JCSE Journal of Computing Science and Engineering

Regular Paper

*Journal of Computing Science and Engineering,*
*Vol. 8, No. 1, March 2014, pp. 25-33*

# Comparing Separate and Statically-Partitioned Caches for Time-Predictable Multicore Processors

**Lan Wu, Yiqiang Ding, and Wei Zhang***

Department of Electrical and Computer Engineering, Virginia Commonwealth University, Richmond, VA, USA
**Wul2@vcu.edu, dingy4@vcu.edu, wzhang4@vcu.edu**

## Abstract

In this paper, we quantitatively compare two different time-predictable multicore cache architectures, separate and statically-partitioned caches, through extensive simulation. Current research trends primarily focus on partitioned-cache architectures in order to achieve time predictability for hard real-time multicore based systems, and our experiments reveal that separate caches actually lead to much better performance and energy efficiency when compared to statically-partitioned caches, and both of them are adequate for timing analysis for real-time multicore applications.

**Category:** Embedded computing

**Keywords:** Performance; Reliability; Hard real-time systems; Multicore processors; Cache architectures; WCET analysis

## 1. INTRODUCTION

Nowadays, multicore processors have reached the mainstream of chip design. However, for real-time systems, especially hard real-time systems, it is very challenging to accurately estimate worst-case execution time (WCET) on a multicore processor. This is because in addition to the complexity of WCET analysis for uniprocessors, the execution time of a thread can be significantly affected by other concurrent threads in a multicore platform due to inter-thread interference in accessing shared resources, such as shared buses and shared caches. This problem can significantly aggravate the complexity of multicore timing analysis, especially if there are a large number of cores, making WCET analysis for multicore chips extremely difficult, if not impossible.

To solve this problem, it is desirable to design a time-predictable multicore architecture that can adequately simplify the complexity of multicore timing analysis. The goal of designing this kind of system is to eliminate or easily bind the timing unpredictability caused by inter-thread interferences in accessing shared resources, while limiting the impact on performance and/or energy dissipation. Recently, Paolieri et al. [1] proposed a time-predictable architecture to provide hardware support for WCET analysis of hard real-time multicore systems. They developed a statically-partitioned, shared L2 cache and an arbiter mechanism to access the bus and the cache to either avoid or maximally bind the interference due to accessing shared resources, and from this, the WCET of real-time threads running on this multicore architecture can be derived. The work of Paolieri et al. [1] was one of the earliest to design time-predictable multicore architectures for hard real-time systems by considering both cache and bus interferences. However, their work did not explore multicore architectures with separate L1 and L2

Open Access http://dx.doi.org/10.5626/JCSE.2014.8.1.25 http://jcse.kiise.org



**Received** 23 September 2014; **Revised** 9 January 2014; **Accepted** 4 February 2014
*Corresponding Author

 pISSN: 1976-4677 eISSN: 2093-8020

caches, which, based on our study presented in this paper, can also achieve time predictability and actually have better performance and energy efficiency.

We quantitatively compare the partitioned-cache architecture, which had been studied by Paolieri et al. [1], with separate-cache architectures and separate buses using extensive simulations. Our experiments quantitatively indicate that the separate-cache architecture, although simple in design, actually results in superior performance and energy efficiency for all real-time and Standard Performance Evaluation Corporation (SPEC) 2006 benchmarks we employed. Considering that both separate and partitioned-cache architectures are suitable for complexity-effective timing analysis for hard real-time systems, our research reveals that instead of primarily focusing on developing elegant partitioned schemes of shared caches for hard real-time multicore systems, it is perhaps more meaningful to revisit multicore chips with separate caches in order to achieve time predictability with improved performance and energy efficiency.

## II. RELATED WORK

### A. Prior Works on WCET Analysis

WCET analysis has become an area of intense interest in the field of real-time systems in the last two or three decades. A recent survey provides an extensive review of the state-of-the-art techniques [2]. Most of the prior efforts, however, focus on WCET analysis for uniprocessors [3-5]. Recently, a number of efforts have been made to analyze WCET for multicore processors [6-13]. However, these studies are already very complex for multicore processors with a small number of cores (e.g., 2 or 4 cores). In manycore or multicore processors with a higher number of cores, the interference among concurrent threads will significantly increase, making WCET analysis extremely hard, if not impossible. Thus, it becomes crucial to design a time-predictable multicore/manycore architecture to make WCET analysis possible, providing a basis to enable safe and reliable hard real-time computing.

### B. Prior Works on Time-Predictable Architectures

The design of real-time systems with time predictability has been gaining increased attention. Reineke et al. [14] first studied the predictability of common cache replacement policies. They introduced three metrics to capture the behaviors of caches and gave quantitative, analytical results for the predictability of cache replacement policies. Edwards and Lee [15] developed an architecture named precision timed (PRET) machine which improved time predictability by fixing instruction execution times and balancing program paths to deal with repeatable concurrent behaviors. Hansson et al. [16] proposed an approach named CoMPSoC to provide templates for the composition of predictable stream-processing architectures. Recently, Paolieri et al. [1] developed a partitioned shared L2 cache and a bus arbiter mechanism to reasonably bind the interference of shared resources and WCET for multicore processors. While the work of Paolieri et al. [1] was one of the earliest efforts to design time-predictable multicore architectures for hard real-time systems, their work did not explore the full cache design space to derive the best architecture to ensure time predictability while providing excellent performance and/or energy consumption. Specifically, Paolieri et al. [1] did not consider a multicore architecture with separate L2 caches, which can also achieve time predictability. Our study demonstrates that a multicore architecture with separate L2 caches achieves better performance and energy efficiency than an equivalent multicore processor with a statically-partitioned L2 cache. While both separate and statically-partitioned caches are not new to the best of our knowledge, this is the first work to comparatively evaluate both the performance and energy efficiency of these two cache architectures for real-time multicore processors.

## III. SEPARATE VS. STATICALLY-PARTITIONED CACHE ARCHITECTURES

### A. Multicore Processors with Separate L2 Caches

The first model that we study to achieve time predictability is a multicore processor with a separate L2 cache and bus system. Fig. 1 gives an overview of this architecture in which each core has its own L1 instruction and data caches as well as a private L2 cache. The L1 caches can access their own L2 cache through the private bus. All the L2 caches share access to the same memory.

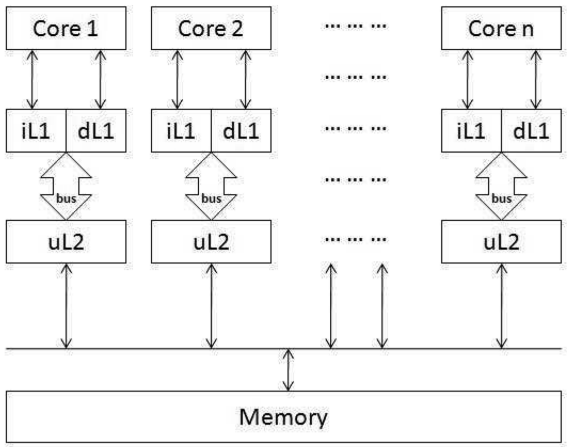



**Fig. 1.** Model I: a multicore processor with separate L2 caches and buses.

In this separate-cache architecture, since there is no shared L2 cache, there is absolutely no inter-thread cache interference. As a result, the timing analysis for the cache subsystem is much easier and can be done by extending prior work on cache timing analysis for uniprocessors [5, 17].

Nevertheless, for the shared bus between the L2 cache and the memory, it may happen that two or more hard real-time tasks (HRTs) running on different cores attempt to access the bus at the same time. In this case, the *Upper Bound Delay* (*UBD*) of the bus access depends on the arbitration policy. In this paper, in a manner similar to that by Paoleri et al. [1], we assume a *Round-Robin* policy between the HRTs, assigning the same priority to all the requests from HRTs to achieve fairness among HRTs and to prioritize HRTs over non-hard real-time tasks (NHRTs) in order to avoid unnecessary delays for HRTs. Obviously, for the *Round-Robin* policy, the maximum delay a request from an HRT may suffer from other HRTs is bound by the total number of HRTs that can issue a bus request at the same time [1], which is a figure that can be calculated by using Eq. (1). In this equation, $UBD_{HRTS}$ is the *Upper Bound Delay* that an HRT bus request may suffer from other HRTs; $N_{HRT}$ is the number of HRTs running at the same time in the processor, which is has an upper bound as a result of the number of cores; and $L_{bus}$ is the time needed to send data through the bus.

$$UBD_{HRTs} = (N_{HRT} - 1) * L_{bus} \quad (1)$$

While an HRT bus request can be prioritized over an NHRT bus request if they are issued in the same cycle, it may happen that the HRT request arrives just one cycle after the request coming from the NHRT had been already granted access to the bus. In this case, the maximum delay an HRT can suffer from the NHRT (denoted as $UBD_{NHRT}$) can be computed with Eq. (2).

$$UBD_{NHRT} = L_{bus} - 1 \quad (2)$$

Overall, the maximum delay of an HRT bus request depends on the combination of the effects of the bus requests from both HRTs and NHRTs, which can be calculated with Eq. (3).

$$UBD = UBD_{HRTs} + UBD_{NHRT} = N_{HRT} * L_{bus} - 1 \quad (3)$$

In a manner similar to the work of Paolieri et al. [1], this paper focuses on addressing time unpredictability caused by shared caches and buses in a multicore platform, and we assume the shared memory is time-predictable. Therefore, based on the discussion above, the WCET of HRTs running on multicore processors with separate L2 caches can be predicted without incurring prohibitive complexity for timing analysis.

### B. A Multicore Processor with a Statically-Partitioned L2 Cache

Another time-predictable architecture we comparatively evaluate is that of a multicore processor with a shared L2 cache and bus system. It is derived from the system designed by Paolieri et al. [1], and Fig. 2 gives an overview of this architecture. In this architecture, each core also has its own L1 instruction cache and L1 data cache. However, they share an L2 cache through a shared bus. To avoid inter-thread cache interference, the L2 cache is partitioned, and each core can only access a predefined set of cache banks. Just like in the separate-cache architecture, all the cores share access to the same memory as well.

For the partitioned L2 cache architecture with a shared bus, the execution time of an application running on one of the cores must consider the interference from both the shared bus accesses and shared L2 cache accesses. According to Paoleri et al. [1], the *UBD* for bus access interference (i.e., $UBD_{bus}$) can be calculated with Eq. (4):

$$UBD_{bus} = N_{HRT} * L_{bus} - 1. \quad (4)$$

The *UBD* for L2 cache bank access interference (i.e., $UBD_{bank}$) can be upper bound with Eq. (5), where $L_{bank}$ is the bank access latency.

$$UBD_{bank} = N_{HRT} * L_{bank} - 1 \quad (5)$$

Since bank access typically takes a longer time than bus access, the bus latency can overlap while accessing the bank. However, if bank access takes less time than bus access, then the bank conflict effect can be hidden because the time to access the bank can be overlapped by the bus latency. Thus in general, the total *UBD* is given

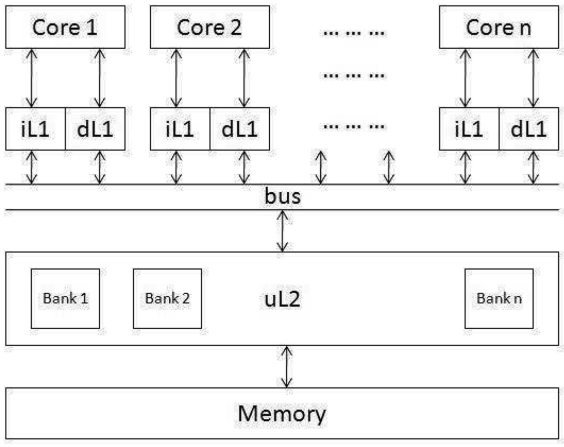



**Fig. 2.** Model II: a multicore processor with a shared (and partitioned) L2 cache and a shared bus.

by Eq. (6). More details in calculating *UBD* for the partitioned-cache architecture can be found in the work of Paoleri et al. [1].

$$UBD = N_{HRT} * max(L_{bank}, L_{bus}) - 1 \quad (6)$$

Paolieri et al. [1] have discussed two cache partitioning techniques, i.e., *columinzation* and *bankization*, in their work. The *UBD* for columnizations can be calculated with Eq. (5) and the *UBD* for bankization can be calculated with Eq. (4). With the designed WCET computation mode [1], every request to access a shared resource will artificially delay the total execution time by *UBD* cycles, which is the maximum delay. Therefore, the worst-case delay of inter-thread interference caused by accessing shared caches and buses is already considered in the WCET computation mode, and hence WCET can be calculated based on the analysis tools for single core systems.

### C. Comparison between Separate and Statically-Partitioned Caches

In this paper, we conduct extensive experiments to quantitatively compare the performance and energy consumption of the two aforementioned time-predictable multicore cache architectures to find out which one is better for real-time multicore computers. Relative to a statically-partitioned cache, a separate cache has the following advantages. First, a separate L2 cache is smaller than a centralized (yet partitioned) cache, thus a separate cache can have shorter access latency and lower power consumption per access. Second, in a separate-cache architecture, since each L1 cache can access the L2 cache through a private bus, the delay due to bus arbitration and interference per L1 miss is avoided. Both of these factors can lead to better performance and energy efficiency for the separate-cache architecture, a fact which is confirmed through our detailed simulations incorporating the cache and bus access latencies into a cycle-accurate multicore processor simulator.

On the other hand, a shared cache may use the aggregate cache space more efficiently since different threads may have different memory footprints and various memory access behaviors. However, this advantage does not hold in a statically-partitioned (though shared) cache, as each core can still only use a given set of cache banks (i.e., partitions) to avoid inter-thread cache interferences, unless the partition is performed through a dynamic and adaptive approach. The dynamic partitioning, however, would negatively affect the time predictability and hence may not be suitable for use in hard real-time systems.

## IV. EVALUATION METHODOLOGY

To compare the performance and energy efficiency of these two cache architectures for use in hard real-time multicore processors, we made extensive experiments based on extended SimpleScalar 3.0 [18]. SimpleScalar is originally designed to simulate a superscalar uniprocessor. We extended this cycle-accurate simulator to simulate the two architectural models described in Section III. We used CACTI [19] to get the latencies of the L1 and L2 caches and energy consumption per access, and these values were incorporated into the simulator to derive the final execution time and energy dissipation.

In our experiments, we simulate two four-core processors, one with the partitioned L2 cache and another with the separate L2 caches. The L1 instruction and data caches are private to each core in both processors. To make a fair comparison, we evaluate multicores with par-

**Table 1.** Real-time benchmark characteristics and description

| Benchmark | Byte | Source | Description |
|---|---|---|---|
| bs | 4248 | Malardalen | Binary search |
| cover | 5026 | Malardalen | Program for testing many paths |
| expint | 4288 | Malardalen | Computing an exponential integral |
| fdct | 8863 | Malardalen | Fast discrete cosine transform |
| fibcall | 3499 | Malardalen | Iterative Fibonacci calculation |
| insertsort | 3892 | Malardalen | Insertion sort on a reversed array |
| jfdctint | 16028 | Malardalen | Discrete-cosine transformation |
| ludcmp | 5160 | Malardalen | Read ten values, output half to LCD |
| matmul | 3737 | Malardalen | Multiplication of two 20×20 matrices |
| minver | 5805 | Malardalen | Inversion of floating point matrix |
| qsort | 4535 | Malardalen | Non-recursive quick sort algorithm |
| select | 4494 | Malardalen | Select the *N*-th largest number |

titioned caches and separate caches that have the same aggregate cache size, block size, set associativity, and replacement policy.

Generally, there are two different cache partition techniques controlled via software: *columnization* and *bankization*. While *columnization* partitions the cache into columns, *bankization* partitions the cache into banks. As discussed by Paoleri et al. [1], in *columnization*, different threads can still access the same bank, leading to bank access conflicts. In contrast, *bankization* can successfully avoid the bank access conflicts and lead to tighter WCET estimation. Therefore, in this work, we only implement the *bankization* technique in our simulator to evaluate the partitioned-cache architecture. We also assume that the bus access time without any interference takes two CPU cycles and the bus arbitrator takes one CPU cycle.

For HRTs, we use real-time benchmarks from Malardalen WCET benchmarks [2], listed in Table 1. The cache configuration for these benchmarks is shown in Table 2. The latency and energy in the table is calculated using CACTI [19].

To ensure the generality of our comparative evaluation, we also randomly select some benchmarks from SPEC CPU2006 [20], which is an industry standardized CPU-intensive benchmark suite. The selected SPEC CPU2006 benchmarks are used to represent typical non-real-time applications, listed in Table 3. Since the SPEC 2006 benchmarks are much larger than the real-time benchmarks, we use the cache configuration (i.e., configuration II) shown in Table 4 for SPEC 2006 benchmarks.

## V. EXPERIMENTAL RESULTS

### A. Performance and Energy Consumption Comparison of Real-Time Benchmarks

We divide the real-time benchmarks into three groups with four benchmarks per group. Each of the four benchmarks in a group runs on one of the four cores, and the assignment of a benchmark to a core is based on the order shown in Table 5. Fig. 3 compares the bus access cycles

**Table 2.** Configuration I for separate and statically partitioned-cache architectures of the four-core processor used to evaluate Malardalen real-time benchmarks

| | Size (kB) | Bsize (B) | Assoc | Bank_num | Latency (ns) | Energy (nJ) |
|---|---|---|---|---|---|---|
| iL1 | 8 | 8 | 4 | 1 | 0.3674 | 0.0055 |
| dL1 | 8 | 8 | 4 | 1 | 0.3674 | 0.0055 |
| L2 (for separate L2 cache) | 32 | 32 | 16 | 1 | 2.5198 | 0.6480 |
| L2 (for shared, partitioned L2 cache) | 128 | 32 | 16 | 4 | 4.1659 | 1.0346 |

**Table 3.** SPEC CPU2006 benchmark description

| Benchmark | Source | Description |
|---|---|---|
| bzip2 | INT | Compression |
| mcf | INT | Combinatorial optimization |
| hmmer | INT | Search gene sequence |
| sjeng | INT | Artificial intelligence: chess |
| lbm | FPT | Fluid dynamics |
| sphinx3 | FPT | Speech recognition |

**Table 4.** Configuration II for separate and statically partitioned-cache architectures of the four-core processor evaluated with the SPEC CPU2006 benchmarks

| | Size (kB) | Bsize (B) | Assoc | Bank_num | Latency (ns) | Energy (nJ) |
|---|---|---|---|---|---|---|
| iL1 | 32 | 16 | 4 | 1 | 0.4454 | 0.0151 |
| dL1 | 32 | 16 | 4 | 1 | 0.4454 | 0.0151 |
| L2 (For separate L2 cache) | 256 | 32 | 16 | 1 | 3.0853 | 0.8671 |
| L2 (For shared, partitioned L2 cache) | 1000 | 32 | 16 | 4 | 4.6496 | 1.4419 |

**Table 5.** Real-time benchmark test groups in our experiments

| Core | Group 1 | Group 2 | Group 3 |
|---|---|---|---|
| 0 | bs | fibcall | matmul |
| 1 | cover | insertsort | minver |
| 2 | expint | jfdct | qsort |
| 3 | fdct | ludcmp | select |

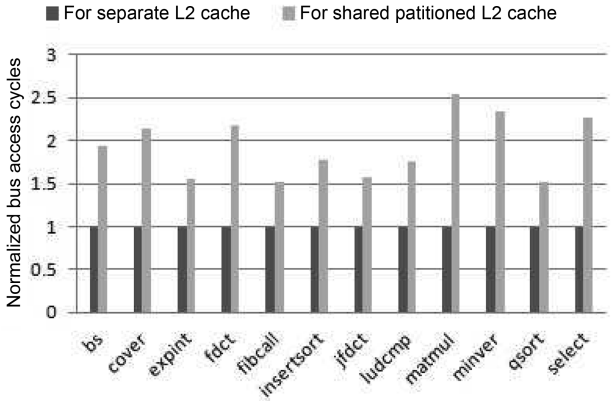



**Fig. 3.** Comparison of the bus access cycles of real-time benchmarks between the two cache architectural models, which are normalized to the separate-cache architecture.

of every real-time benchmark between these two models of cache architectures. As we can see, the number of bus access cycles of the partitioned cache is at least 50% larger than that of the separate cache. This is because whenever there is an L1 miss, either it is an L1 instruction cache miss or an L1 data cache miss, and access to the L2 cache or even the main memory is then necessary. In the separate-cache architecture, the L2 cache can be directly accessed through the private bus of each core without any inter-thread interference. By comparison, in a partitioned-cache architecture, since the centralized L2 cache structure is shared by different cores through the shared bus, the bus access latency becomes much longer due to both arbitration and waiting delay. It should be noted that in a separate L2 cache architecture, different L2 caches need to compete for the bus to access shared memory. However, this only happens for each L2 miss, which occurs much less frequently than an L1 miss.

Fig. 4 compares the total execution time of the real-time benchmarks between these two cache architectures. We observe that the execution time of the partitioned-L2-cache model is at least 9.8% longer than that of the separate-cache architecture. This is mainly because the separate-cache architecture has lower latencies in both bus accesses and L2 accesses than the partitioned-cache architecture, leading to overall better performance.

Fig. 5 shows the comparison of the total cache energy

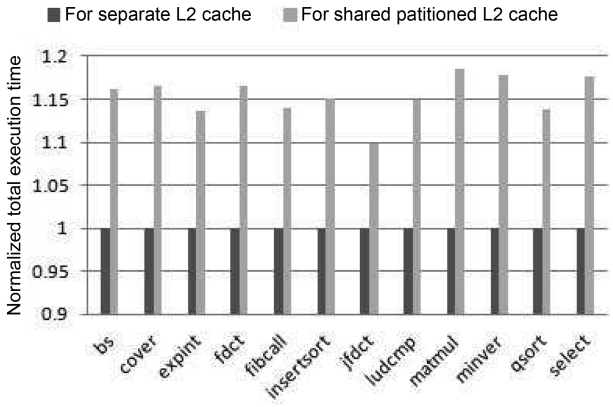



**Fig. 4.** Comparison of the total execution time of real-time benchmarks between the separate-cache architecture and the partitioned-cache architecture, which is normalized to that of the separate-cache architecture.

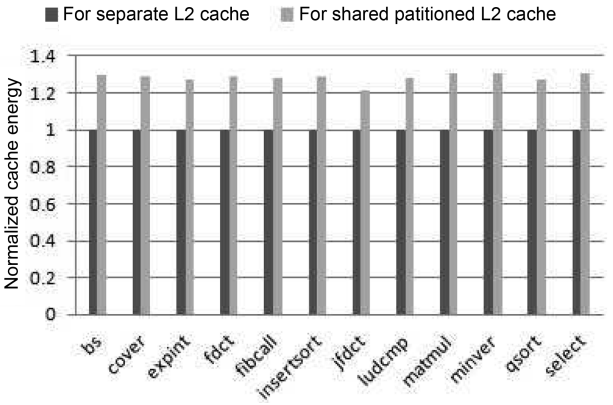



**Fig. 5.** Comparison of the total cache energy consumption of real-time benchmarks between the two architectural models, which is normalized to that of the separate-cache architecture.

consumption of during real-time benchmarks between these two cache architectures. As can be seen in this figure, the energy dissipation for the partitioned-L2-cache model is at least 21% larger than the separate one. This is because the shared partitioned cache consumes both more dynamic energy per access and more leakage energy due to the longer execution time.

To compare the performance of the two architectures' performance, we also calculated the throughput of the systems for the three benchmark groups by using Eq. (7), where it is the execution time of the benchmark running on the *i*th core of the system. In Fig. 6, we can see that the throughput of the shared partitioned-L2-cache model is 87% that of the separate one at most, which confirms the performance superiority of the separate-cache architecture.

$$\text{Throughput} = 1/\max(ti),\ i = 0,1,2,3. \quad (7)$$

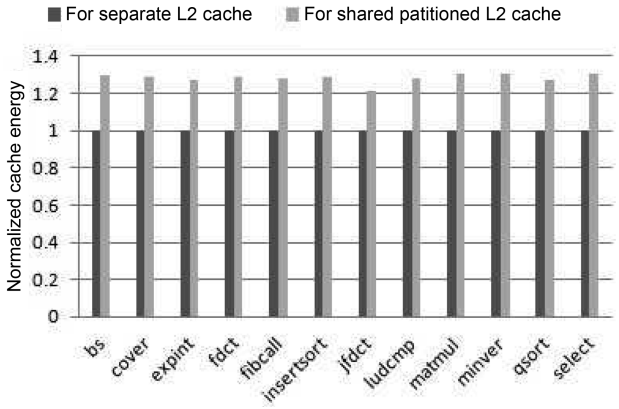



**Fig. 6.** Throughput comparison of 3 groups of real-time benchmarks between the separate-cache architecture and the partitioned-cache architecture, which is normalized to the throughput of the separate-cache architecture.

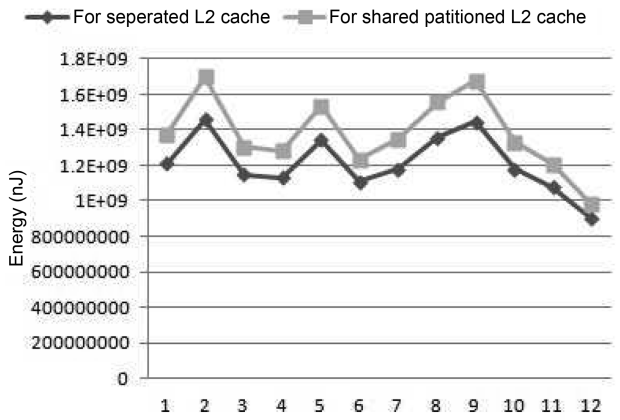



**Fig. 7.** Comparison of the cache energy dissipation between the separate-cache architecture and the statically partitioned-cache architecture for SPEC 2006 benchmarks.

In summary, from the above evaluation and analysis, we conclude that both the performance and energy efficiency of the separate-L2-cache architecture are much better than those of the statically partitioned-L2-cache architecture for the real-time benchmarks used in our simulation.

### *B. Performance and Energy Consumption Comparison of SPEC CPU2006 Benchmarks*

We have run 12 groups of SPEC CPU2006 benchmark combinations as shown in Table 6. Fig. 7 gives the total cache energy consumption of the four-core processor with both cache architectural models. As can be seen, the energy consumed by the partitioned-L2-cache architecture is higher than that of the separate-cache architecture for the SPEC 2006 benchmarks as well. Specifically, the cache energy consumed by the partitioned L2 cache is at least 9% larger than that of the separate cache.

In addition to the energy results, we also compared the throughput between these two cache architectures using the SPEC 2006 benchmarks, seen in Fig. 8. We observe that for those SPEC 2006 benchmarks, the throughput of the shared partitioned-cache architecture is about 93% that of the throughput of the separate-cache architecture. Therefore, the same conclusion can be made as a result of both real-time and SPEC 2006 benchmarks which is that

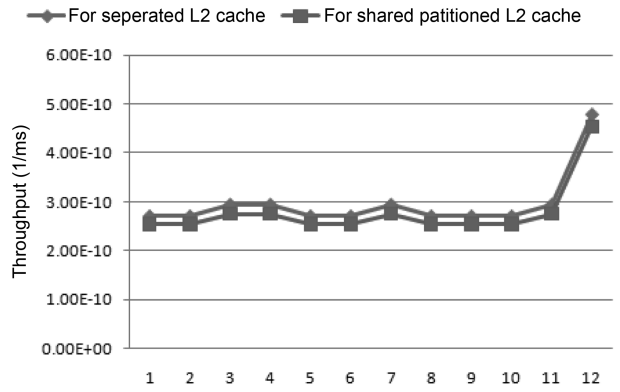



**Fig. 8.** Compare the throughput between the separate-cache architecture and the statically partitioned-cache architecture for SPEC 2006 benchmarks.

the separate-cache architecture is superior to the statically partitioned-cache architecture in terms of both performance and energy efficiency.

## VI. CONCLUSION

This paper quantitatively compares two types of time-predictable multicore cache architectures. One involves a

**Table 6.** SPEC CPU2006 benchmark test groups in this experiment

| Core | Group 1 | Group 2 | Group 3 | Group 4 | Group 5 | Group 6 | Group 7 | Group 8 | Group 9 | Group 10 | Group 11 | Group 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | bzip2 | mcf | hmmer | sjeng | lbm | sphi nx3 | bzip2 | hummer | lbm | sjeng | sphinx3 | bzip2 |
| 1 | mcf | hmmer | sjeng | lbm | sphinx3 | bzip2 | hummer | lbm | bzip2 | mcf | hummer | hummer |
| 2 | hmmer | sjeng | lbm | sphinx3 | bzip2 | mcf | sjeng | sphinx3 | mcf | hummer | lbm | sjeng |
| 3 | sjeng | lbm | sphinx3 | bzip2 | mcf | hmmer | lbm | mcf | sjeng | sphinx3 | bzip2 | sphinx3 |

multicore processor with separate L2 caches, and the other is that of a multicore processor with a shared but statically-partitioned L2 cache, both of which are suitable for worst-case timing analysis. This paper focuses on comparatively evaluating the performance and energy efficiency of both cache architectures, and our experimental results indicate that due to faster bus access and shorter L2 access latency, the separate-cache architecture is superior in both performance and energy consumption in real-time and SPEC 2006 benchmarks, and thus is preferable for a multicore-based hard real-time systems.

It should be noted that this work only compares separate L2 caches with a statically, evenly-partitioned L2 cache on a four-core processor. Generally, shared L2 cache can be partitioned in different ways, which may improve the efficiency of the cache space utilization when the partition matches the cache access behavior of the concurrent programs. However, due to the timing and energy efficiency per access to a smaller separate cache in conjunction with the scalability of the separate cache design, we believe the separate-cache architecture have significant advantages over statically-partitioned caches in achieving time predictability as well as good performance and energy efficiency during real-time multicore computing.

In future work, we would like to measure the execution time and energy dissipation on real, multicore processors with different cache architectures to further the comparative evaluation. Also, we would like to vary the number of cores to study the performance trends of these two cache architectures with respect to an increased number of cores.

## ACKNOWLEDGMENTS

This work was funded in part by the NSF grant CCF 0914543.

## REFERENCES

1. M. Paolieri, E. Quinones, F. J. Cazorla, G. Bernat, and M. Valero, "Hardware support for WCET analysis of hard real-time multicore systems," in *Proceedings of the 36th Annual International Symposium on Computer Architecture*, Austin, TX, 2009, pp. 57-68.
2. Malardalen WCET benchmarks, http://www.mrtc.mdh.se/projects/wcet/benchmarks.html.
3. R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, and P. Stenstrom, "The worst-case execution-time problem: overview of methods and survey of tools," *ACM Transactions on Embedded Computing Systems*, vol. 7, no. 3, article no. 36, 2008.
4. J. M. Calandrino, J. H. Anderson, and D. P. Baumberger, "A hybrid real-time scheduling approach for large-scale multicore platforms," in *Proceedings of the 19th Euromicro Conference on Real-Time Systems*, Pisa, Italy, 2007, pp. 247-258.
5. C. A. Healy, D. B. Whalley, and M. G. Harmon, "Integrating the timing analysis of pipelining and instruction caching," in *Proceedings of the 16th Real-Time Systems Symposium*, Pisa, Italy, 1995, pp. 288-297.
6. J. Stohr, A. von Bulow, and G. Farber, "Bounding worst-case access times in modern multiprocessor systems," in *Proceedings of the 17th Euromicro Conference on Real-Time Systems*, Palma de Mallorca, Spain, 2005, pp. 189-198.
7. J. Rosen, A. Andrei, P. Eles, and Z. Peng, "Bus access optimization for predictable implementation of real-time application on multiprocessor system-on-chip," in *Proceedings of the 28th IEEE International Real-Time Systems Symposium*, Tucson, AZ, 2007, pp. 49-60.
8. J. Yan and W. Zhang, "WCET analysis for multi-core processors with shared L2 instruction caches," in *Proceedings of the 14th IEEE Real-Time and Embedded Technology and Applications Symposium*, St. Louis, MO, 2008, pp. 80-89.
9. Y. Li, V. Suhendra, Y. Liang, T. Mitra, and A. Roychoudhury, "Timing analysis of concurrent programs running on shared cache multi-cores," in *Proceedings of the 30th IEEE Real-Time System Symposium*, Washington, DC, 2009, pp. 57-67.
10. S. Chattopadhyay, C. L. Kee, A. Roychoudhury, T. Kelter, P. Marwedel, and H. Falk, "A unified WCET analysis framework for multi-core platforms," in *Proceedings of the IEEE Real-Time and Embedded Technology and Applications Symposium*, Beijing, China, 2012, p. 99-108.
11. T. Kelter, H. Falk, P. Marwedel, S. Chattopadhyay, and A. Roychoudhury, "Bus-aware multicore WCET analysis through TDMA offset bounds," in *Proceedings of the 23rd Euromicro Conference on Real-Time Systems*, Porto, Portugal, 2011, pp. 3-12.
12. L. Wu and W. Zhang, "A model checking based approach to bounding worst-case execution time for multicore processors," *ACM Transactions on Embedded Computer Systems*, vol. 11, no. S2, article no. 56, 2012.
13. W. Zhang and J. Yan, "Accurately estimating worst-case execution time for multi-core processors with shared direct-mapped instruction caches," in *Proceedings of the 15th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, Beijing, China, 2009, pp. 455-463.
14. J. Reineke, D. Grund, C. Berg, and R. Wilhelm, "Timing predictability of cache replacement policies," *Real-Time Systems*, vol. 37, no. 2, pp. 99-122, 2007.
15. S. A. Edwards and E. A. Lee, "The case for the precision timed (PRET) machine," in *Proceedings of the 44th ACM/IEEE Design Automation Conference*, San Diego, CA, 2007, pp. 264-265.
16. A. Hansson, K. Goossens, M. Bekooij, and J. Huisken, "CoMPSoC: a template for composable and predictable multi-processor system on chips," *ACM Transactions on Design Automation of Electronic Systems*, vol. 14, no. 1, article no. 2, 2009.
17. D. Hardy and I. Puaut, "WCET analysis of multi-level non-inclusive set-associative instruction caches," in *Proceedings of the 29th Real-Time Systems Symposium*, Barcelona, Spain, 2008, pp. 456-466.
18. SimpleScalar, http://www.simplescalar.com/.

19. CACTI, http://www.hpl.hp.com/research/cacti/.

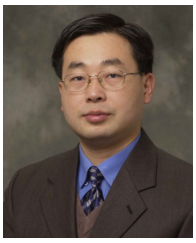
20. SPEC CPU2006, http://www.spec.org/cpu2006/.

## Lan Wu

Lan Wu received her B.S. of Computer Science in July 2004 from University of Science and Technology of China and her M.S. of Computer Engineering in July 2007 from North China Institute of Computing Technology. She is now a Ph.D. student of Computer Engineering at Virginia Commonwealth University. Her research interests focus on real-time and embedded systems, computer architecture and virtualization.

## Yiqiang Ding

Yiqiang Ding is currently a Ph.D. student in Electrical and Computer Engineering at Virginia Commonwealth University. He received the B.S. degree in Computer Science in 2002 and the M.S. degree of Computer Engineering in 2005 from the Beijing University of Posts and Telecommunications in China. He worked in the Motorola China Design Center as a systems engineer from 2005 to 2007. His research interests are in embedded and real-time computing systems, computer architectures, and compilers.



## Wei Zhang

Wei Zhang is an associate professor (tenured) in Electrical and Computer Engineering at Virginia Commonwealth University. Dr. Wei Zhang received his Ph.D. from Pennsylvania State University in 2003. From August 2003 to July 2010, Dr. Zhang worked as an assistant professor and then as an associate professor (tenured) at Southern Illinois University-Carbondale. His research interests are in embedded and real-time computing systems, computer architectures, compilers, and low-power systems. Dr. Zhang received the 2009 SIUC Excellence through Commitment Outstanding Scholar Award from the College of Engineering and the 2007 IBM Real-time Innovation Award. Dr. Zhang has received 5 research grants from the National Science Foundation. In addition, his research and educational efforts have been supported by industry including leading IT companies such as IBM, Intel, Motorola, and Altera. Dr. Zhang has published more than 100 papers in refereed journals and conference proceedings. He is a senior member of the IEEE, and an associate editor of the Journal of Computing Science and Engineering. He has served as a member of the organizing or program committees for several IEEE/ACM international conferences and workshops.