

Short Paper

Adaptive Multi-Grain Remote Access Cache in Ring Based NUMA System

JONG WOOK KWAK AND CHU SHIK JHON
*Department of Electrical Engineering and Computer Science
Seoul National University
Seoul, 151-742 Korea*

Due to the ease of implementation and the alleviation of memory bottleneck effect, NUMA system with Remote Access Cache (RAC) has dominated multiprocessor systems for the past several years. In this paper, we suggest a Multi-Grain RAC to adaptively control the RAC line size for each application behavior. We simulate the NUMA systems with the Multi-Grain RAC using MINT, an event-driven memory hierarchy simulator, and analyze the performance of Multi-Grain RAC. At first, with a profile-based determination method, we verify the best RAC line size for each application and then we compare and analyze the performance differences among NUMA systems with difference RAC line size. The simulation result shows that the worst case can be mostly avoided in Multi-Grain RAC and, in addition, the results are very close to the best case with any combination of application and RAC format.

Keywords: NUMA system, remote access cache, cache line size, application behavior, multi-grain

1. INTRODUCTION

To provide an enough level of parallelism, many multiprocessor systems have been developed. One of the most representative types among them is the distributed shared memory (DSM) multiprocessor system [1]. The DSM multiprocessor system can be generally classified into UMA and NUMA with respect to whether the memory access time is identical or not. Although UMA is simple and easy to construct, it has some drawbacks in its scalability, availability, and memory bottleneck effect. To alleviate these problems, NUMA distribute its memory across each processing node. In NUMA, each processing node is usually connected via an interconnection network, thus the processor request which communicates with remote memory far from its local location has a longer response time than the request which communicates with the closer one. Therefore, the reduction of remote memory access delay is critical for overall performance in NUMA [2]. To resolve this problem, we have additionally used the Remote Access Cache (RAC) in the NUMA. The RAC is used to cache the data originated from other processing nodes and is shared by all processors on a local node. Therefore, the NUMA architecture which contains the RAC is superior to other DSM multiprocessors [3].

Received August 4, 2004; revised November 18, 2004 & April 14, 2005; accepted December 12, 2005.
Communicated by Chu-Sing Yang.

Besides, although the system types are superior to others, the memory performance is usually affected by memory access patterns, degrees of sharing, and the locality of application programs, and these characteristics have a close relationship with proper cache line size. That is, if the cache line is a coarse grain, it is advantageous in applications having a spatial locality or a sequential memory access pattern. However, if the grain is too coarse, it may cause many false sharings and unnecessary cache line invalidations. The false sharing is the ill effects of poor spatial locality within the cache line itself. On the contrary, if the cache line is a fine grain, it is advantageous in the reduction of many unnecessary cache line invalidations which are caused by the false sharings. However, if the application program has a sequential memory access pattern, the fine grain approach generates more transactions than the coarse grain one, in order to fetch the whole requested data [4].

After all, the problem is the mismatch between adequate cache line sizes and the application behaviors. However, there is no single cache line size that always satisfies the best performance for all applications and, moreover, even though we find and use the exact cache line size by profiling the application, it does not always guarantee the best result throughout the whole executions. Therefore, we propose the Multi-Grain RAC to adaptively control the cache line size for each application behavior. It can dynamically analyze the memory reference patterns in program execution time and change the line size within the predefined cache line sizes. We are especially focused on the RAC, because the RAC has high miss rates and huge miss penalties, compared to processor caches. The followings show the main contributions of our proposal.

- Multi-Grain RAC is the first try to apply the dynamic cache line management to RAC in NUMA system.
- Multi-Grain RAC does not require any modification of cache coherence protocols.
- Multi-Grain RAC can manage the different grain size for specific memory addresses at the same time.
- Multi-Grain RAC can control the prefetching data size dynamically. It can increase the cache line size and moreover it can decrease the line size more adaptively, resulting in the efficient reduction of false sharing. Moreover, Multi-Grain RAC manages the inter-cache blocks, not the intra-cache blocks, and this allows to handle the larger size granularity, resulting in more efficient prefetching for the continuous sequential access data.

2. RELATED WORK

In the NUMA system, the miss penalty for the remote memory could be several orders of magnitude higher than the local cache access time. Therefore, memory allocation or data placement in the NUMA systems is one of main issues in designing the NUMA systems. There have been several literatures to ensure the efficient memory management. Wilson *et al.* [5] show the dynamic page placement scheme to maximize the memory performance by improving the locality in the NUMA systems. Bhuyan *et al.* [6] show the impact of CC-NUMA memory management on the application of multistage switching networks. Several studies have been proposed to manage the dynamic cache access block.

Stride-based prefetching is an example of adaptive cache block management by the hardware implementation [7]. Sub-block placement is another example [1]. The former technique is for reducing the miss rate by the detection of memory request stride in hardware, and the latter one is for reducing the miss penalty by the division of the cache block into sub-cache blocks. Adjustable block management [8] and adapting cache line size algorithm [9] are also proposed. However, [8] should modify the cache coherence protocol and this factor critically impacts on implementation complexity of the multi-processor system design. The concept of [9] is similar to sub-block placement, which manages sub-cache lines within a physical cache line.

3. MULTI-GRAIN REMOTE ACCESS CACHE

Before designing the Multi-Grain RAC, we first choose 32 byte, 64 byte, and 128 byte as candidate RAC line sizes and each size is used as the Minimum (32 byte), Initial (64 byte), and Maximum (128 byte) data transfer between RAC and remote memory. Fig. 1 shows the basic idea of Multi-Grain RAC. The shaded rectangle in Fig. 1 shows the referenced cache line (32 byte), and four referenced cache lines make one *virtual cache line*. The algorithm for deciding RAC line size is as followings. The system first begins the analysis of the processor's memory reference pattern and checks the referenced cache line status with the initial 64 byte cache line size. When the reference miss occurs in the RAC, the system analyzes the previous reference patterns, and decides how many lines are fetched in next time. With this operation, the system makes the state transition into the Minimum Line or Maximum Line. Generally, if reference patterns are continuous or sequential, the transition moves gradually to the Maximum Line, and if reference patterns are scattered across cache modules, the transition moves to the Minimum Line.

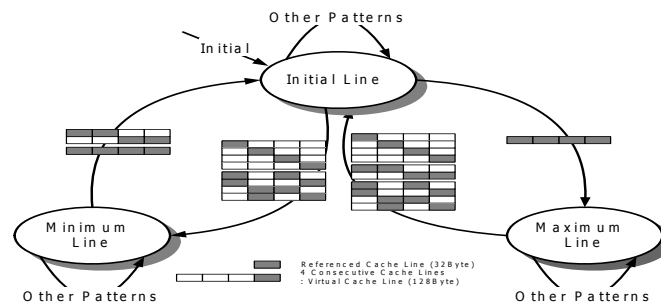


Fig. 1. State transition diagram for multi-grain RAC.

The hardware model of Multi-Grain RAC is shown in Fig. 2. The basic structure of the Multi-Grain RAC is similar to existing RAC, but the Multi-Grain RAC requires four consecutive cache modules. The RAC is implemented as a direct-mapped cache to ensure the address continuity. Another difference in Multi-Grain RAC is the R bit in each cache module. R bit specifies whether the cache line is re-referenced or not and it indicates an additional reference of the processor, after firstly being placed in each module. And then,

when the cache line is replaced, four R bit combination is used as the state transition of cache line size. Finally, the Granularity Determinator, which interfaces with Processor Local Controller (PLC), decides the state of cache line size. As shown in Fig. 2, the tag bits and R bit in each module are transferred to the Granularity Determinator. The internal structure of the Granularity Determinator is shown in the second diagram of Fig. 2. It is composed of a Granularity Specifier, a Reference Pattern Table, and a Split & Merge Counter. The goals of these sub-modules are for changing the state and deciding the next fetch size, when the reference miss of the processor occurs.

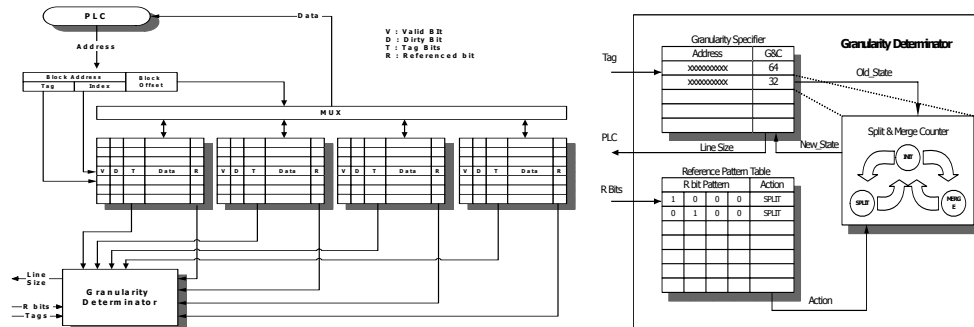


Fig. 2. The multi-grain RAC and granularity determinator.

In Multi-Grain RAC, the memory address is transferred from the PLC. For this address, the RAC returns the requested data to the PLC and sets the R bit in each referenced cache block. When any RAC line is replaced in the future, the RAC examines four R bits in four consecutive cache modules which include the replaced cache block, and then these R bits are transferred to the Reference Pattern Table. If the same R bit combination is found in the Reference Pattern Table, the matched Action field is produced, and this output initiates the G&C (Grain & Counter) field of Granularity Specifier in the Split & Merge Counter. G&C field has current granularity of specified address and Split & Merge Counter with new state and old state. In the Counter, the old state and the current input produce a new state, and this makes the granularity change. Finally, the counter stores a new grain into the G&C field and line size is transferred to PLC.

Table 1 shows the Reference Pattern Table. The table produces one of four actions. "SPLIT" or "MERGE" are transferred to the Split & Merge Counter with proper state transitions. "No Action" produces null-output, i.e., it does not change the state. In case of "INIT", it acts as "SPLIT" when the state of current line is Maximum Line, "MERGE" when the state of current line is Minimum Line, and "No Action" when the state of the current line is Initial Line.

Fig. 3 shows the state transition diagram of the Split & Merge Counter. If the state transition changes the state into DIVIDE, the 64(128) byte cache line size is changed into the 32(64) byte. If the current line size is already 32 byte, the line size does not change. Similarly, if the state transition changes the state into COMBINE, the 32(64) byte line size is changed into the 64(128) byte, and if current line size is 128 byte, the line size does not change. After the counter changes its state, the adjacent four R bits are reset.

Table 1. Reference pattern table.

The Value of Adjacent R bits				Action
1	0	0	0	SPLIT
0	1	0	0	SPLIT
0	0	1	0	SPLIT
0	0	0	1	SPLIT
1	0	1	0	SPLIT
1	0	0	1	SPLIT
0	1	1	0	SPLIT
0	1	0	1	SPLIT
1	1	1	1	MERGE
1	1	0	0	INIT
0	0	1	1	INIT
Other Patterns				NO ACTION

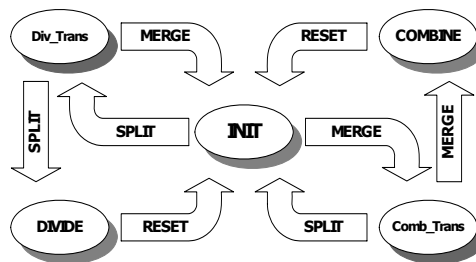


Fig. 3. The state transition diagram of SPLIT & MERGE counter.

The states, Div_Trans and Comb_Trans, are a means for prohibiting the frequent state changes and ignoring partial or short temporal granularity change, e.g., job distribution or synchronization.

Let's consider the following two code examples. If we assume that the array is stored in row major order, the first code accesses all the words in sequential and continuous. On the contrary, the second code skips memory in distances of 1,000 words. That is, memory access of the second code is scattered across the cache lines about the distances of 1,000 words. If we assume that the array does not fit in the cache, these two examples may cause frequent cache misses. However, once recognized by the Granularity Determinator in Multi-Grain RAC, the miss rate in the first code can be reduced by improving spatial locality through the increase of granularity in the memory address of the array *cont*. In case of second code, the memory access pattern is scattered across the cache lines and the Granularity Determinator reduces the size of granularity in the memory address of the array *scatt*.

<pre>for (i = 0; i < 5,000; i = i + 1) for (j = 0; j < 1,000; j = j + 1) cont[i][j] = 2 * cont[i][j];</pre>	<pre>for (i = 0; i < 1,000; i = i + 1) for (j = 0; j < 5,000; j = j + 1) scatt[j][i] = 2 * scatt[j][i];</pre>
---	---

Table 2 shows the additional hardware costs of the Multi-Grain RAC for each RAC size. By Eq. (1), we calculated the number of required transistors based on the overall roadmap technology characteristics [10].

$$\text{Additional_Cost}(\%) = \frac{\text{Multigrain_Control_Portion}}{\text{SRAM_Portion} + \text{Control_Portion}} \times 100(\%) \quad (1)$$

Table 2. Additional hardware cost of multi-grain RAC (Unit: required Tr.).

Cache Size	Tag-Arrays	Data-Arrays	Valid Field	Dirty Field	R bit	Tag-Arrays	Grain Field	Counter Field	Additional Cost (%)
64K	8	128	0.5	0.5	0.5	8	1	1.5	8.02
128K	7.5	128	0.5	0.5	0.5	7.5	1	1.5	7.69
256K	7	128	0.5	0.5	0.5	7	1	1.5	7.35
512K	6.5	128	0.5	0.5	0.5	6.5	1	1.5	7.01

In Table 2, the hardware requirement of conventional RAC is the field from Tag-Arrays to Dirty Field, and the additional hardware cost that is required in the Multi-Grain RAC is the field from the R bit to Counter Field. As shown in Table 2, the additional hardware cost is mostly less than 8.02% and the cost ratio gradually decreases as the cache size increases. Thus, the hardware cost of Multi-Grain RAC is modest and it may be trivial as the cache size increases.

4. PERFORMANCE EVALUATION

In this section, we simulate and analyze the relationship between cache line sizes and the application behaviors and further evaluate the performance of the Multi-Grain RAC. We use the execution-driven simulator, MINT [11], to evaluate the performance of the Multi-Grain RAC, and we select FFT, LU, OCEAN, BARNES, RADIX, and WATER from SPLASH-2 [12] as workloads. We conduct our simulation in ring-based eight 4-processor SMP node (i.e., 32 processors) NUMA system.

4.1 The Results of Profile-Based Determination

At first, we find the best cache line size for each application. Then we compare the results of the Multi-Grain RAC with that of the best cache line size. The miss rates of the profile-based determination are shown in Fig. 4. The cache size was changed from 64K to 512K, and the cache line size was changed from 32 byte to 128 byte. As shown in Fig. 4, the miss rates (%) of FFT, LU and OCEAN decrease as the cache line size increases. On the other hand, the miss rates of BARNES and RADIX increase as the cache line size increases. From the results, we can infer that FFT, LU, and OCEAN are coarse grain applications, and BARNES and RADIX are fine grain applications. Compared to other programs, WATER has a unique result. It is a medium grain application. In case of WATER, the miss rate decreases as the cache line size increases from 32 byte to 64 byte, whereas the miss rate increases as the cache line size increases from 64 byte to 128 byte.

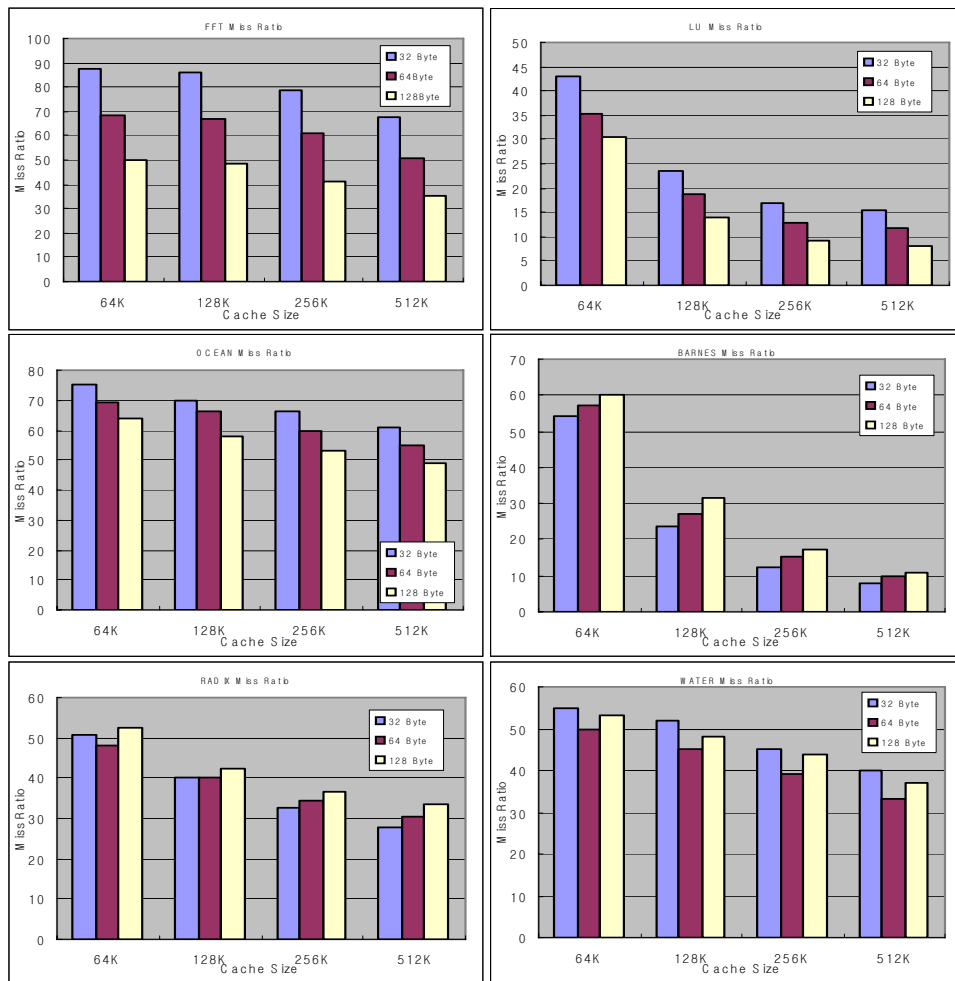


Fig. 4. The miss ratio for each application.

In addition to these results, we can infer another fact; the relationship between the cache size and the cache line size. In Fig. 4, we can see that the miss rate of FFT with 128 byte line size in 64K RAC is lower than that of FFT with 32 byte line size in 512K RAC. This result indicates that, although the cache is one of key resources and its size decision is important, the adequate cache line size decision is sometimes more critical. On the other hand, the miss rate of LU is much more affected by the actual cache size, not by the cache line size. It means that, although LU has a sequential memory access pattern, LU requires much more working sets than FFT, thus LU is more affected by the actual cache size, not cache line size. We can find similar results in other applications. RADIX, which is shown in the miss rate of 32 byte line size with 256K RAC and 128 byte line size with 512K RAC, is much more affected by cache line size. BARNES, on the other hand, is much more affected by the actual cache size. From these results, we can infer the followings. LU and BARNES require a large working set, whereas FFT and

RADIX require a small working set, despite their different memory access patterns. These results additionally show that the adequate cache line size decision is more important when the applications require a small working set. From the results, we can confirm that there is no single “optimal” cache line size for all applications. Therefore, we should use the Multi-Grain RAC and dynamically change the cache line size for each application at the execution time. Anyway, we conclude that, in addition to the fact that the cache line size is sometimes more critical than the actual cache size, the best cache line size for FFT, LU, and OCEAN are 128 byte, BARNES and RADIX are 32 byte, and WATER is 64 byte.

4.2 The Results of Dynamic Determination

In this section, we evaluate the performance of Multi-Grain RAC by the dynamic determination. We first analyze the “Normalized Miss rate Difference (NMD)” for each application, with the data presented in section 4.1. To describe the NMD, we use the following equation.

$$NMD(\%) = \frac{(MR_{WC} - MR_{BC})}{MR_{BC}} \times 100 \quad (2)$$

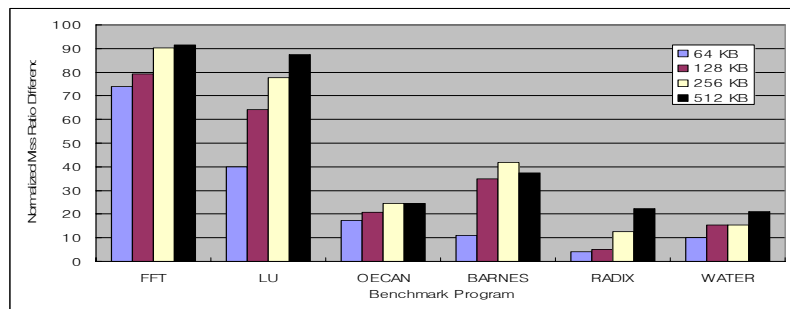


Fig. 5. Normalized miss ratio difference (NMD).

In Eq. (2), MR stands for Miss Rate, and the subscript BC and WC stands for the Best Case and the Worst Case results in Fig. 4. Fig. 5 shows the NMD (%). As shown in Fig. 5, miss rate differences between the best case and the worst case in FFT and LU are more than 75% and 40% respectively, and OCEAN, BARNES, RADIX, and WATER are more than 20% in average. Furthermore, as the RAC size increases, the percentage of NMD is generally increased. For the trend of today’s cache size increase, this result indicates that the NMD becomes larger and larger as time goes on, and it emphasizes the importance of the adequate cache line size decision.

The simulation results of Multi-Grain RAC are shown in Fig. 6. The cache size was changed from 64K to 512K, and comparison targets are the Worst Case, Profiled Case, and Multi-Grain. The Worst Case is the worst results line size in profile-based determination and Profiled Case is the best case result. Multi-Grain is the result of dynamic determination which uses the Multi-Grain RAC.

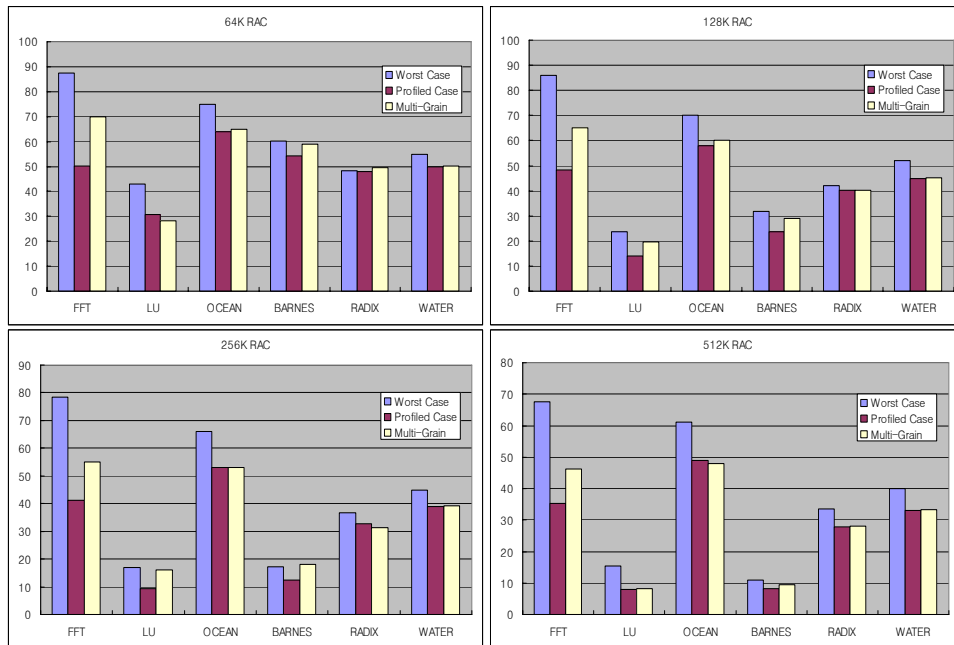


Fig. 6. The miss ratio of multi-grain policy.

As shown in Fig. 6, Multi-Grain RAC in FFT outperforms about 21% in average, compared to the Worst Case. BARNES has the similar result pattern with FFT. Although the Multi-Grain RAC in BARNES slightly inferior to Profiled Case, it avoids the Worst Case as well. The Multi-Grain RAC in LU also outperforms the Worst Case for every RAC size, and in cases of 64K and 512K, the results of Multi-Grain RAC are very similar to the Profiled Case, or even outperform the Profiled Case. The result patterns of OCEAN are similar to those of LU. It avoids the Worst Case and sometimes outperforms Profiled Case. The reason of these results is that the Multi-Grain RAC can control and adjust the exact change of the granularity, during the specific period of the execution time. Further, we can also infer that the best results presented in section 4.1 are not true optimal results for the whole program execution time. From these results, we can conclude that the Multi-Grain RAC can trace the temporal change of a working set or locality due to the job distribution or synchronization, and it can reflect these effects as well. The results of Multi-Grain RAC in RADIX and WATER are generally similar with each other. They naturally avoid the Worst Case and the miss rates of Multi-Grain RAC are very similar with those of Profiled Case.

Fig. 7 shows the average execution time for all cache sizes. In Fig. 7, we break each result down into following categories: *Remote Data*, *Remote Overhead*, and *Local Data*. Remote Data is the traffic caused by all data transferred between nodes, and Remote Overhead is the traffic related with remote data request messages. Local Data is data transmitted by processor requests. Remote Data can be broken further down into *Remote shared*, *Remote cold*, *Remote capacity*, and *Remote writeback*. The results are normalized to those of the Worst Case. As shown in Fig. 7, the result patterns of execution time

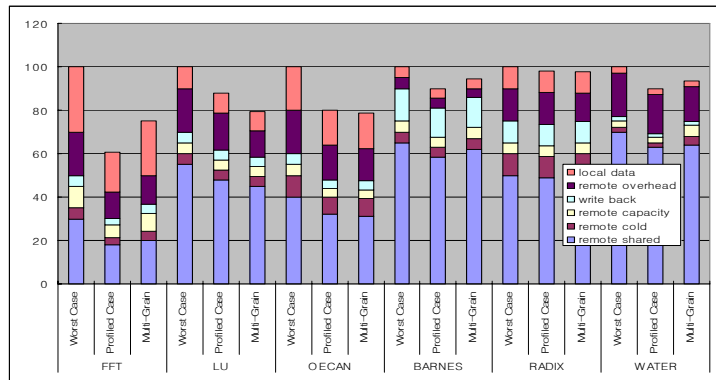


Fig. 7. The breakdown of average execution time.

are similar to those of miss rate results. It means that the miss rates of RAC are critical for system performance, due to the huge miss penalty of RAC. In the breakdown charts, the performance improvement in the Profiled Case is mainly originated from the reduction of *Remote Data* and *Remote Overhead*. Compared to other categories, these two categories are generally reduced and this makes the overall performance improvement. In case of Multi-Grain, it avoids the Worst Case for all applications due to its dynamic cache line size management, and it even outperforms the Profiled Case in LU. In cases of OECAN and RADIX, they slightly outperform the Profiled Case as well. Although the multi-grain results of FFT, BARNES, and WATER require more execution time than Profiled Case, they are better than Worst Case, and the multi-grain results are very near to Profiled Case. In summary, the results of this study indicate that the Multi-Grain RAC generally avoids the worst case, and the miss rate and the execution time are similar to, or sometimes better than, those of the Profiled Case, for any application programs with any cache sizes and any cache line sizes.

5. CONCLUSION

To achieve the best match between cache line sizes and application behaviors, we proposed the Multi-Grain RAC to adaptively control the RAC line size, for each application behavior in program execution time. Multi-Grain RAC can control not only the granularity of each application program but also can change the granularity even in specific parts of memory address within one application program. The hardware requirements of our scheme are modest, and it can be further improved by changing candidate cache line size. In the simulation, with the profile-based determination, we found the best cache line size for each application and discussed the relationship between cache size and cache line size. In addition, we obtained the NMD and it shows the importance of proper cache line size selection, once more. Then, we evaluated the performance of the Multi-Grain RAC. The result shows that the worst case can be avoided mostly and the results are very close to the best case with any combination of application and RAC format.

REFERENCES

1. J. L. Hennessy and D. A. Patterson, *Computer Architecture: A Quantitative Approach*, 2nd ed., Morgan Kaufmann Publishers, Inc., San Francisco, 1996.
2. K. Hwang and Z. Xu, *Scalable Parallel Computing: Technology, Architecture, Programming*, McGraw-Hill, New York, 1998.
3. Z. Zhang and J. Torrelas, "Reducing remote conflict misses: NUMA with remote cache versus COMA," in *Proceedings of the 3rd IEEE Symposium on High Performance Computer Architecture*, 1997, pp. 272-281.
4. S. Dwarkadas, *et al.*, "Comparative evaluation of fine- and coarse-grain approaches for software distributed shared memory," in *Proceedings of the IEEE Symposium on High performance Computer Architecture*, 1998, pp. 260-269.
5. K. Wilson and B. Aglietti, "Dynamic page placement to improve locality in CC-NUMA multiprocessors for TPC-C," in *Proceedings of the ACM/IEEE Conference on Supercomputing*, 2001, pp. 33-41.
6. L. Bhuyan, *et al.*, "Impact of CC-NUMA memory management policies on the application performance of multistage switching networks," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 11, 2000, pp. 230-246.
7. M. J. Charney and A. P. Reeves, "Generalized correlation based hardware prefetching," Technical Report No. EE-CEG-95-1, School of Electrical Engineering, Cornell University, New York, 1995.
8. C. Dubnicki and T. J. LeBlanc, "Adjustable block size coherent cache," *International Symposium on Computer Architecture*, 1992, pp. 170-180.
9. A. V. Veidenbaum, W. Tang, R. Gupta, A. Nicolau, and X. Ji, "Adapting cache line size to application behavior," in *Proceedings of the 13 ACM International Conference on Supercomputing*, 1999, pp. 145-154.
10. Semiconductor Industry Association, *The National Technology Roadmap for Semiconductors*, 1994.
11. J. E. Veenstra and R. J. Fowler, "MINT: a front end for efficient simulation of shared-memory multiprocessors," in *Proceedings of the 2nd International Workshop on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, 1994, pp. 201-207.
12. S. C. Woo, *et al.*, "Methodological considerations and characterization of the splash-2 parallel application suite," in *Proceedings of the 22nd International Symposium on Computer Architecture*, 1995, pp. 24-36.

Jong Wook Kwak (郭鍾旭) received the B.S. degree in Computer Engineering from Kyungpook National University, Taegu, Korea in 1998, the M.S. degree in Computer Engineering from Seoul National University, Seoul, Korea in 2001, and the Ph.D. degree in Electrical Engineering and Computer Science from Seoul National University, Seoul, Korea in 2006. He is currently a senior engineer in SOC R&D Center, Samsung Electronics Co., Ltd. His research interests are computer architecture, high-performance parallel computing, and low-power embedded systems.

Chu Shik Jhon (全洲植) received the B.S. degree in Applied Mathematics from Seoul National University, Seoul, Korea in 1975, the M.S. degree in Computer Engineering from Korea Advanced Institute of Science and Technology, Taegeon, Korea in 1977, and the Ph.D. degree in Computer Engineering from the University of Utah in 1982. He was an Associated Professor in Computer Engineering Department at Iowa State University from 1983 to 1985. He is currently a professor in the Department of Electrical Engineering and Computer Science at Seoul National University. His research interests include parallel computing, and VLSI design automation.