [9]. The on-chip multiprocessor in [4] also assumed shared L2 caches, while there are instances of on-chip multiprocessors without shared caches and assuming cache-to-cache transfer for data sharing among processors within a chip [5].

In a DSM multiprocessor, the latency of the remote data access affects its performance, and several techniques to reduce it have been proposed. Page caching, migration and replication are techniques to hold remote data locally by aliasing it to the local physical page [14,15]. Addition of an extra cache that holds remote data to the memory hierarchy can also alleviate the remote access latency [16]. Moga et al. proposed the *remote victim cache*, a cache that only stores remote data replaced from the L2 cache [17].

6. Conclusions

In this paper, two design options to better utilize the cache resources in on-chip multiprocessors were examined. The effectiveness of these options was evaluated through execution-driven simulations. Using the single-level caching for the instruction and/or private data to leave more space on the L2 cache for the shared data improved the performance of the on-chip multiprocessor up to 15%. However, in FFT and Radix, this option used for the instruction degraded the performance. Therefore, this option has to be used selectively. The effectiveness of the remote victim cache was also evaluated by taking total cache area into consideration. Addition of the remote victim cache of a small size (1/8 of L2 cache) was effective in three out of four benchmark programs: it improved the performance by 4–8%. However, having L2 and victim caches of the same size was less effective than having only an L2 cache of twice the larger size except in Radix. Therefore, if a small amount of extra space is available, using it for a remote victim cache is a good idea. However, the size of the L2 cache should not be sacrificed to accommodate a large remote victim cache.

Topics of further study include the following. In this paper, a configuration of four processors and on-chip memory modules per chip was assumed as a case study. It is possible that the size of on-chip memory modules is not sufficient to hold all the instructions and the data, and off-chip memory modules are attached to the on-chip multiprocessor system. These off-chip memory modules introduce another level in the memory hierarchy. Efficient data placement and migration techniques between the on-chip and the off-chip memories need to be investigated in the future.

In this study, the access speeds of L2 and remote victim caches were kept constant while their sizes were varied. Generally speaking, the smaller the cache, the faster its access [18]. However, since the clock cycles is a discrete number, so is the cache access speed. As long as the cache size varies within "some" range, its access speed stays the same. Also, the actual access time is affected by several factors, such as semiconductor device technology used in the chip, and the structure of memory hierarchy. The effects of these factors on the design options proposed in this paper would be another topic of further research.

Acknowledgements

This research was supported in part by a National Science Foundation Grant No. MIPS 9522265.

References

[1] K.C. Yeager, Mips R10000 superscalar microprocessor, IEEE Micro 16 (2) (1996) 28–40.
[2] L. Hammond, B.A. Nayfeh, K. Olukotun, A single-chip multiprocessor, IEEE Computer 30 (9) (1997) 79–85.
[3] K. Farkas, N. Jouppi, P. Chow, Register file considerations in dynamically scheduled processors, Proceedings of International Symposium on High Performance Computer Architecture, February 1996, pp. 40–51.
[4] K. Olukotun et al., The case for a single-chip multiprocessor, Proceedings of Seventh International Conference on Architectural Support for Programming Languages and Operating Systems, ACM Press, New York, October 1996, pp. 2–11.
[5] M. Takahashi et al., A shared-bus control mechanism and a cache coherence protocol for a high-performance on-chip multiprocessor, Proceedings of International Symposium on High Performance Computer Architecture, February 1997, pp. 314–322.
[6] T. Yamauchi, L. Hammond and K. Olukotun, A single chip multiprocessor integrated with high density DRAM, Technical Report: CSL-TR-97-731, Stanford University, August 1997.
[7] H. Oi, N. Ranganathan, Utilization of cache area in on-chip multiprocessor, Proceedings of International Symposium on High Performance Computing (ISHPC'99), Kyoto, Japan, May 1999, pp. 373–380.
[8] Y. Nunomura, T. Shimizu, O. Tomisawa, M32R/D-integrating DRAM and microprocessor, IEEE Micro 17 (6) (1997) 40–48.
[9] B.A. Nayfeh, K. Olukotun, J.P. Singh, The impact of shared-cache clustering in small-scale shared-memory multiprocessors, Proceedings of International Symposium on High Performance Computer Architecture, February 1997, pp. 74–84.
[10] D. Sunada, D. Glasco, M. Flynn, ABSS v2.0: a SPARC simulator, Technical Report: CSL-TR-98-755, Stanford University, April 1998.
[11] E. Lusk, et al., Portable Programs for Parallel Processors, Holt, Rinehard & Winston, New York, 1987.

[12] S.C. Woo et al., The SPLASH-2 programs: characterization and methodological considerations, Proceedings of the 22nd International Symposium on Computer Architecture, June 1995, pp. 24–36.
[13] A. Gupta, W.-D. Weber, Cache invalidation patterns in shared-memory multiprocessor, IEEE Transactions on Computers 41 (7) (1992) 794–810.
[14] A. Saulasbury, T. Wilkinson, J. Carter, An argument for simple COMA, Proceedings of the 22nd International Symposium on Computer Architecture, June 1995, pp. 276–285.
[15] J. Laudon, D. Lenoski, The SGI origin: a ccNUMA highly scalable server, Proceedings of the 24th International Symposium on Computer Architecture, June 1997, pp. 241–251.
[16] Z. Zhang, J. Torrellas, Reducing remote conflict misses: NUMA with remote cache versus COMA, Proceedings of International Symposium on High Performance Computer Architecture, February 1997, pp. 272–281.
[17] A. Moga, M. Dubois, The effectiveness of SRAM network caches in clustered DSMs, Proceedings of the Fourth International Symposium on High Performance Computer Architecture, February 1998, pp. 103–112.
[18] N.P. Jouppi, S.J.E. Wilton, Tradeoffs in two-level on-chip caching, WRL Research Report 93/3, Digital Equipment Corporation, October 1993.

Hitoshi Oi is a Computer Architect at HAL Computer Systems, Campbell, California. He received a PhD degree in Computer Science and Engineering from University of South Florida. He received Bachelor's degree in Electronics and Communication Engineering, and Master's degree in Electrical Engineering, both from Meiji University. His research interests include computer architecture and performance evaluation, VLSI design and verification, and non-standard logic.

N. Ranganathan received the PhD degree in Computer Science from the University of Central Florida, Orlando in 1988. He is currently a professor in the Department of Computer Science and Engineering and the Center for Microelectronics Research at the University of South Florida, Tampa. His research interests include VLSI design, design automation, hardware algorithms, computer architecture and parallel processing.