

Power Management for Real-Time Embedded Systems on Block-Partitioned Multicore Platforms*

Xuan Qi and Dakai Zhu
Department of Computer Science
University of Texas at San Antonio
San Antonio, TX, 78249
{xqi, dzhu}@cs.utsa.edu

Abstract

Power management has become a very important research area and various approaches have been proposed. As an energy-efficient architecture, chip multiprocessor (CMP) has been widely adopted by chip manufacturers. In this paper, we study power management schemes for real-time systems on block-partitioned multicore platforms, where the processing cores are grouped into different blocks and cores on one block share the same power supply voltage (thus have the same frequency). We evaluate the energy efficiency of different block configurations. Simulation results show that block-partitioned CMP has its inherent advantages to fulfill the goal of power efficiency and low design complexity for future CMPs.

1. Introduction

Power efficiency has become a crucial issue in current computing systems. For the mobile and portable devices, battery is one dominant constraint with fixed energy budget. For the high-performance servers, the ever-increasing power consumption brings in not only the tremendous difficulty and high cost of building and operating the cooling system, but also the reliability concerns.

Power management for uniprocessor or multiprocessor real-time systems has been well explored and many techniques have been proposed [3, 5, 11, 14, 16, 19, 20]. Based on the *dynamic voltage/frequency scaling (DVFS)* technique, most power management schemes for multiprocessor real-time systems try to maximize energy saving by balancing the workload among processors and slowing down each processor individually [3, 20]. This is possible as those approaches assume that each processor can run at individual frequency/voltage to optimize energy savings. Traditional multiprocessor systems give good support for it since in general each processor is

plugged into a corresponding socket on the motherboard and has an individual voltage supply.

As an energy efficient architecture for higher performance, *chip-multiprocessor (CMP)* has been adopted by chip manufacturers as the main technology for future processors. Therefore, it is imperative to develop power management schemes for systems on multicore platforms. However, applying the power management schemes for multiprocessor systems directly to CMP might bring in some potential problems due to the inherent differences between these two architectures. For CMPs, even though it is possible to provide each core an individual voltage supply [6, 10, 15, 13], it could lead to very complex chip design, especially when a chip may contain tens (or even hundreds) of processing cores [18].

Following the same line of research as in [8], in this work, we study *block-partitioned* CMP configurations, where more than one cores are grouped together to form one block. The cores on one block will share the voltage supply (thus have the same frequency) to reduce the design complexity. We consider both *symmetric* (where the number of cores on each block is the same) and *asymmetric* (where the number of cores on block is different) block configurations. We study both static and dynamic power management schemes for block-partitioned CMPs. For different block configurations, we evaluate their power efficiency through simulations and discuss the design complexity tradeoffs. To the best of our knowledge, this is the first work on studying the energy efficiency and design complexity for real-time and embedded systems on block-partitioned CMP platforms.

The remainder of this paper is organized as follows. Section 2 introduces the concept of block-partitioned CMPs and presents the problem. Section 3 discusses static power management (SPM) for block-partitioned CMPs. Section 4 addresses dynamic power management (DPM) schemes. Section 5 presents the simulation results and discusses the power efficiency of different block configurations. Section 6 concludes the paper.

*This work was supported in part by NSF awards CNS-0720651 and Faculty Research Award of The University of Texas at San Antonio.

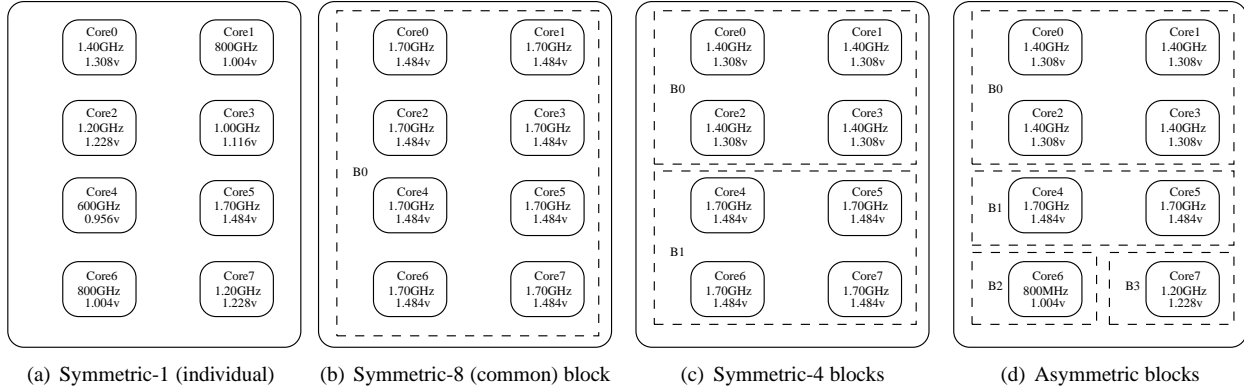


Figure 1. Different block configurations for a CMP with 8 cores

2. Block-Partitioned CMPs (BP-CMP)

Thanks to the fast-increasing clock rate, the performance of modern computers has increased steadily in the last few decades. However, the price for such performance increase is tremendously increased power consumption. To address such problems, instead of focusing on complex design techniques (e.g. superscalar and deeper pipelining) that exploit instruction level parallelism (ILP), alternative architectures have been studied taking advantages of thread level parallelism (TLP) of modern applications. For example, to provide the same level of performance, instead of having a superfast but complex processor, we can have several simple cores running in parallel at a comparatively low frequency on a single chip (CMP), which is also energy efficient.

Considering the nice features on high performance and energy efficiency, CMP has been adopted by the major chip manufacturers for the next generation processors (e.g., Intel Core2 Quad processor [1] and AMD Phenom Quad-Core processor [9]). On these multicore processors, to support flexible power management, each core has individual voltage supply [1, 9]. However, as the number of cores on a chip becomes larger (tens [18] or even hundreds), it could be very complex and expensive to provide individual power supply for each core.

2.1. Block Partitions

In our work, we consider a CMP with 2^k processing cores, where $k \geq 1$. The cores are assumed to be homogeneous, i.e., they have identical functions with each other. Considering the fact that it is possible to provide different supply voltages for different regions on a chip using the voltage island technique [6, 10, 15], the cores on one chip will be partitioned into *blocks*. For the cores that are on the same block, they will share the same power supply voltage and thus have the same frequency.

With clock gating, a core can be easily put into sleep when it has no workload for execution. Moreover, when

all cores on one block become idle, the block can be switched/powered off for more energy savings. With the ability of managing the power consumption at two different levels (i.e., cores and blocks), this BP-CMP structure could significantly reduce the design complexity/cost while providing certain flexibilities to adjust voltage and frequency for energy savings.

In what follows, depending on the different numbers of cores on each block, we focus on two categories of block-partition configurations: *symmetric* and *asymmetric* configurations.

2.2. Symmetric Configurations

For symmetric block partitioning, *the number of cores on each block will be the same*. However, for a CMP with given number of cores, depending on the number of cores on one block, we can have different *symmetric* block configurations.

As one example, suppose that we have one CMP with 8 cores. If each core form one individually block and has a separate power supply as shown in Figure 1(a), each core may adjust its supply voltage and frequency independently based on its workload. However, if there is only one power supply and all cores belong to one common block as shown in Figure 1(b), the supply voltage of the block will be determined by the most loaded core and all cores are assumed to have the same high frequency¹. Here, the dotted rectangles represent the blocks.

Here, the individual block configuration provides the most flexible power management, but it will incur the highest design complexity/cost. In comparison, the common block configuration only needs one power supply with reduced design complexity/cost, but with limited power management abilities on the cores. For a better

¹Although it is possible to have the cores run at different lower processing frequencies with the same high supply voltage, the addition circuits needed will make the design more complex and we will not consider this aspect in this paper.

tradeoff between design complexity/cost and power management flexibility, we may have either two cores per block (denoted as *symmetric-2*), or four cores per block (denoted as *symmetric-4*; which shown in Figure 1(c)). Note that, for symmetric block configurations, the number of cores on each block is normally the power of two.

2.3. Asymmetric Buddy Configuration

Although having the same number of cores on each block could lead to simplified design, the symmetric block configurations may not be the most energy efficient for different workload requirements. In this section, to provide flexible support for various workloads, we study asymmetric block configurations where the number of cores on the blocks may be different. In particular, borrowing the idea from buddy memory allocation, we consider the asymmetric *buddy* configuration.

For the above example, the asymmetric buddy configuration is shown in Figure 1(d), where there are four blocks and the number of cores on each of them is 4, 2, 1, and 1, respectively. That is, in the asymmetric buddy configuration, the first block contains half of all the cores on the chip, and the second block contains half of all the remaining cores, and so on. The last two blocks will contain one core each. In general, for a CMP with 2^k ($k \geq 1$) processing cores, there will be $(k+1)$ blocks in the asymmetric buddy configuration and the number of cores on the blocks will be $2^{(k-1)}$, $2^{(k-2)}$, ..., and 2^0 , respectively.

Note that, with the asymmetric buddy configuration, by appropriately selecting the blocks to be powered on, it is possible to run exactly *any* p ($1 \leq p \leq 2^k$) number of cores required by various workloads. For instance, if a certain workload needs 5 processing cores to obtain the maximum energy savings, we can use the blocks B_0 and B_2 in Figure 1(d). However, for the case of symmetric-4 configuration shown in Figure 1(c), we have to use both blocks to satisfy the performance requirements, which could be energy inefficient. It is also possible to run exactly 5 cores by exploiting the symmetric-1 (i.e., individual block) configuration as shown in Figure 1(a). However, symmetric-1 configuration has 8 blocks and will require 8 different power supply, compared to the asymmetric buddy configuration that only needs 4 power supply.

For the general case of a CMP with 2^k processing cores, compared to the 2^k power supply for the symmetric-1 configuration, as mentioned early, the asymmetric buddy configuration has $(k+1)$ blocks and thus only needs $(k+1)$ power supply, which can significantly reduce the design complexity/cost while providing flexible power management ability for various workloads.

2.4. Power Consumption for BP-CMPs

To effectively evaluate the energy efficiency of different configurations, in this section, we first study how

to model the power consumption for block-partitioned CMPs. Note that, instead of focusing on power management for individual components, system-wide power management has caught researchers' attention recently [2, 12, 17]. A simple but powerful system-wide power model has been studied for uniprocessor systems, where the power consumption of a computer system running at frequency f is modeled as [21]:

$$P(f) = P_s + P_{ind} + P_d = P_s + P_{ind} + C_{ef} * f^m \quad (1)$$

Despite its simplicity, this model includes all essential power components of a system. Here, P_s is the power used to maintain the basic circuit of the system (e.g., keeping the clock running), which can be removed only by powering off the whole system. Whenever the system is executing some workload, the active power, which has two parts P_{ind} and P_d , will be consumed. P_{ind} does not depend on the supply voltage and frequency, and can be effectively removed by putting the power manageable components of the system into sleep when the system is idle. P_d is the active power that depends on the supply voltage and processing frequency [4].

Following the similar idea and extending the above power model, we develop the power model for BP-CMP based computing systems. As the first step, we define a few terms. The total number of cores on the CMP considered is denoted as n_c and the number of blocks is defined as n_b . It is assumed that there are n_{c_i} processing cores on the i 's block ($1 \leq i \leq n_b$), which is denoted as B_i . The j 'th processing core on block B_i is denoted as $c_{i,j}$, which is suppose to run at frequency $f_{i,j}$. The maximum processing frequency for the cores is f_{max} .

Considering the fact that processing cores can be easily put into power saving state with the clock gating technique [1], it is assumed that each core has two states: *active* and *sleep*. A core is in active state if it is actively executing some workload; otherwise, it is idle and can be efficiently (in couple of cycles) put into sleep state. Therefore, one dynamic power component P_d will be associated with each core. In addition, when all the cores on one block are in sleep state, we may switch off the power supply for the block to obtain more power/energy savings. Therefore, each block has two states: *on* and *off*. Considering that the leakage power may depend on the number of circuits, each block will have one voltage/frequency independent active power component P_{ind} . Thus, for one BP-CMP based computing system, its power consumption P can be modeled as:

$$P = P_s + \sum_{i=1}^{n_b} \left(x_i \cdot P_{ind}^i + \sum_{j=1}^{n_{c_i}} y_{i,j} \cdot P_d^{i,j} \right) \quad (2)$$

The same as in Equation 1, P_s denotes the static power from all the power consuming components, which is *constant* and can only be removed by powering off the whole

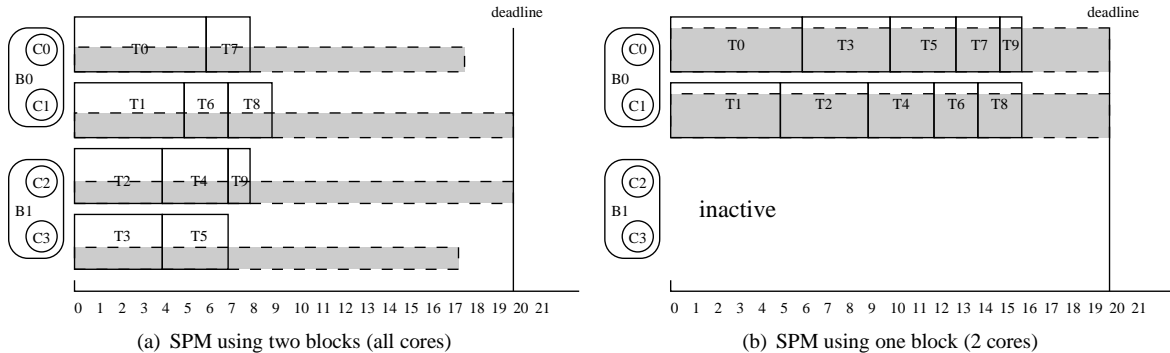


Figure 2. SPM for a CMP with four cores and the symmetric-2 partition.

system. x_i represents the state of the block B_i . If any core on the block is active and the block is on, $x_i = 1$; otherwise, $x_i = 0$ and B_i is switched off. Similarly, $y_{i,j}$ represents the state of the j 'th core on block B_i . As before, the frequency dependent active power for the core is defined as $P_d^{i,j} = C_{ef} \cdot f_i^m$, where both C_{ef} and m are system dependent parameters. Note that all cores on block B_i run at the same frequency f_i .

2.5. Problem Description

In this work, we consider an real-time application consisting of a set of independent tasks $\Gamma = \{T_1, T_2, \dots, T_n\}$ with a period D running on multicore platforms. Here, the worst case execution time (WCET) of task T_i is assumed to be known and denoted as wc_i . Tasks arrive at the beginning of each period and must finish before the end of the period (which is also denoted as frame-based applications). Considering both static power management (SPM) and dynamic power management (DPM) schemes, we focus on exploring different block-partitioned configurations for the multicore platforms. Such results will provide a guideline for studying the tradeoff between design complexity and power efficiency for multicore platforms.

3. SPM for BP-CMPs

In this section, we first study the SPM schemes for block-partitioned multicore platforms. The first and most important step in the scheduling for parallel systems is to map tasks to the active processing units. Different strategies for assigning tasks' priorities will lead to different task mappings, which result in different schedule lengths. For efficient power management in parallel systems, previous work shows that the maximum energy savings can be obtained if the workload can be perfectly balanced among the processing units [3, 5]. However, it has been shown that finding the optimal priority assignment for tasks to balance the workload is NP-Hard [3]. In this

work, we adopt the *longest task first (LTF)* heuristics as the priority assignment policy [20].

As one example, suppose that one real-time application consists of 10 independent tasks $\Gamma = \{T_0(6), T_1(5), T_2(4), T_3(4), T_4(3), T_5(3), T_6(2), T_7(2), T_8(2), T_9(1)\}$, where the value associated with each task is its WCET. With LTF priority assignment heuristics (where the tie is broken by giving higher priority to the task with smaller task id), the tasks in the task set Γ are ordered by their priorities with task T_0 having the highest priority and task T_9 having the lowest priority.

Assuming that each task uses its WCET, when the application runs on one multicore platform with four cores at the maximum frequency f_{max} , the schedule is shown in Figure 2(a). In the figure, X-axis represents time and Y-axis represents the processing speed/frequency (e.g., cycles per second) for the tasks. The area of the task box represents the workload of the task. Here, we can see that both the first and third cores (C_0 and C_2) complete their tasks at time 8, the second core (C_1) at time 9 and the fourth core (C_3) at time 7.

Suppose that the deadline of the application is $D = 20$, static slack exists on all the cores, which can be used to slow down the execution of the tasks for energy savings. If each core could be managed individually, the scaled frequencies for the four cores would be $0.4f_{max}$, $0.45f_{max}$, $0.4f_{max}$ and $0.35f_{max}$, respectively. However, as shown in the figure, the four cores are partitioned into two blocks where the first two cores form block B_0 and the last two cores form block B_1 . Recall that, the cores on the same block share the same supply voltage and frequency. To meet the timing constraints, the common frequency for the cores on the same block should be determined by the core with the most workload on that block. Therefore, the first two cores on block B_0 will run at frequency of $0.45f_{max}$ while other two cores on block B_1 run at frequency $0.4f_{max}$ as shown (the dotted gray boxes) in Figure 2(a).

However, considering static and frequency independent active power on each block (see Equation 2), it may

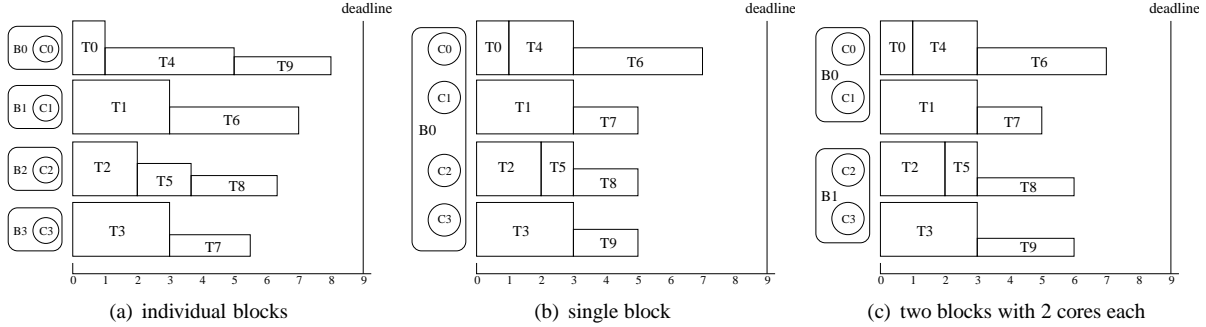


Figure 3. GSSR for different BP-CMP configurations

not always be energy efficient to use all the blocks. For the above example, if only two cores on the first block B_0 are used while the second block B_1 is switched off, Figure 2(b) shows that all tasks complete at time 16 before the deadline and the two cores on block B_0 can be slowed down to the frequency of $0.8f_{max}$.

For illustration purpose, suppose that $P_s = 0.1$, $C_{ef} = 1$ and $m = 3$. If the frequency independent active power is relatively small, for example, $P_{ind} = 0.1$ for each block, simple calculation shows that the energy consumption within one period of the application for the first case of using all the blocks (Figure 2(a)) is 16.20, which is much better than the second case of using only one block (Figure 2(b)) with the energy consumption of 26.48. However, when the frequency independent active power on each block dominates (for example, $P_{ind} = 1.0$), the energy consumption for the second case of using one block would be 42.48, which is less than that of the first case as 48.21. Therefore, depending on the relative value of P_{ind} , the optimal number of blocks used for a certain application could be different. In general, for smaller values of P_{ind} , more cores should be used and the optimal number of blocks becomes larger.

Recall that the number of cores on each block is the power of 2. For a given application running on a multi-core platform with n_c cores, to find out the most energy efficient symmetric block configuration, we can first find out the optimal number of blocks to be used for minimizing energy consumption while meeting the timing constraint for the configuration with nc_i cores on one block. By comparing the minimum energy consumption from different symmetric block configurations, the optimal symmetric block configuration can be found.

For the case of asymmetric configuration, after finding out the optimal number of cores nc_{opt} to be used to minimize the energy consumption for the application, appropriate blocks can be selected and the energy consumption can be calculated accordingly. Compared to that of the optimal symmetric block configuration, the optimal configuration can be obtained.

4. DPM for BP-CMPs

As most real-time tasks use only a fraction of their WCETs at run-time [7], significant amount of dynamic slack is expected and various DPM schemes have been proposed to use such slack for more energy savings. We have studied one global scheduling based DPM algorithm for a set of independent real-time tasks on multiprocessor systems, namely *global scheduling with slack sharing (GSSR)* [20], where dynamic slack is shared among processors at run-time to obtain more energy savings. In this work, we extend GSSR to the block-partitioned CMPs.

4.1. Illustration Example

For the previous example in Figure 2, suppose that the application has the deadline $D = 9$. During one actual running, suppose that the 10 tasks take 1, 3, 2, 3, 2, 1, 2, 1, 1 and 1 time units, respectively.

For the case of one core per block as shown in Figure 3(a), the supply voltage and frequency for each core can be managed separately, and GSSR works the same as in previous work [20]. That is, at time 0, all cores start to run at the maximum frequency f_{max} . At time 1, task T_0 finishes early on core c_0 and 5 units of dynamic slack are generated. After sharing 2 units of the slack with core c_2 , 3 units of the slack will be used to slow down the execution of the next task T_4 running on core c_0 to the speed of $0.5f_{max}$. The detailed explanation of GSSR can be found in [20] and is omitted for brevity. Similarly, after task T_2 finishes early at time 2, the frequency for core c_2 will be set as $0.6f_{max}$ as shown in Figure 3(a).

For the case where all the cores are on the same block as shown in Figure 3(b), when task T_0 finishes early at time 1, core c_0 cannot execute the next task T_4 at the lower frequency $0.5f_{max}$, since other cores are expected to run at f_{max} to meet the deadline. At time 3, there is dynamic slack on every core and the *expected* running frequencies for the cores are $0.5f_{max}$, $0.4f_{max}$, $1/3f_{max}$, and $0.2f_{max}$, respectively. Thus, one *block-wide* frequency adjustment can be performed. To ensure

all cores meet the timing constraints, the common frequency will be set as the maximum expected running frequencies of all cores on the same block ($0.5f_{max}$ in this case). Figure 3(c) further shows the case of symmetric-2 configuration, where the two blocks adjust their frequencies to $0.5f_{max}$ and $1/3f_{max}$, respectively, at time 3.

4.2. GSSR for BP-CMPs

From the above example, we can see that, for block-partitioned CMPs, the dynamic voltage and frequency adjustment has to consider all the cores on one block. That is, depending on the amount of available dynamic slack, each core may have one *expected* running frequency. However, to ensure that other cores on the same block can meet the timing constraint, the common frequency for the cores on the same block will be the *maximum* of their expected running frequencies.

Algorithm 1 GSSR for BP-CMPs

```

1: /* The current core is  $c_x$ ;*/
2: while  $Ready-Q \neq \emptyset$  do
3:    $T_k =$  get next task from  $Ready-Q$ 
4:   /* Slack sharing.*/
5:   Find core  $c_i$  with minimum  $STNT_i$ ;
6:   if ( $STNT_x > STNT_i$ ) then
7:     switch  $STNT_x$  and  $STNT_i$ 
8:   end if
9:    $EFT_k = STNT_x + wc_k$ 
10:   $STNT_x = EET_k$ 
11:   $f_x = wc_k / (EET_k - t)$ ; //expected frequency
12:  For block  $b_i$  where  $c_x \in B_i$ ,
13:   $f_i = \max\{f_j | c_j \in B_i\}$ ;
14:  all cores on this block will run at  $f_i$ ;
15: end while

```

Algorithm 1 summarizes the steps of the GSSR algorithm when being applied to BP-CMPs. The terms used are the same as in [20]. The expected finish time (EFT) for a task is the time when the task finishes execution if it uses all the time allocated to it. The start time of next task (STNT) for a core is defined as the time when the next task starts its execution on this core. Following the same steps as in [20], when a core c_x finishes its task at time t , the expected running frequency f_x will be calculated after proper slack sharing (lines 3 to 11). After that, suppose that core c_x on block b_i , we find out the common frequency f_i for block b_i as the maximum expected running frequency of all cores on this block (lines 12 and 13). At last, we set f_i for all cores on this block.

Note that, since the common frequency is no lower than the expected running frequency on any core, all tasks will finish before its EFT, which in turns guarantees to meet the deadline [20].

5. Simulation Results

In this section, we evaluate the power efficiency of different block configurations for the BP-CMPs through extensive simulations. For comparison, in the **baseline** approach, all cores on the chip are used and execute the tasks at f_{max} . When a core finishes all workload on it, it is put into sleep for energy saving.

As the number of cores on single chip keeps increasing [18], in our simulations, we consider CMPs with 32 cores, 64 cores, and 128 cores, respectively. For the parameters in the power model, it is assumed that $C_{ef} = 1$, $m = 3$ and normalized frequency is used with $f_{max} = 1$. That is, the maximum frequency dependent active power on each core is $P_d^{max} = 1$. Moreover, we assume that the static power $P_s = 0.01$. For the block b_i with nc_i cores on it, the block-wide frequency independent active power is assumed to be $P_{ind}^i = 0.1 * nc_i$. The synthetic real-time applications are randomly generated with each task set contains 10,000 tasks, where their WCETs are randomly generated following uniform distribution between [10, 100]. The results reflects an average of the normalized energy consumption over 100 runnings.

5.1. SPM Results

Figure 4 first shows the normalized energy consumption for different symmetric block configurations under SPM. Here, X-axis is the ratio of the deadline over the minimum schedule length assuming that all cores run at f_{max} and the workload is perfectly balanced. It represents the amount of available static slack, where larger ratio values indicate more static slack. Y-axis is the normalized energy consumption with the one consumed by the baseline approach being 1.

In the figure, for each block configuration, the optimal number of blocks are used to obtain the minimum energy consumption with given amount of static slack. From the figure, we can see that, when the amount of available static slack is limited (e.g., when the ratio is less than 5), all cores will be used and similar performance is obtained for different block configurations. However, for the situations where excessive amount static slack is available, for the symmetric block configuration with more cores on one block (such as 32 cores per block for CMPs with 32 cores in Figure 4(a), 32 or 64 cores per block for CMPs with 64 cores in Figure 4(b), and 64 or 128 cores per block for CMPs with 128 cores in Figure 4(c)), SPM may consume more energy compared to the baseline. The reason is that, for the block contains more cores, at least one block will be used, and all cores on it will be used, incurring more frequency independent power. When excessive amount of static slack is available, all cores on that block will be slowed down to a very lower frequency, which could be energy inefficient [21].

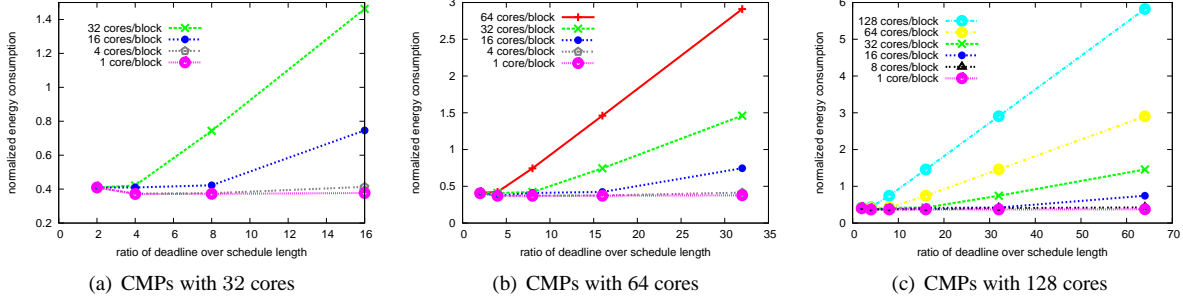


Figure 4. Normalized energy consumption for SPM with symmetric-partitioned CMPs.

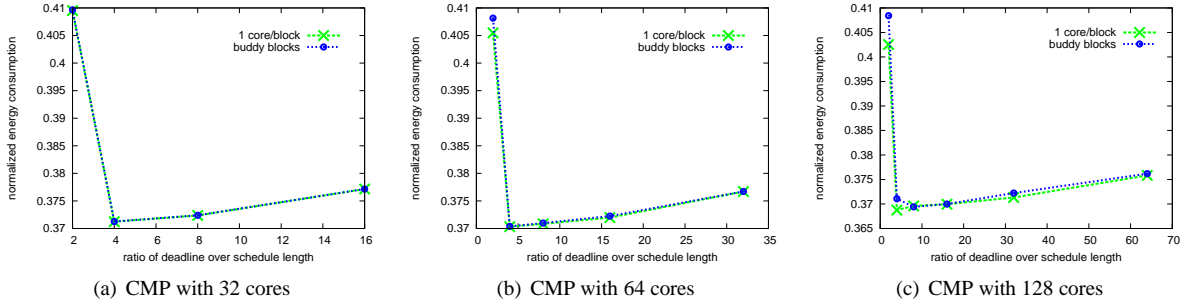


Figure 5. Normalized energy consumption for buddy asymmetric and symmetric-1 configurations.

As expected, for the block configurations with fewer number of cores on each block (e.g., 1, 4 or 8 cores per block), we can always use appropriate number of blocks (while switching off other unused blocks) to obtain the best energy savings by cutting down the frequency independent power. However, with fine-grained block configurations, more power supplied will be needed, which could incur high design complexity.

To illustrate the energy efficiency of the asymmetric configuration, Figure 5 further shows the normalized energy consumption for the asymmetric and symmetric-1 (i.e., cores having individual blocks) configurations. From these results, we can see that the asymmetric configuration can achieve almost the same level of energy efficiency as the symmetric-1 configuration for all the different settings. However, recall that the number of power supplies needed under asymmetric configuration is much smaller compared to that of symmetric-1 configuration.

5.2. DPM Results

To emulate the run-time behaviors of real-time tasks, we use one parameter $\alpha = \frac{AVET}{WCET}$ to control the amount of dynamic slack to be generated online, which is represented as the X-axis in Figure 6. Here, AVET denotes the average case execution time of real-time tasks, and the

smaller α is, the more dynamic slack will be available at run-time. The actual execution of a task is generated based on the task's AVET following a uniform distribution. For this part, we assume that all cores on the chip will be used and tasks finish just in time (i.e., there is no static slack).

From the results in Figure 6, we can see that, first, when less dynamic slack is available (i.e., larger values of $\frac{AVET}{WCET}$), all cores need to run at higher frequency which lead to more energy consumption. Second, the performance difference between different configurations is rather small (less than 5%). At last, the symmetric-1 configuration performs the worst, especially when modest amount of dynamic slack is available (e.g., $0.5 \leq \frac{AVET}{WCET} \leq 0.7$). The reason is that, with individual power supply for each core, the processing frequency for the cores will be adjusted independently. Considering that GSSR is greedy oriented scheme, where all available slack will be given to the next ready task running on one core, the cores will run at different frequencies most of the time. On the other hand, for the configurations with more cores on one block, due to the limitation of the power supply, the cores are more likely to execute at the same frequency, which turns out to be more energy efficient under DPM schemes.

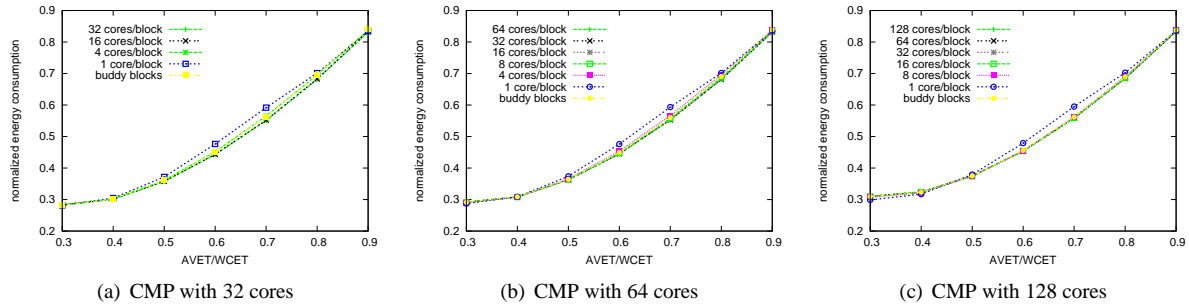


Figure 6. Normalized energy consumption for DPM

6. Conclusions

In this work, we propose block-partitioned configurations for multicore platforms, where the processing cores on a CMP chip are partitioned into different blocks and cores on one block share the same power supply (thus have the same frequency). Focusing on real-time applications consisting a set of independent tasks, we study both static and dynamic power management schemes for real-time systems on such block-partitioned multicore platforms. For different block configurations (both symmetric and asymmetric), we discuss the tradeoff between energy efficiency and design complexity. Simulation results show that block-partitioned CMPs (especially the ones with asymmetric buddy block partitions) have its inherent advantages to fulfill the goal of power efficiency and low design complexity for future CMPs, and warrant further investigations.

References

- [1] Intel Core 2 Quad. <http://www.intel.com/products/processor/core2quad/>, 2008.
- [2] H. Aydin, V. Devadas, and D. Zhu. System-level energy management for periodic real-time tasks. In *Proc. of RTSS*, 2006.
- [3] H. Aydin and Q. Yang. Energy-aware partitioning for multiprocessor real-time systems. In *Proc. of IPDPS, Workshop on WPDRTS*, 2003.
- [4] T. D. Burd and R. W. Brodersen. Energy efficient cmos microprocessor design. In *Proc. of HICSS*, 1995.
- [5] J. Chen. Multiprocessor energy-efficient scheduling for real-time tasks with different power characteristics. In *Proc. of ICPP*, 2005.
- [6] J. M. Cohn, D. W. Stout, P. S. Zuchowski, S. W. Gould, T. R. Bednar, and D. E. Lackey. Managing power and performance for system-on-chip designs using voltage islands. in *Proc. of ICCAD*, 2002.
- [7] R. Ernst and W. Ye. Embedded program timing analysis based on path clustering and architecture classification. In *Proc. of ICCAD*, 1997.
- [8] S. Herbert and D. Marculescu. Analysis of dynamic voltage/frequency scaling in chip-multiprocessors. In *Proc. of ISLPED*, 2007.
- [9] AMD Quad-Core processor <http://multicore.amd.com/user/AMD-Multi-Core/Quad-Core-Advantage/At-Work-AMD-Opteron/Power-Efficiency.aspx>, 2007.
- [10] J. Hu, Y. Shin, N. Dhanwaday, and R. Marculescu. Architecting voltage islands in core-based system-on-a-chip designs. In *Proc. of ISLPED*, 2004.
- [11] T. Ishihara and H. Yauura. Voltage scheduling problem for dynamically variable voltage processors. In *Proc. of ISLPED*, 1998.
- [12] R. Jejurikar and R. Gupta. Dynamic voltage scaling for system wide energy minimization in real-time embedded systems. In *Proc. of ISLPED*, 2004.
- [13] W. Kim, M. Gupta, G.-Y. Wei, and D. Brooks. System level analysis of fast, per-core dvfs using on-chip switching regulators. In *Proc. of HPCA*, 2008.
- [14] C. M. Krishna and Y. H. Lee. Voltage clock scaling adaptive scheduling techniques for low power in hard real-time systems. In *Proc. of RTAS*, 2000.
- [15] L. Leung and C. Tsui. Energy-aware synthesis of networks-on-chip implemented with voltage islands. In *Proc. of DAC*, 2007.
- [16] D. Mossé, H. Aydin, B. R. Childers, and R. Melhem. Compiler-assisted dynamic power-aware scheduling for real-time applications. In *Proc. of Workshop on Compiler and OS for Low Power*, 2000.
- [17] S. Saewong and R. Rajkumar. Practical voltage scaling for fixed-priority rt-systems. In *Proc. of RTAS*, 2003.
- [18] S. Vangal, J. Howard, G. Ruhl, S. Digne, H. Wilson, J. Tschanz, D. Finan, P. Iyer, A. Singh, T. Jacob, S. Jain, S. Venkataraman, Y. Hoskote, and N. Borkar. An 80-tile 1.28tflops network-on-chip in 65nm cmos. In *Proc. of ISSCC*, 2007.
- [19] F. Yao, A. Demers, and S. Shenker. A scheduling model for reduced cpu energy. In *Proc. of SFCS*, 1995.
- [20] D. Zhu, R. Melhem, and B. R. Childers. Scheduling with dynamic voltage/speed adjustment using slack reclamation in multi-processor real-time systems. *IEEE Trans. on Parallel and Distributed Systems*, 14(7):686–700, 2003.
- [21] D. Zhu, R. Melhem, D. Mossé, and E. Elnozahy. Analysis of an energy efficient optimistic tmr scheme. In *Proc. of ICPADS*, 2004.