Contents lists available at SciVerse ScienceDirect

Journal of Systems Architecture

journal homepage: www.elsevier.com/locate/sysarc

An efficient and comprehensive scheduler on Asymmetric Multicore Architecture systems

Jiun-Hung Ding [a], Ya-Ting Chang [a], Zhou-dong Guo [a], Kuan-Ching Li [b], Yeh-Ching Chung [a,*]

[a] Dept. of Computer Science, National Tsing Hua University, Taiwan
[b] Dept. of Computer Science and Information Engineering, Providence University, Taiwan

ARTICLE INFO

Article history:
Available online 29 May 2013

Keywords:
Asymmetric architecture
NUMA architecture
Single-ISA heterogeneous multicore processors
Scheduling

ABSTRACT

Several studies have shown that Asymmetric Multicore Processors (AMPs) systems, which are composed of processors with different hardware characteristics, present better performance and power when compared to homogeneous systems. With Moore's law behavior still lasting, core-count growth creates typical non-uniform memory accesses (NUMA). Existing schedulers assume that the underlying architecture is homogeneous, and as consequence, they may not be well suited for AMP and NUMA systems, since they, respectively, do not properly explore hardware elements asymmetry, while improving memory utilization by avoid multi-processes data starvation. In this paper we propose a new scheduler, namely NUMA-aware Scheduler, to accommodate the next generation of AMP architectures in terms of architecture asymmetry and processes starvation. Experimental results show that the average speedup is 1.36 times faster than default Linux scheduler through evaluation using PARSEC benchmarks, demonstrating that the proposed technique is promising when compared to other prior studies.

1. Introduction

With advances in microprocessor technologies and accelerated with development of multicore, manycore- and embedded systems-related technologies last years, processors evolve to include more processing units – hundreds to thousands of cores – into one single die and widely exploited in High Performance Computing, by harnessing processor architectures in parallel with other technologies and techniques to achieve such high performance.

Asymmetric Multicore Processors (AMPs) system is recently introduced, as composed of processors with different characteristics, e.g., clock speed, cache capacities, power consumption, occupied area and the complexity of execution pipeline, containing or not the same Instruction Set Architecture (ISA), also known as single-ISA heterogeneous multicore [1,3]. Instances of AMP system may contain a few powerful and effective cores and a larger number of cores with slower speed and less power consumptions [1,3]. Many strategies are employed to explore few powerful out-of-order with higher clock speeds and large cache capacities, suitable for executing the throughput oriented applications and single-threaded sequential applications, while for slower but less power-consuming cores, for parallel execution. Such an idea has been considered by major manufacturers as IBM, AMD and Intel, to combine 32- or 64-bit $\times 86$ or Power cores with capable graphics processing units (GPUs) or Synergistic Processor Elements (SPEs) on a single silicon die, e.g., IBM's cell processor [21], AMD's APU [20] and Intel's Larrabee [19]. Prior studies show that the typical AMP system has significant energy benefits and occupies minor die area, yet maximize the power efficiency [1,2,8]. As result, given the core-count growth, access time to memory is variable and depends on the relative location of a processor, which characterizes it as Non-Uniform Memory Access architecture (NUMA) [17]. With rapid growth on the number of cores in computing systems, the amount of memory requests issued by processor cores increases memory starvation.

This limitation on the number of memory accesses decreases the performance of modern multicore systems, and can starve several processors at the same time. In NUMA systems, this problem is settled by providing separate memory for each processor, which is likely to lift the performance when several processors attempt to access same memory. Unfortunately, current OS schedulers assume that the underneath hardware is homogeneous, that is, AMP systems and NUMA architecture are not considered as well as decoupled. Taking as example Linux 2.6 Completely Fair Scheduler (CFS), this scheduler uses a red–black tree implementation to manage the executable processes instead of running queue per processor. The main idea of CFS is to provide processor time to each task fairly. For instance, in a system with n executable processes, each of them should be given $1/n$ process time of a tiny period.

* Corresponding author. Address: Dept. of Computer Science, National Tsing Hua University, No. 101, Section 2, Kuang-Fu Road, Hsinchu 30013, Taiwan. Tel.: +886 3 5742971.
E-mail address: ychung@cs.nthu.edu.tw (Y.-C. Chung).

1383-7621/$ - see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.sysarc.2013.05.006

Since the abilities of processors in AMP systems are different, $1/n$ process time in faster and slower processor cores are completely different. Hence, the scheduler should take the AMP architecture into account. In another direction, NUMA architecture can be used to avoid contention of memory accesses between processes, by dividing the memory into multiple nodes, exploring the high-speed interconnections among them, e.g., Intel's Quick Path Interconnect (QPI) and AMD's Hyper Transport (HT). However, given the higher core-count growths and consequent large NUMA architectures formed, combined to the different AMP hardware adaptive design, can provide smaller memory resource contention and avoid data starvation. Again, the scheduler must consider the NUMA architecture in order to get additional benefits from this computer memory design. Based on this tendency, we believe that the AMP and NUMA are essential as the next generation of hand-held devices' architecture. In order to make OS working well with AMP and NUMA, we propose a new scheduler policy, NUMA-aware Scheduler for Asymmetric Multicore Processors, to support AMP and NUMA architectures. Interesting components as target in our proposed scheduler policy are twofold. The former one is Asymmetric-aware schedule policy, where dynamically trigger AMP scheduler to place the suitable processes on the specific type of cores, while the latter one is NUMA-aware schedule policy, in which precisely calculates the current system performance degradation due to resource contention, minimizing the degradation by thread migration and memory management.

The proposed NUMA-aware Scheduler for AMP (Asymmetric Multicore Processors) is implemented in Linux CentOS release 6.0 and evaluated on a 8-core, 32 GB Dell PowerEdge R910 system. Using performance counters, we independently modulated the CPU frequency as a performance asymmetry factor and explored the NUMA memory space to avoid resource contention. Comparing to Linux CFS scheduler and execution of PARSEC benchmarks, the proposed scheduler improves performance by a factor of $1.36\times$.

The remaining of this paper is organized as follows. In Section 2, some related works are presented, while the overview of NUMA-aware Scheduler for Asymmetric Multicore Processors is given in Section 3. The design of the NUMA-aware Scheduler for Asymmetric Multicore Processors is discussed in Section 4, and evaluation is shown in Section 5. Finally, Section 6 summarizes our findings, as also brings some remarks and topics for future research.

2. Related work

There are several references in literature showing energy benefits of Asymmetric Multicore Architectures [1,2,8]. The research study in [1] showed that this architecture could achieve a large amount of energy reduction with small performance penalty. In order to accommodate the heterogeneity of Asymmetric Multicore Processors, there are several researches [1–9] that discussed scheduling algorithms. Some of them considered the load balancing policy and then implemented an Asymmetric-aware load-balancing [3]. Therefore, the processes' characteristics were not taken into account. Some of them made use of static-time profiling data [4,5], in which could not detect the phase change of a process during runtime. Krumar et al. [1,2] proposed a dynamic core selection based on actual execution performance between different types of cores, and therefore, threads migrate between different cores. Unfortunately, the thread migration overhead is redundant and the cost is high, especially on NUMA architectures. Some of them proposed a dynamic way [6] to implement the scheduler during runtime and profiling data simultaneously, though the periodic profiling and computing are time consuming. Furthermore, the NUMA architecture is not fully considered in such study.

The proposed scheduler based on dynamic computed metric is surprisingly accurate and processes did not have to execute on different type of cores. We made use of hardware counter to gather periodically system's information with tiny overhead and computed the corresponding metric when necessary. Therefore, we minimized the overhead as best as possible and scheduled the processes properly. Hence, modern multicore systems increasingly use the NUMA architecture, and it had been discussed for several years [10–15]. NUMA architectures have benefits, but the system could not learn to profit with proper utilization. Yang et al. [10] analyzed the on-chip interconnect and intra-core bandwidth contention, and then showed the importance of load-balancing between threads. Blagodurov et al. [11] presented a NUMA-aware contention management to reduce the performance degradation; Majo et al. [14] solve the problem by taking both interconnect overhead and cache contention into consideration. In addition, Pusukuri et al. [15] dynamically reduced the performance variation due to NUMA architectures.

3. Proposed scheduler

The purpose of a process scheduling is to optimally sort independent processes according to a given parameter and then execute them. In proposed NUMA-aware Scheduler for AMP, the schedule policy is based on ranking processes according to two metrics, Online AMP Speedup Factor and Resource Contention Degradation Factor, to determine how appropriate they are to be run on certain type of core, the faster core or the slower core, or domain. In the aim of deriving these two metrics for a process, we need a runtime profiler to get ready this information. The AMP Speedup Factor and Contention Degradation Factor are recalculated once the schedule function is invoked.

In order to avoid redundant calculation yet minimize the overhead, the schedule function has to be invoked properly. We implemented two ways to invoke such the schedule function. The former one occurs when the process voluntarily releases the faster core, we have to invoke the scheduler directly for averting from losing greater ability of faster core, while the latter one is a lazy way to trigger the schedule function, invoked when we predict that there will be suitable candidate to run on faster core.

3.1. Framework components

The proposed NUMA-aware AMP scheduler is composed of two components, a runtime profiler and a scheduler. We use OProfile [22] as our system-wide profiler, leveraging the hardware performance counters to profile and analyze the statistic information at low overhead. After a time interval, we dump the profiling data to the data dealer. Once the data dealer receives the profiling data, it will update the records. In case the current situation cause the schedule function invoked, it computes two dynamic metrics: Online AMP Speedup Factor and Contention Degradation Factor. In addition, two important linked lists are maintained, AMP-list and NUMA-list, according to the dynamic values of metrics. In this way, the design method makes the data dealer

Algorithm 1. PROFILER: online profiling mechanism

```
1 Create a new thread for receiving and dealing with the
  online profiling data
2     Repeat profiling until NUMA-aware P-AMP scheduler
  stop
3         Sleep for an OPROFILE_PERIOD amount of time
4             Dump the profiling report
5   End Repeat loop
```

execute the heavy computation when needed, so the overhead is minimized. It is shown in Algorithm 1 the procedure of profiler.

Algorithm 2. DATA DEALER: receiving and dealing with the online profiling data

```
Input: online profiling data
1  Repeat until online profiling mechanism stop
2     Receiving data and filtering them from online profiling dump
3     Update the LLC miss rate and LLC Eviction Rate
4     If loading of system is unbalanced or behavior of the processes changed or powerful core is idle
5        Computing the Online P-AMP Speedup Factor
6        Sorting the process P-AMP-list by Online P-AMP Speedup Factor
7        Computing the Contention Degradation Factor
8        Sorting the process NUMA-list by Contention Degradation Factor
9        Sending signal to trigger schedule
10    End If
11 End Repeat loop
```

Once the data dealer receives the profiling data, it will update the records. In case the current situation cause the scheduler be called, it would compute the two metrics: Online P-AMP Speedup Factor and Contention Degradation Factor. We would maintain and sort the two important list, P-AMP list and NUMA list, based on those two metrics. This design method makes the data dealer execute the heavy computation when needed, so the overhead could be minimized. The procedure of data dealer is shown in Algorithm 2.

Algorithm 3. SCHEDULE: NUMA-aware P-AMP schedule

```
Input: P-AMP-list and NUMA-list
1 Computing the number of powerful core candidate based on fairly load balanced policy
2 Retrieving suitable processes to be scheduled on powerful cores from P-AMP-list
3 If Retrieved processes $\neq$ current processes on powerful cores
3     Migrate the processes
4 End If
5 If current Resource Contention Degradation is too big
6    Scatter the processes with heavy Contention Degradation Factor in order to minimize Resource Contention Degradation
7    Migrate the processes and the its sticky pages
8 End If
```

Moreover, we must have a schedule procedure to complete our framework. This schedule dynamically decides the number of candidates to run on the fast core based on fairly load balanced policy and schedules the processes on account of AMP list. In order to minimize the Resource Contention Degradation, the schedule also takes the NUMA-list into account to divert the resource degradation, as depicted in Algorithm 3.

3.2. Execution flow

It is illustrated in Fig. 1 the execution flow of NUMA-aware Scheduler for Asymmetric Multicore Processors. Once this scheduler starts, it will begin to profile the data and making them processed as useful information. Moreover, the schedule would be triggered when certain event occurs, then processes migrate as also memory pages for considerations of better performance.

3.3. Design

The proposed NUMA-aware Scheduler for AMP has two important parts: Asymmetric-aware schedule policy and NUMA-Aware schedule policy. The former one assists with better utilization on the different abilities of cores, making the processes of different characteristics to be executed on distinct cores. The latter policy is used to compensate for the lack of NUMA management and minimize the resource contention as best as possible. We will explain how these two parts work in following subsections.

3.3.1. Asymmetric-aware schedule policy

A typical AMP system contains two types of processors. One is processor with a couple of powerful and effective cores, whilst the other type of processor contains a larger number of cores with slower speed. We named the core in the first type of processor as "faster core" and the other core as "slower core".

The faster core differs from slower core in several points, such as execution performance, complexity of execution, cache capacities, among other characteristics. The scheduler for homogeneous system executes the runnable process on the least loaded core, working on the premise that the abilities of the cores are the same. To better making use of these different cores, we need to be acquainted with them and understand most suitable type of threads should be run on given cores. Hence, a metric called Online AMP Speedup Factor is introduced, to understand the process' speedup gain on the faster core compared to slower core.

Online AMP Speedup Factor is used to specify what kind of thread could gain more profit form fast core for the purpose of improving the efficiency of AMP system. We classify the conditions systematically into two categories, that is, the ability variation of the heterogeneous cores and the thread-level parallelism.

Ability variation of the heterogeneous cores is one of the features in AMP system. As mentioned before, they differ in many points. We only focus on the two important and obvious diversities: the clock speed and pipeline complexity. For the fast core, it has high clock speed and complex instruction pipeline. Therefore, the thread run on fast core must sufficiently use processor's computation ability and has highly instruction-level parallelization in order to get the benefit from fast core. Unfortunately, not all of the thread utilizes the CPU in such an efficient way, and that, some threads have to stall for data fetched from main memory. Such threads have poor data locality and must frequently access the memory. Based on Fig. 1, we find out that each of the processes' asymmetric speedup from running on a fast core versus a slower core is highly varied because of the diversity of the process behavior.

To illustrate how the scheduler works in terms of core selection, we exemplify with Canneal (from Parsec suite), which is a memory-intensive program. Comparatively to the slower-clock CPU, for the higher-clock CPU, Canneal stalls additional clock cycles to get data ready. That is justified since the higher frequency CPU generates more clock cycles per second though to access the memory.

Fig. 3 shows the cycle spending ratio of higher CPU frequency to lower CPU frequency. Fig. 4 illustrates the behavior of memory-bound programs, since they present higher L3 cache miss ratios.

Such behavior is observed by analyzing Figs. 3 and 4, where is shown that the memory-intensive applications are not easy to get the benefit from faster cores, which can be used to accommodate CPU-intensive ones Therefore, we take this issue into consideration for our Online AMP Speedup Factor. Still from these two

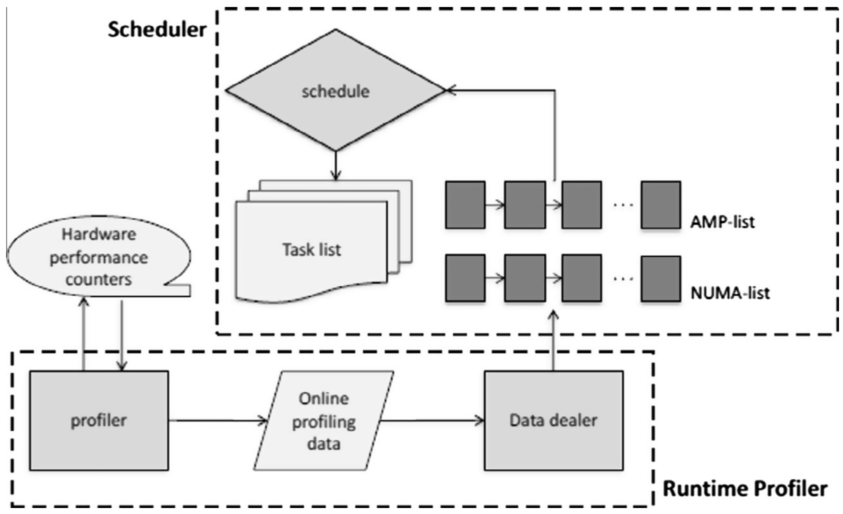


Fig. 1. Execution flow of NUMA-aware Scheduler for Asymmetric Multicore Processors.

figures, the Online AMP Speedup Factor shows its accuracy based on experimental results of the real asymmetric speedup, as depicted in Fig. 2.

Just as a reference, the default Linux scheduler has a function that implements the scheduler. Its purpose is to identify appropriate process and assign next to the CPU. It could be invoked directly or in a lazy way. In a direct way, the schedule function is called since the current process must be blocked immediately. Otherwise, it would set the certain flag to 1 and then check the flag to invoke the schedule function at some future close time in a lazy way. In our scheduler, we implement a lazy way schedule and construct some events that will trigger the scheduler to be invoked. In this way, the proposed scheduler would be more efficient with low overhead.

We have also implemented a direct way to invoke schedule. As for performance consideration, we have to make the fast core as busy as possible and not idle it if there is still a runnable process available. For this purpose, we propose an algorithm, schedule fast core first, that ensures the thread would run on fast core if it is under-utilized. If we find out the fast core is under-utilized and there is a runnable process run on slower core, we will invoke the schedule function directly to completely use the fast core's greater execution ability. This is our direct way to call schedule function.

We introduce a metric called Online AMP Speedup Factor (AMP_SF), to estimate the relatively real asymmetric speedup gain from running on a faster core versus a slower core based on the following considerations.

- Thread-level parallelism is an important issue that needs to be taken into consideration. If the number of threads in a process is higher than the number of faster cores, redundant threads would be executed on the slower cores' domain. This will induce to two situations: synchronization and communication overhead. Synchronization overhead happens for the reason that threads on slower cores would take more execution time to complete the execution, while those threads on faster cores must wait on their completion, in which may extend the execution time of that process. The other situation is the communication overhead. Since a process' threads would be placed in different domains, the communication between threads would cross domains. It is better to avoid this situation happen owing to the time consuming of inter-domain communication is larger than the intra-domain communication.
- Observing the situations above presented, we could find out that a process would not get any speedup gain from faster cores if its threads are on different domains. Experiments with three different numbers of threads were performed in PARSEC benchmarks, where half of threads were included in powerful domain and another half on slower one. As depicted in Fig. 5, we detected that the execution time of a process was almost the same as it executed all of its threads on slower domains or even worse one. Therefore, we believe that is better not to proceed with the execution of a process' threads on different type of domains.
- It is well known that the speedup of a program using multiple computing nodes concurrently is limited by the sequential fraction of that program, as stated in Amdahl's law. Additionally, there is limited number of faster cores in such a proposed AMP system [16]. In this way, we would run parallel processes on slower cores as best as we can. If we run processes on faster cores during the sequential phase, the performance would be better than execution on slower core, due to the great ability of the faster core without overhead as presented earlier.
- We then incorporate in the proposed scheduler the following metrics.
- CPU-memory intensiveness: if the number of threads in a process (THN_process) is less than or equal to the number of faster cores (N_fc), its characteristics will affect the speedup. Thus, we only need to consider whether it is a CPU- or memory-intensive process through the CPU-bound Factor (CF) computed by the last level cache miss rate (llcm) in the system. If the number of threads of a process is greater than the number of faster cores, we need to take the thread-level parallelism into account.
- Communication overhead: as for the communication overhead, due to the inter-domain communication is getting faster and faster nowadays, so CPU-intensive process does not significantly suffer.

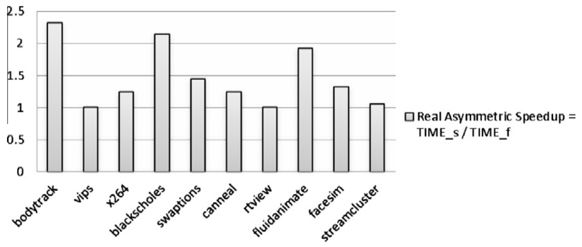


Fig. 2. The Real Asymmetric Speedup factors from running on a fast core (2 GHz) versus a slower core (1 GHz) are varied between different benchmarks from PARSEC benchmark suit.

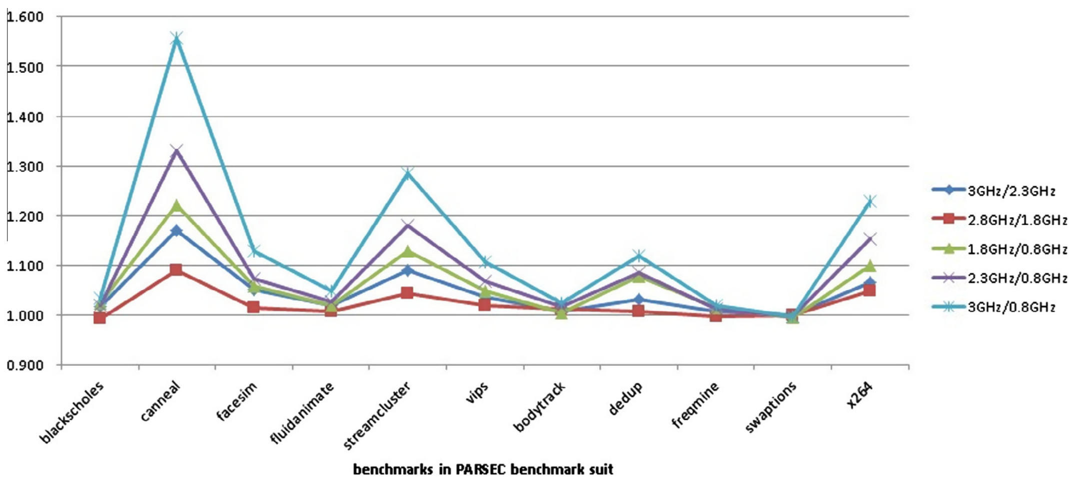


Fig. 3. The clock cycle spending ratio of higher CPU frequency to lower CPU frequency.

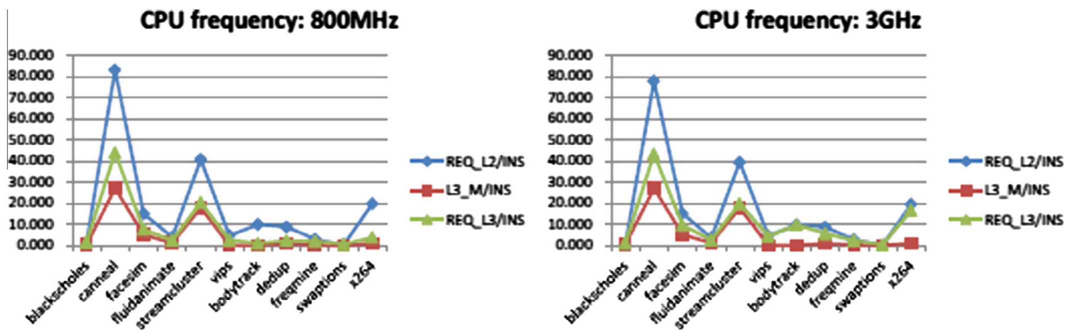


Fig. 4. The process with higher cache access frequency and L3 cache miss rate indicate that it is a memory-intensive process.

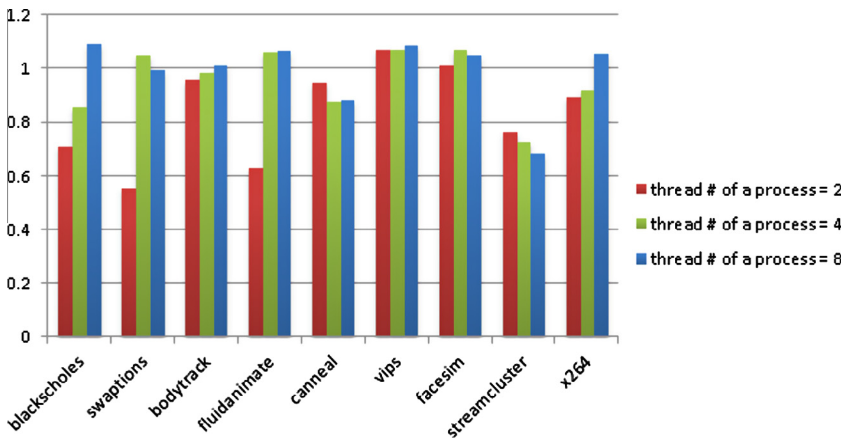


Fig. 5. Real asymmetric speedup of the process with half number of threads run on powerful domain and half run on slower domain.

- Synchronization overhead: in terms of synchronization overhead, it will sacrifice its total execution time and hold the lock. Consequently, its total execution time would be as slow as it did not run on faster core or even worse. The best to avoid these processes allocated to run on faster cores, and thus, for this class of situations we added a constant value 1, to make their AMP_SF be larger. In this way, they would be hard to be selected to fit in faster cores. The formula to deal with the synchronization overheads is presented in (1):

$$\mathrm{AMP\_SF} = \mathrm{CF} + (\mathrm{THN\_process} \leqslant \mathrm{N\_fc}?1:0) \tag{1}$$

The Event-triggered Scheduler invokes the schedule function in time with little overhead. As this function is invoked, it would balance the system loading and adjust the processes executed on faster cores. That is, the scheduler will do its work when the system loading is unbalanced and the suitable processes executed on fast cores have changed. Therefore, we would lazily trigger the scheduler when these classes of events happen. Unbalanced system loading would happen when the number of threads or the number of processes in the system changes. As we detect this event taking place, the scheduler flag will be set to invoke the schedule function.

Suitable processes executed on faster cores signify that top candidates from AMP list – constructed by Online AMP Speedup Factor – have changed. In other words, this event would happen when some process become the new member or is eliminated from top candidates of AMP list. By definition of Online AMP Speedup Factor, we found out that, as the last level cache miss rate (llcm) changed in a special way or number of threads of a process has changed, we need to reschedule. We make use of heuristic method to construct the Cache Miss Variation Ratio (CVR), to express what the special changing of last level cache miss rate is. The formula is presented in (2):

$$\mathrm{CVR} = \frac{\Delta \mathrm{llcm}}{\mu} \qquad (2)$$

where μ is the minimum between old last level cache miss rate (llcm) and current last level cache miss rate (llcm). Based on previous discussion, a model is proposed to decide whether to trigger the scheduler or not, as depicted in Fig. 6. It is illustrated in Fig. 7 an example on Cache Miss Variation Ratio (CVR).

3.3.2. NUMA-aware schedule policy

The NUMA architecture is increasingly being used by multicore systems, due to increase on the request for memory access and data starvation of processors. The processors starve for data easily due to its high execution performance as also increase on the amount of requests for memory access owing to the increasing number of processing units. Since the memories operate considerably slower than CPUs, the NUMA architecture solves the problem by providing more memory domains in multicore systems. Hence, running processes will not easily contend accesses to memory, which is exactly NUMA architecture's advantage. Though, if the operating system does not take NUMA into account, it might be unable to take advantage of the NUMA benefit, which is one of the NUMA architecture's drawbacks.

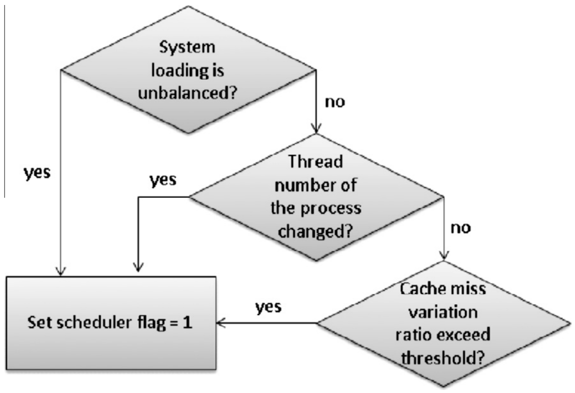

System loading is unbalanced?
no
yes
yes
Thread number of the process changed?
no
Set scheduler flag = 1
yes
Cache miss variation ratio exceed threshold?

Fig. 6. The flow chart of event-trigger scheduler.

The other problem is the remote memory access. Although the transfer rate of inter-communication is getting faster and faster, frequent remote access of the memory-intensive process would still drop its performance.

Alternatives identified to solve these two problems are discussed in following sections.

- NUMA-Aware Memory Migration

In terms of remote memory access, we move its sticky pages to local memory, in order to reduce the frequency of remote accesses. The sticky page is the memory page that the process frequently accesses, and it is determined as a sticky page by profiling its information. Further, threads do not migrate between domains. Therefore, the thread migration between domains will happen only when the performance degradation due to resource contention is high.

- Reduction of Degradation due to Resource Contention

In order to take advantages of NUMA, we need to prevent processors from addressing to same memory. Therefore, the performance of the processes' execution will not degrade due to resource contention. It is introduced a new metric to compute the performance degradation of each process, and found that heavy resource-needed processes in. Therefore, in case we put the heavy resource needed processes in a same domain, there would have big chances to compete for the memory resource, and their performance would decrease dramatically. Such a situation is exemplified in Fig. 8.

In order to solve this problem, the executions of heavy resource-needed processes are kept in the same memory domain. As a result, they will not contend each other, and therefore, the performance will not drop off. The metric, performance degradation of a process, is the sum of the degradation caused by all other processes in the same domain. For instance, there are five processes in a domain, A, B, C, D and E. We define the degradation of process A as Dg_A, the sum of degradation of A caused by each other process B, C, D and E, as shown in formula (3).

$$\mathrm{Dg\_A} = \mathrm{Dg\_AB} + \mathrm{Dg\_AC} + \mathrm{Dg\_AD} + \mathrm{Dg\_AE} \qquad (3)$$

LLC miss rate used to compute the Dg_AB and DgAC is not accurate enough since last level cache (LLC) miss rate is a heavy resource-needed process, and he process would access the memory resource when the LLC miss happened. Additionally, it may also occupy the hardware prefetcher, caches as also other memory resources with higher frequency, affecting LLC usage. As shown in Fig. 10 shows that process with the highest LLC miss rate does not necessary mean it is the most resource-needed one. Based on this result, additional information should be taken into account.

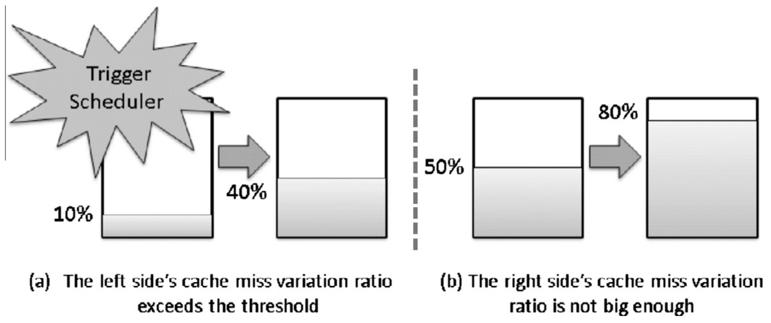

Trigger Scheduler
10%
40%
50%
80%
(a) The left side's cache miss variation ratio exceeds the threshold
(b) The right side's cache miss variation ratio is not big enough

Fig. 7. A simple example to explain when the cache miss variation ratio would exceed the threshold.

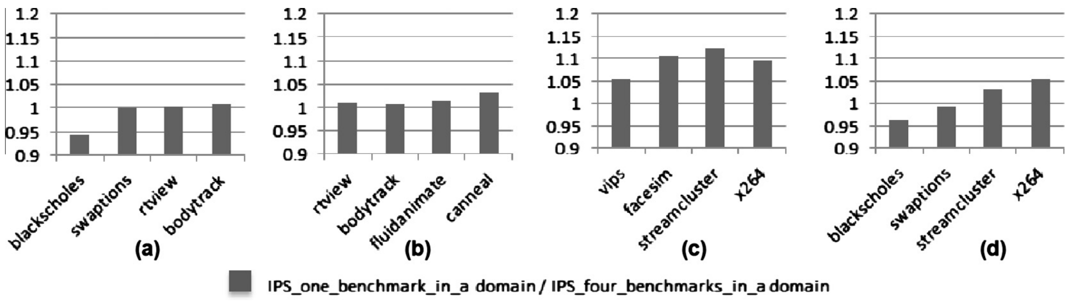


Fig. 8. Performance Degradation due to resource contention. Comparison of degradation in (d) with others, it is relatively large since other four benchmarks are memory-intensive.

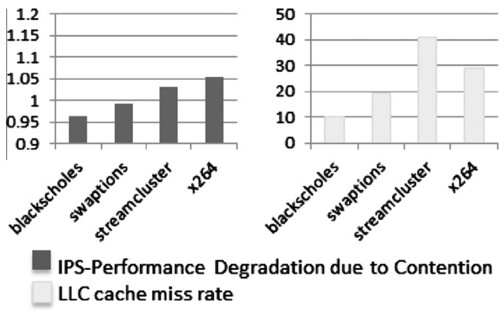


Fig. 9. The streamcluster benchmark program shows the highest LLC misses, though the performance has not most degraded.

For heavy resource-needed processes with higher precision, we take the Eviction Rate (ER) into consideration. To understand that, we suppose that there are three levels of cache in a system, this means that L3 is the last level cache. The process' ER is high if the data in the L3 cache are being expelled often, not accessing data often again, so the temporal locality of the process in L3 cache is poor. As a result, the degradation of process A caused by process B should consider the last level cache miss rate (llcm) as also the Eviction Rate as well. Additionally, the process with high ER would have smaller degradation because that the L3 cache is un-useful for it if others are compete with it for the L3 cache. That is, if the process A has high Eviction Rate, then its degradation has to be minimized. The formula of Dg_AB and ER is shown in (4) and (5), respectively:

$$\mathrm{Dg\_AB} = \mathrm{llcm\_A} * \mathrm{llcm\_B}/\mathrm{ER\_A} \quad (4)$$

$$\mathrm{ER} = \frac{\mathrm{N\_e}}{\mathrm{N\_fe}} \quad (5)$$

where N_e is the number of L3 cache line eviction and N_fe is the number of L3 cache line fills caused by L2 evictions. To simplify the computation complexity, we make of use the metric called Contention Degradation Factor (CDF) to define if the process is heavy resource-needed process or not. It is presented in (6):

$$\mathrm{CDF} = \frac{\mathrm{llcm}}{\mathrm{ER}} \quad (6)$$

4. Evaluation

To evaluate the accuracy of two metrics proposed as also the performance of NUMA-aware AMP Scheduler, the PARSEC benchmark suite is considered. The performance of asymmetric architectures was implemented on a server Dell PowerEdge R910 with CentOS Linux release 6.0 (Linux 2.6.32). The frequency of three CPUs was reduced by half, with settings conform to typical AMP system with 4 faster cores and 12 slower cores, emulating future generation of asymmetric architectures.

4.1. Accuracy of Online AMP Speedup Factor

The real asymmetric speedup is a ratio between the number of instructions per second executed on faster cores and the number of instructions per second executed on slower cores. That is, the higher the real asymmetric speedup of a process is, the more processes gain from the faster cores. Hence, we would rather to allocate such a kind of process on faster cores. We use the metric called Online AMP Speedup Factor to predict the real asymmetric speedup and the values are relative among each other. By observing the ability as also the number of threads in each process considered as presented before, the results showed are correspondingly accurate, as illustrated in Fig. 10.

4.2. Accuracy of Contention Degradation Factor

The metric, Contention Degradation Factor, is computed during runtime mining the profiling data, last level cache miss rates and

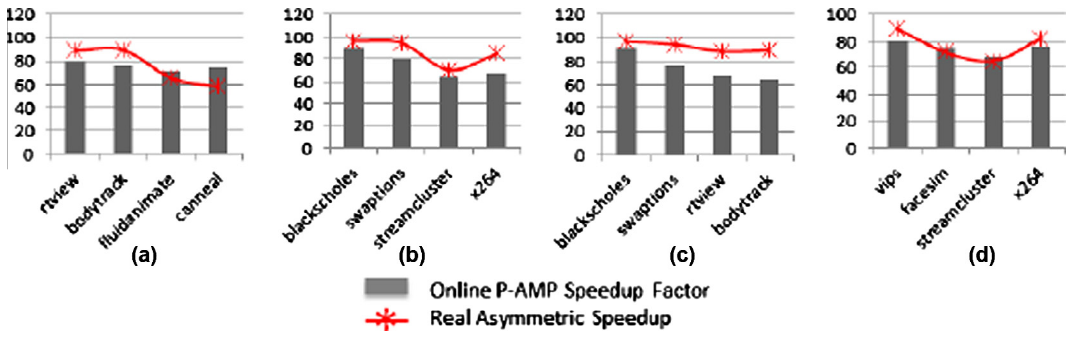


Fig. 10. Comparing the Online P-AMP Speedup Factor with Real Asymmetric Speedup. It is presented in charts (a–d) different combinations of benchmarks executed in the proposed system, and the Online AMP Speedup Factor was calculated using runtime profiling data.

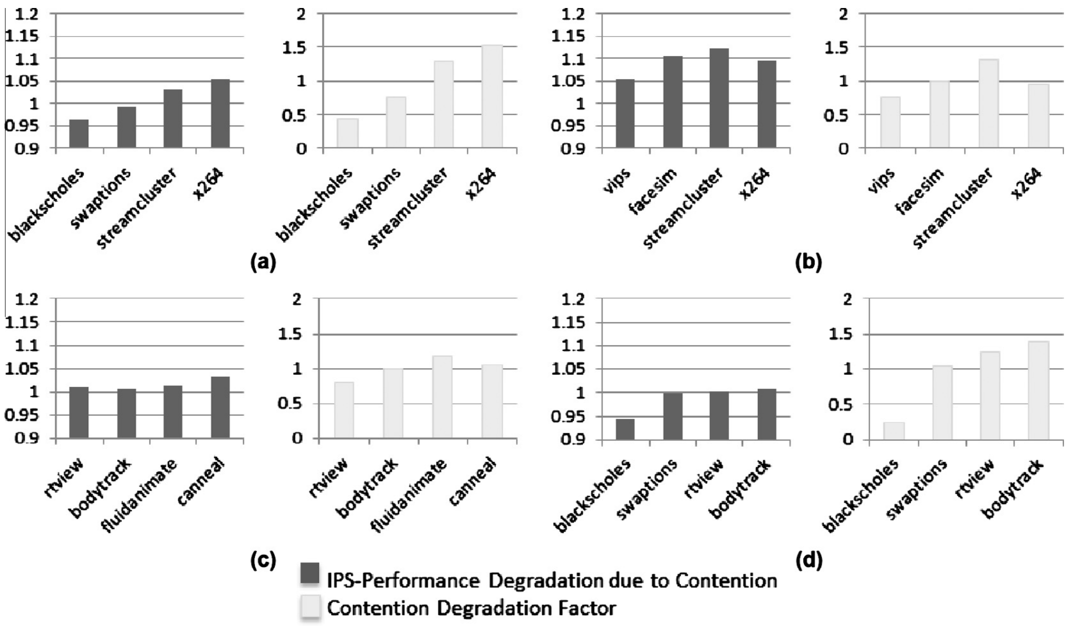


Fig. 11. The accuracy of the Contention Degradation Factor.

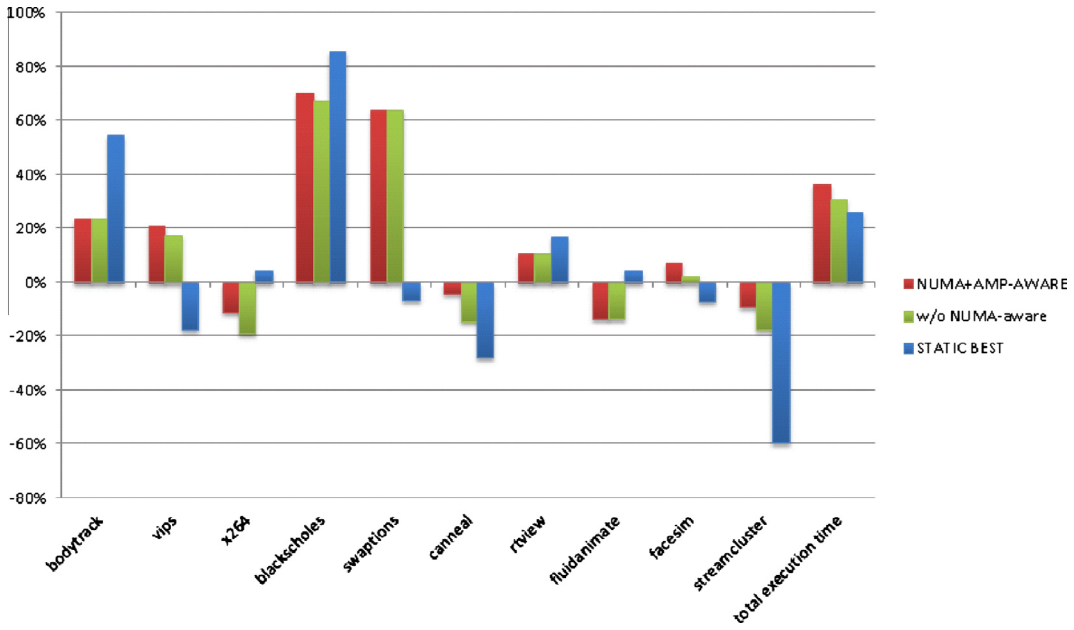


Fig. 12. The speedup of our NUMA + AMP AWARE, w/o NUMA-AWARE and the STATIC BEST on the 4 fast cores + 12 slower cores platform.

Eviction Rate. The processes' Contention Degradation Factors are relative among each other like the Online AMP Speedup Factor. Such a metric assist with precise identification of the heavy resource-needed processes, when the performance degrades dramatically. Experimental results show such a metric correspondingly accurate as well, as illustrated in Figs. 9 and 11.

4.3. Aggregate result

PARSEC [18] is a benchmark suite of programs focused on the emerging workloads and considered as the next-generation shared-memory programs for CMPs, composed of thirteen different multithreaded programs. We tested different combinations of programs from the PARSEC, in order to understand different average speedup gain from different combinations of workloads. From experiments, if the combination of workloads consisting of more CPU-intensive processes, higher the speedup. Therefore, we selected half CPU-intensive and half memory-intensive processes as the combination of our workload testing.

Experimental results show that, the appropriate assignment of processes to execute on faster cores may achieve higher speedup when compared to others, and just slightly degrading other processes' performance. Moreover, with the NUMA-aware contention reduction, the performance shown is better than original one.

The experimental result of Static Best was generated as follows. Before the process is submitted for executed, we statically decide whether it should run on faster cores or not by evaluating static-time profiling data, the Real Asymmetric Speedup. As a process exits and released the faster cores, we select a current suitable process by the static-time profiling data to migrate to faster cores. Given that phase change of a process cannot be detected through the static-time profiling data, therefore the results show worser performance than those results obtained from proposed method. As a process in the phase that is suitable to be executed on faster cores, the Static Best will not be able to detect and hence, it cannot obtain better speedup gain. Moreover, some benchmarks' performance were heavily decreased.

The average speedup of total execution time of PARSEC benchmarks is 1.36× times faster. As in Fig. 12, the experimental results showed that our scheduler could dynamically choose the proper processes and assign them to appropriate domains in times to get better utilization of AMP and NUMA architectures.

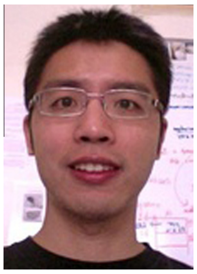
5. Summary and conclusions

An AMP system is a newly introduced computing system. In order to permit the operating system understand the underneath heterogeneous architecture, we proposed an AMP aware schedule policy. We introduced a new metric called Online AMP Speedup Factor to define which runnable processes should utilize the high efficiency of fast cores. The Online AMP Speedup Factor took the characteristic of a process and the thread-level parallelism into account, and the experimental results showed the metric was considerably accurate. Moreover, the process would degrade its performance if it could not get the wanted resource in time. In NUMA system, if we do not scatter the heavy resource-needed process, the performance would degrade dramatically and we could not get the same result in different runs. Hence, we proposed a new metric called Contention Degradation Factor to define that which is the heavy resource-needed process. We considered more factors than others compared to the prior studies. That's the reason why could predict the behavior precisely.

The average speedup of total execution time of PARSEC benchmarks is 1.36× times faster, and the result is quite good compared to the prior studies.

References

[1] Rakesh Kumar, Keith I. Farkas, Norman P. Jouppi, Parthasarathy Ranganathan, Dean M. Tullsen, Single-ISA Heterogeneous multi-core architectures: the potential for processor power reduction, in: Proceedings of the 36th International Symposium on Microarchitecture, San Diego, USA, December 03–05, 2003.
[2] Rakesh Kumar, Keith I. Farkas, Norman P. Jouppi, Parthasarathy Ranganathan, Dean M. Tullsen, Single-ISA heterogeneous multi-core architectures for multithreaded workload performance, in: Proceedings of the 31st Annual International Symposium on Computer Architecture (ISCA'04), München, Germany, June 19–23, 2004.
[3] Tong Li, Dan Baumberger, David A. Koufaty, Scott Hahn, Efficient operating system scheduling for performance-asymmetric multi-core architectures, in: Proceedings of the 2007 ACM/IEEE Conference on Supercomputing (SC '07), Reno, USA, 2007.
[4] Alexandra Fedorova, Juan Carlos Saez, Daniel Shelepov, Manuel Prieto, Maximizing power efficiency with asymmetric multicore systems, Communications of the ACM 52 (12) (2009).
[5] Daniel Shelepov, Juan Carlos Saez, Stacey Jeffery, Alexandra Fedorova, Nestor Perez, Zhi Feng Huang, Sergey Blagodurov, Viren Kumar, HASS: a scheduler for heterogeneous multicore systems, Operating Systems Review 43 (2) (2009) 66–75.
[6] Juan Carlos Saez, Manuel Prieto, Alexandra Fedorova, Sergey Blagodurov, A comprehensive scheduler for asymmetric multicore processors, in: Proceedings of the 5th ACM European Conference on Computer Systems (EuroSys 2010), Paris, France, April 13–16, 2010.
[7] Felipe L. Madruga, Henrique C. Freitas, Philippe O.A. Navaux, Parallel shared-memory workloads performance on asymmetric multi-core architectures, in: 18th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP), 2010.
[8] Vishal Gupta, Ripal Nathuji, Analyzing performance asymmetric multicore processors for latency sensitive datacenter applications, in: Proceedings of the 2010 International Conference on Power Aware Computing and Systems (HotPower'10), Vancouver, BC, Canada, 2010.
[9] Lina Sawalha, Sonya Wolff, Monte P. Tull, Ronald D. Barnes, Phase-guided scheduling on single-ISA heterogeneous multicore processors, in: Proceedings of the 14th Euromicro Conference on Digital System Design (DSD'11), 2011.
[10] R. Yang, J. Antony, A.P. Rendell, A simple performance model for multithreaded applications executing on non-uniform memory access computers, in: Proceedings of the 2009 11th IEEE International Conference on High Performance Computing and Communications (HPCC'09), Seoul, Korea, June 25–27, 2009.
[11] Sergey Blagodurov, Sergey Zhuravlev, Alexandra Fedorova, Ali Kamali, A case for NUMA-aware contention management on multicore systems, in: Proceedings of the 19th International Conference on Parallel Architectures and Compilation Techniques (PACT '10), Austria, September 11–15, 2010.
[12] Jeffery A. Brown, Leo Porter, Dean M. Tullsen, Fast thread migration via cache working set prediction, IEEE 17th International Symposium in High Performance Computer Architecture (HPCA), Texas, USA, February 12–16, 2011.
[13] Kishore Kumar Pusukuri, David Vengerov, Alexandra Fedorova Simon, Vana Kalogeraki, FACT: a framework for adaptive contention-aware thread migrations, in: Proceedings of the 8th ACM. International Conference on Computing Frontiers (CF'11), Ischia, Italy, May 3–5, 2011.
[14] Zoltan Majo, Thomas R. Gross, Memory management in NUMA multicore systems: trapped between cache contention and interconnect overhead, in: Proceedings of the International Symposium on Memory Management (ISMM'11), San Jose, USA, June 4–5, 2011.
[15] Kishore Kumar Pusukuri, Rajiv Gupta, Laxmi N. Bhuyan, Thread tranquilizer: dynamically reducing performance variation, Journal of ACM Transactions on Architecture and Code Optimization (TACO) 8 (4) (2012).
[16] Amdahl's Law. Available from: <http://en.wikipedia.org/wiki/Amdahl's_law> (accessed 19.11.12).
[17] Non-Uniform Memory Access. Available from: <http://en.wikipedia.org/wiki/Non-Uniform_Memory_Access>.
[18] PARSEC benchmarks. Available from: <http://parsec.cs.princeton.edu/> (accessed 5.12.12).
[19] Available from: <http://software.intel.com/en-us/articles/larrabee> (accessed 9.12.12).
[20] Available from: <http://www.amd.com/us/products/technologies/apu/Pages/apu.aspx> (accessed 9.12.12).
[21] Available from: <http://researcher.watson.ibm.com/researcher/view_project.php?id=2649>.
[22] OProfile. Available from: <http://oprofile.sourceforge.net/> (accessed 5.12.12).



Jiun-Hung Ding received a BS in Industrial Engineering and Management from National Chiao Tung University in 2004, and the MS in Computer Science from National Tsing Hua University in 2006. His research interest includes Embedded System, Hardware–Software Codesign, Multi-core Optimization, and Parallel Computing.

Ya-Ting Chang received a BS in Mathematics from National Cheng Kung University in 2010, and the MS in Computer Science from National Tsing Hua University in 2012. Her research interest includes parallel processing and multi-core embedded systems.

Zhou-dong Guo received a BA in Computer Science and Technology from Zhejiang University in 2011, and now studying for the MS int National Tsing Hua University. As a student of Professor Chung, he is doing research in the area of system software and embeded system.

Kuan-Ching Li is currently a Professor in the Department of Computer Science and Information Engineering at the Providence University, Taiwan. He received the PhD and MS in Electrical Engineering and Licenciatura in Mathematics from University of Sao Paulo, Brazil. He was a chair in 2009 and the Special Associate to the University President since 2010. He has served in a number of journal editorial boards and guest editorship, as also served many international conference chairmanship positions as steering committee, advisory committee, general and program committee chairs and member of program committees. His research interests include networked computing, parallel software design, and performance evaluation and benchmarking. He is a senior member of the IEEE and a Fellow of the IET.

Yeh-Ching Chung received a BS in Information Engineering from Chung Yuan Christian University in 1983, and the MS and PhD in Computer and Information Science from Syracuse University in 1988 and 1992, respectively. He joined the Department of Information Engineering at Feng Chia University as an Associate Professor in 1992 and became a Full Professor in 1999. From 1998 to 2001, he was the Chairman of the Department. In 2002, he joined the Department of Computer Science at National Tsing Hua University as a Full Professor. His research interests include parallel and distributed processing, cluster systems, grid computing, multi-core tool chain design, and multi-core embedded systems. He is a Member of the IEEE computer society and ACM.