Energy Efficient Block-Partitioned Multicore Processors for Parallel Applications*

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

Energy management has become an important research area in the last decade. As an energy efficient architecture, multicore has been widely adopted. However, with the number of cores on a single chip continuing to increase, it has been a grand challenge to effectively manage the energy efficiency of multicore-based systems. In this paper, based on voltage island *and* dynamic voltage and frequency scaling (DVFS) *techniques, we investigate the energy efficiency of* block-partitioned *multicore processors, where cores are grouped into* blocks *and each block has a DVFS-enabled power supply. Depending on the number of cores on each block, we study both* symmetric *and* asymmetric *block configurations. We develop a* system-level *power model (which can support various power management techniques) and derive both block- and system-level* energy-efficient frequencies. *Based on the power model, we prove that, for* embarrassingly parallel *applications, having all cores on a single block can achieve the same energy savings as that of the individual block configuration (where each core forms a single block and has its own power supply). However, for applications with limited degrees of parallelism, we show the superiority of the* buddy-asymmetric *block configuration, where the number of required blocks (i.e., power supplies) is* logarithmically *related to the number of cores on the chip, in that it can achieve the same amount of energy savings as that of the individual block configuration. The energy efficiency of block-partitioned multicore systems is further evaluated through extensive simulations with both synthetic as well as a real life application.*

Keywords: Multicore Processors; Energy Management; Dynamic Voltage and Frequency Scaling (DVFS); Voltage Islands; Parallel Applications.

*This work was supported in part by NSF awards CNS-0720651.

1 Introduction

The ever-increasing power density due to the increased clock frequencies in traditional processors and the demand for low-power computing devices have stimulated active research on energy management in recent years. Although extensive work has been done to investigate various power management techniques for different computing systems (from battery-powered devices [4, 35, 41] to high performance servers connected directly to the power grid [7, 48]), effective power management remains as one of the most important challenges for the research and engineering community, both in industry and academia [23].

One common strategy to save energy in computing systems is to operate system components at low-performance (thus, low-power) states, whenever possible. For instance, as one of the most effective power management techniques, *dynamic voltage and frequency scaling (DVFS)* exploits the convex relation between processor dynamic power consumption and processing frequency/supply voltage [9] and scales down processors' processing frequency and supply voltage simultaneously to save energy [52]. For DVFS-enabled processors, there have been many research studies targeted at managing the power consumption of processors [4, 41, 53]. However, considering the increased static/leakage power due to scaled feature sizes [39] as well as other power consuming components (such as main memory [35] and I/O devices [37, 49]), system-wide energy management is gaining increasing importance [3, 14, 31].

Multicore processors, where multiple processing cores are integrated on a single chip [40], have emerged to be the popular and powerful computing engines for modern computing systems. An important design objective in multicore processors is *energy efficiency*. For example, to achieve the same level of performance, an application can be executed in parallel on multiple processing cores with each core running at a lower frequency for energy savings. Several recent research work has explored such features of multicore processors to save energy for different applications [6, 30, 32, 44].

Major chip makers have currently several processor lines with 4, 8 and 16 cores (e.g., Intel Core2 Quad [25], AMD Phenom [24] and Sun Niagara [45]), which have integrated various advanced power management features (e.g. DVFS) and multiple idle states (e.g. Halt, Sleep and Off [27, 47]). However, most state-of-the-art commercial multicore processors have only one common power supply voltage for all cores [17, 25], which may require the cores to run at the same processing frequency and limit the flexibility for power management (and thus result in sub-optimal energy savings).

With the advancement of *voltage island* techniques [15, 26, 36] and fast *on-chip voltage regulators* [33], it is expected that future multicore processors can have multiple supply voltage domains on a chip [45]. However, the increased design complexity and associated area overhead of the additional supply voltages

and on-chip voltage regulators [20] would make it very costly to support a separate DVFS-enabled supply voltage for each core, especially considering that the number of cores on a single chip continues to increase (where extensive research activity is underway to build chips with potentially tens and even hundreds of cores [8, 22, 38]). Therefore, **the problem of how to optimally place the cores on different voltage islands in a multicore chip for effective system-level power management remains open**.

In this paper, based on the voltage island and DVFS with on-chip voltage regulator techniques, we study the *block-partitioned* core configurations for multicore processors and evaluate the energy efficiency for different block configurations. Specifically, the cores on a multicore chip are grouped into several *blocks* (with each block being essentially a *DVFS-enabled voltage island*) and the cores on each block share a common supply voltage (and thus have the same processing frequency). Depending on how the cores are partitioned to blocks, we study both *symmetric* and *asymmetric* block configurations, which contain the *same* and *different* number of cores on each block, respectively. In particular, a *buddy-asymmetric* block configuration is studied for its flexibility to efficiently support different system workloads. The number of required blocks (i.e., power supplies) in buddy-asymmetric block configuration is *logarithmically* related to the number of cores on the chip. Furthermore, for such block-partitioned multicore systems, we develop a *system-level power model* that can effectively support various levels of power management techniques and derive both block- and system-level *energy-efficient frequencies*.

Based on the power model, we analyze the energy efficiency of different block configurations. First, for *embarrassingly parallel* applications, we prove that the *common block* configuration that has all cores on a single block (thus needs only one power supply) can achieve the same energy savings as that of the *individual block* configuration (where each core forms a single block and has its own power supply). Then, for applications with limited degrees of parallelism, we show the superiority of the *buddy-asymmetric* block configuration in that it can achieve comparable energy savings as that of the individual block configuration. Moreover, the energy efficiency of block-partitioned multicore systems is further evaluated through extensive simulations with both synthetic as well as real life applications, and the results confirm our analysis.

To the best of our knowledge, this is the first work that systematically analyzes and evaluates the energy efficiency of both symmetric and asymmetric block configurations of multicore systems for applications with various degrees of parallelism. To summarize, the main contributions of this paper are threefold:

- We propose block-partitioned core organizations for multicore processors, where both symmetric and asymmetric block configurations are considered;
- We develop a system-level power model for systems with such block-partitioned multicore processors

and derive both block- and system-level energy-efficient frequencies;

- We analyze the energy efficiency of block-partitioned multicore systems for both embarrassingly and limited parallel applications, where the analysis results are confirmed through extensive simulations.

The remainder of this paper is organized as follows. The closely related work is reviewed in Section 2. Section 3 presents block-partitioned core configurations for multicore processors and the corresponding system-level power model. In Section 4, we analyze the energy efficiency of different block configurations in multicore systems for applications with various degrees of parallelism. The simulation results are presented and discussed in Section 5 and Section 6 concludes the paper.

2 Related Work

Power aware computing has become an important research area that attracts extensive attention in the last decade. As the dynamic energy consumption of CMOS devices is quadraticly related to its supply voltage [9], *dynamic voltage and frequency scaling (DVFS)* technique that slows down the processing speed (and supply voltage) of CMOS devices can lead to significant energy savings [52]. Based on the DVFS technique, various power management schemes have been developed for uniprocessor real-time systems with different scheduling policies [4, 29, 41]. However, the research on power-aware scheduling for multiprocessor systems is comparatively limited, especially for multicore-based systems.

Based on the partitioned scheduling policy, Aydin *et al.* studied the problem of how to partition real-time tasks to processors for minimizing energy consumption for multiprocessor systems [5]. They showed that, for earlist deadline first (EDF) scheduling, balancing the workload among all the processors evenly gives the optimal energy consumption and the general partition problem for minimizing energy consumption in multiprocessor real-time system is NP-hard [5]. The work was extended to consider rate monotonic scheduling (RMS) in their later work [1]. Anderson and Baruah investigated how to synthesize a multiprocessor real-time system with periodic tasks such that the energy consumption is minimized at run-time [2]. Chen et al. proposed a series of approximation scheduling algorithms for maximizing energy-efficiency of multiprocessor real-time system, where both frame-based tasks and periodic tasks are considered, with and without leakage power consideration [11, 12, 10, 51]. In our previous work, based on global scheduling, power-aware algorithms have been developed for real-time multiprocessor systems, which exploits the slack reclamation and slack sharing for energy saving [53]. More recently, Choi and Melhem studied the interplay between parallelism of an application, program performance, and energy consumption [13]. For an application with given ratio of serial and parallel portions and the number of processors, the authors derived

optimal frequencies allocated to the serial and parallel regions in an application to either minimize the total energy consumption or minimize the energy-delay product.

For soft real-time applications running on multicore systems, Bautista *et al.* recently studied a novel fairness-based power aware scheduler that adapts the global frequency for all cores at the same time and evaluated the power efficiency of multicore processors [6]. The scheduler pursues to minimize the number of DVS transitions by increasing or decreasing the voltage and frequency of all the cores at the same time. Along the line of assuming all cores on a chip share the same frequency, Seo *et al.* also studied one dynamic re-partitioning algorithm for real-time systems which dynamically balances the task loads on the cores to optimize overall power consumption [44]. For non-real-time applications, Donald *et al.* concluded that the most effective approach for energy management in multicore-based systems is the combination of *core throttling* (i.e., powering off the unused cores) and *per-core DVFS* that adjusts the processing frequency and supply voltage of cores independently [16]. Although it is possible to provide a separate power supply voltage for each individual core to get the maximum flexibility, as the number of cores on a chip continues to increase (with tens [50] or even hundreds of cores [22]), the increased overhead [20] of such a feature may come with additional circuit complexity, stability, and power delivery problems [8, 21, 38].

In [21], based on detailed VLSI circuit simulations, Herbert *et al.* found that the potential energy gains of *per-core* DVFS are likely to remain too modest for justifying the complicated design problems. Similar conclusion has been reached independently in [28], which states that the additional energy gains from per-core DVFS would not be substantial for reasonably-balanced workload-to-core distributions. Therefore, several recent studies [8, 26, 36] have focused on the *voltage/frequency island* technique, where the cores on a multicore chip are partitioned into a few number of groups with each group being placed on a voltage island that has a separate power supply and independent voltage regulator. For the cores on the same voltage island, they will have the same processing frequency (and supply voltage) that can be adjusted at the same time but may independently enter low-power sleep states to save energy when idle [34]. Some very recent experimental studies indicate that innovative techniques could facilitate the implementation of fast on-chip voltage regulators [33].

Following this line of research, in this work, we investigate energy efficient organizations of cores on multicore processors. Specifically, we study block-partitioned multicore processors, where the processing cores are grouped into different blocks and each block is essentially a DVFS-enabled voltage island. However, different from the existing work that focused on only voltage islands containing the same number of cores [8, 21, 26, 36], we consider both *symmetric* blocks (where the number of cores is the same on each block) and *asymmetric* blocks (where the number of cores on the blocks are different), and evaluate the

energy efficiency of these configurations for application with various degrees of parallelism.

3 Block-Partitioned (BP) Multicore Processors

In this section, we first explain how the cores are partitioned under different block configurations. Then, for such block-partitioned multicore systems, we develop a simple system-level power model which at the same time can effectively support various power management techniques. Based on the power model, we derive the energy efficient frequencies that form the foundation to analyze the energy efficiency of different block configurations for applications with various degrees of parallelism.

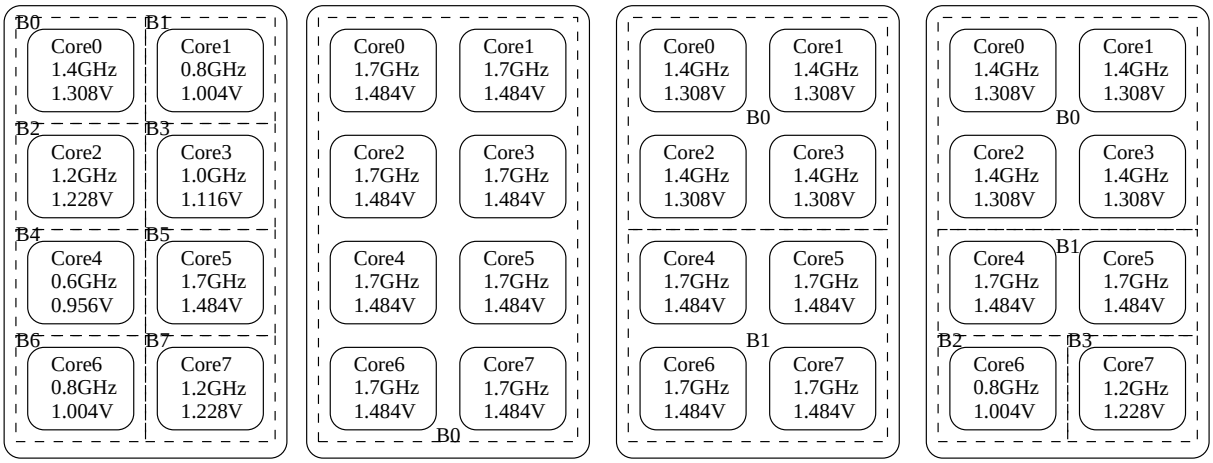
3.1 Partition Cores to Blocks

As previously mentioned, the most flexible approach for efficient energy management for multicore systems is to have a separate DVFS-enabled supply voltage for each individual core [16]. However, such configuration requires a separate voltage island and on-chip voltage regulator for each core, where the increased design complexity and associated overhead cost can be prohibitive (e.g., a single on-chip voltage regulator can take 12.7% of the whole chip area for the $90nm$ technology [20]). Moreover, with reasonably-balanced distributions of workload to cores, additional energy savings from such per-core DVFS facilities can be limited [21, 28]. Therefore, it would be more appropriate to place several cores on one voltage island and allow them to share the supply voltage (thus have the same processing frequency) [26, 36], especially considering the fact that the number of cores on a chip will continue to increase [8, 22, 38]. In general, having more cores on a single voltage island can reduce the number of required power supply voltages (and associated on-chip voltage regulators) and thus reduce the design complexity and overhead cost, it may also limit the energy management opportunities, and vise versa. Therefore, there is an interesting tradeoff between the overhead cost (i.e., the number of required power supply voltages and associated on-chip voltage regulators) and energy efficiency.

In this work, it is assumed that the multicore processor under consideration has n *homogeneous* processing cores, where the cores have *identical* processing capacity. Moreover, for simplicity and ease of discussions, we further assume that n is a power-of-two value (i.e., $n = 2^k$; $k \geq 1$). To reduce the design complexity and the number of required power supplies, the cores will be partitioned into *blocks*, where each block is essentially a *DVFS-enabled voltage island*. For the cores on the same block, they will share the same supply voltage. Although it is possible for the cores to run at different *lower* processing frequencies with the same high supply voltage, doing so is not energy efficient and the additional circuits needed will make the design more complex. For simplicity, we will not consider this aspect in this paper. That is, we

assume that all cores on the same block will have the same *highest allowable* processing frequency for a given supply voltage. Therefore, without introducing ambiguity, we will use *frequency scaling* to stand for adjusting both voltage and frequency simultaneously for the remaining part of this paper.

In what follows, depending on how the cores are partitioned, we focus on two categories of partitions: *symmetric* and *asymmetric* blocks.



(a) Symm-1 (b) Symm-8 (c) Symm-4 (d) Asym

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

3.1.1 Symmetric Blocks

For symmetric blocks, each block will have the *same* number of cores. However, for a multicore processor with a given number of cores, we can have different symmetric configurations depending on the number of cores on each block. These configuration will have different numbers of blocks, which in turns require different number of power supply voltages and have different design complexity and overhead cost. Note that, quantifying the exact design complexity and overhead cost of different block configurations is well beyond the scope of this paper. For simplicity, we use the number of resulting blocks to represent the design complexity and overhead cost.

As one concrete example, suppose that we have a multicore processor with 8 cores. One case is to have each core have its own power supply and form one individual block (denoted as *Symm-1*) as shown in Figure 1(a). Here, the dotted rectangles represent the blocks. In this case, the cores can have different processing frequencies depending on their own workload. Moreover, when some cores are idle, they can be powered off separately. On the other hand, we can have all cores share a single power supply to form a

common block (denoted as *Symm-8*) as shown in Figure 1(b). Here, the supply voltage will be determined by the most loaded core and all cores will operate at the same high processing frequency. Although the idle cores can be put to sleep states to save energy, we cannot power off the block for better energy savings even if there is only one core that is actively processing its workload.

Therefore, we can see that the *Symm-1* configuration can provide the best flexibility for power management. However, it will require 8 separate power supply voltages (and associated on-chip voltage regulators), which will lead to high design complexity and overhead cost. For comparison, the *Symm-8* configuration only needs a single power supply voltage, which will have the lowest design complexity and overhead cost. However, it may result in limited power management opportunities for the cores. For a better tradeoff between power management flexibility and design complexity/cost, we can have either four (4) cores on each block (denoted as *Symm-4* as shown in Figure 1(c)) or two cores for each block (denoted as *Symm-2*), which will require 2 and 4 power supply voltages, respectively. The energy efficiency of these configurations with symmetric blocks will be analyzed and evaluated in Section 4 and Section 5, respectively.

3.1.2 Asymmetric Buddy Blocks

Instead of having the same number of cores on each block, we can have *asymmetric* blocks, where each block has different number of cores. With the objectives of reducing the number of blocks (thus the number of power supply voltages) and providing flexible power management support for various workloads, we consider in particular the *asymmetric buddy blocks* in this work. It follows the similar idea of buddy memory allocation in operating systems [46]. For instance, for the above example with 8 cores, Figure 1(d) shows the asymmetric buddy configuration, where there are 4 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 the cores on a chip, and the second block contains half of the remaining cores, and so on. The last two blocks will contain one core each.

In general, for a multicore processor with $n = 2^k (k \geq 1)$ processing cores, there will be $(k+1)$ blocks for the asymmetric buddy configuration and the number of cores on the blocks will be $2^{(k-1)}, 2^{(k-2)}, ..., 2^1$, 2^0, 2^0, respectively. From the configuration, with each block having a separate power supply, the number of required supply voltages (and on-chip voltage regulators) under the asymmetric buddy configuration N^b_{buddy} is *logarithmically* related to the number of cores on a multicore chip. That is $N^b_{buddy} = log_2(n) + 1$.

Note that, for multicore processors with the asymmetric buddy configuration, to satisfy the requirement of various workload, we can adopt *exactly* p $(1 \leq p \leq 2^k)$ cores by appropriately selecting the blocks to be powered on. More specifically, the first block to be selected will be the largest block B_i with its number of

cores n_i being no more than p. If that block contains exactly p cores (i.e., $n_i = p$), only the block B_i will be selected. Otherwise, more blocks are needed to provide $(p - n_i)$ cores. The next block to be selected will be the one (from the remaining blocks) that has the largest number of cores n_j being no larger than $(p - n_i)$. The above steps will be repeated until the total number of cores from the selected blocks is exactly p.

For the above example, if a given 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). For comparison, with the Symm-4 configuration as shown in Figure 1(c), we have to use both blocks to satisfy the performance requirements, which could be sub-optimal for energy savings. It is also possible to run exactly 5 cores by exploiting the Symm-1 configuration as shown in Figure 1(a). However, the Symm-1 configuration requires 8 power supply voltages compared to the asymmetric buddy configuration that only needs 4 power supply voltages. Furthermore, in Sections 4 and 5, we will show that the resulting energy efficiency of the asymmetric buddy configuration is the same as that of the Symm-1 (i.e., individual block) configuration.

3.2 System-Level Power Model for BP-Multicore Systems

To effectively evaluate the energy efficiency of different block configurations, we develop in this section the power model for computing systems with a block-partitioned multicore processor. Note that, power management schemes that focus on individual components may not be energy efficient at system level. For instance, to save the energy consumption of processors, DVFS tends to run a system at the lowest processing frequency that satisfies a given performance requirement to save processor's energy consumption. However, such low processing frequency will need more time to execute the application under consideration and thus incur more energy consumption from memory and I/O devices. Therefore, system-wide power management becomes a necessity and has caught researchers' attention recently [3, 31, 43].

For a uniprocessor system, by dividing its power consumption into three parts, we have studied in our previous work a simple system-level power model [3, 54], where the power consumption of a computer system running at frequency f is modeled as:

$$P(f) = P_s + \hbar(P_{ind} + P_d) = P_s + \hbar(P_{ind} + C_{ef} \cdot f^m) \tag{1}$$

Here, P_s is the *static power* used to maintain the basic circuits of the system (e.g., keeping the clock running), which can be removed only by powering off the whole system. Whenever the system is active and executing some workload (i.e., $\hbar = 1$), the *active power*, which has two parts P_{ind} and P_d, will be consumed. Here, P_{ind} denotes the *frequency-independent active power*, which is a constant can be effectively

removed by putting the power manageable components of the system into sleep states. P_d denotes the *frequency-dependent active power*, which includes the processor's dynamic power as well as any power that depends on system supply voltages and processing frequencies [9, 18]. Despite its simplicity, this power model includes all essential power components of a system and can support various power management techniques (e.g., DVFS and sleep states).

Following the similar idea and extending the above power model, we develop in what follows a system-level power model for block-partitioned multicore systems. Note that, the processing cores in modern multicore processors can be efficiently (e.g., in a few cycles) put into power-saving sleep states [27, 47]. With the processing cores being partitioned into blocks and each block has a DVFS-enabled supply voltage, there are several places at different levels to manage the power consumption of a block-partitioned multicore system. First, at the processing core level, we can exploit DVFS techniques and scale down the processing frequency (and corresponding supply voltage) for all cores on one block to save energy provided that the performance requirement of the most loaded core(s) can still be met. Second, whenever a core on a block finishes its workload and becomes idle, we can put it into *sleep* states for more energy savings. Third, at the block level, if all cores on a block are in sleep states and are expected to remain in sleep states in the near future, we can switch off the power supply for the block and put it to the *off* state to save part of the static and leakage power. Finally, at the system level, we may completely *power off* the whole system when it is not in use and all power consumption will be removed. However, considering the excessive time overhead for completely powering on/off a computing system (e.g., tens of seconds [7]), for the system under consideration, we assume that it is never powered off completely and focus on, in this paper, the power management techniques at the core and block levels.

To effectively support these different opportunities for power management in block-partitioned multicore systems, following the principles of the power model shown in Equation (1), we also divide the system power consumption into several distinct components. Associated with each core, there is *frequency-dependent active power* that depends on the processing frequency (and supply voltage) of the block and can be efficiently removed by putting the core into power saving sleep states. The *frequency-independent active power* is assumed to associate with a block, which is proportional to the number of cores on the block (as the leakage power normally depends on the number of circuits) and can be removed by switching off the power supply for the block. Finally, there is a *static system power* that is a constant and can only be removed by powering off the whole system. Before formally presenting the system-level power model, we first define a few important terms:

- b: the number of blocks on the multicore processor in the system under consideration;

- B_i: the ith block of the multicore processor, where $i = 0, \ldots, b-1$;
- n_i: the number of cores on the block B_i. We have $n = \sum_{i=0}^{b-1} n_i$;
- $f_i(t)$: the processing frequency for the cores on the block B_i at time t;
- $x_i(t)$: a binary variable to indicate the state of block B_i at time t. If the block is powered off, $x_i(t) = 0$; otherwise, $x_i(t) = 1$;
- $y_{i,j}(t)$: a binary variable to indicate the state of the jth ($j = 1, \ldots, n_i$) core on block B_i at time t. If the core is in sleep state, $y_{i,j}(t) = 0$; otherwise, $y_{i,j}(t) = 1$;

It can be seen that the power consumption of a block-partitioned multicore system depends on the states of its blocks as well as individual cores. For a given run-time state of the system at time t, which is defined by $x_i(t)$, $f_i(t)$ and $y_{i,j}(t)$ ($i = 0, \ldots, b-1$ and $j = 1, \ldots, n_i$), the power consumption $P(t)$ of a multicore system can be modeled as:

$$P(t) = P_s + \sum_{i=0}^{b-1} x_i \cdot \left(P_{ind,i} + \sum_{j=1}^{n_i} (y_{i,j} \cdot P_{d,i,j}(f_i)) \right) \quad (2)$$

$$P_{d,i,j}(f_i) = C_{ef} \cdot f_i^m \quad (3)$$

where P_s is the static power. With the assumption that the system is never powered off completely due to the switching overhead, P_s is always consumed. However, the blocks can be powered off and other components can be put to power-saving sleep states for energy savings. C_{ef} and m (generally it is assumed that $m = 3$ [9]) are system dependent parameters. Recall that all cores on block B_i run at the same frequency f_i due to the limitation of block-wide power supply.

For ease of presentation, the maximum frequency-dependent active power for one core at the maximum frequency f_{max} is defined as P_d^{max}. In addition, to capture the relationship between the frequency-independent and frequency-dependent active powers, we further assume that each core contributes $\beta \cdot P_d^{max}$ to the frequency-independent active power for the block it resides in. That is, $P_{ind,i} = n_i \cdot \beta \cdot P_d^{max}$. Note that, for the available modern DVFS-enabled processors (e.g. Intel [25] and AMD [24]), they normally have only a few frequency levels. In this work, we assume that there are k frequency levels: f_1, ..., f_k, where $f_1 = f_{min}$ is the minimum available frequency and $f_k = f_{max}$ is the maximum frequency. Moreover, we use normalized frequencies in this work and it is assumed that $f_{max} = 1$. For simplicity, the time overhead of adjusting frequency (and supply voltage) for the blocks is assumed to be negligible[1].

[1] Such overhead can be easily incorporated into the execution time of the applications under consideration when exploiting slack time to scale down the processing frequency [4, 53].

3.3 Energy-Efficient Frequencies for BP-Multicore Systems

From previous discussions, we can see that, for block-partitioned multicore systems, processing frequencies for cores will be determined at block level. That is, depending on the allocated workload and the number of active cores on each block, blocks may have different processing frequencies for better energy savings. Focusing on a single block B_i that has n_i processing cores, we first derive in what follows the *optimal* frequency setting for the block to achieve the maximum energy savings, which will be exploited in Section 4 to analyze energy efficiency of different block configurations for various parallel applications.

For the cores on the block B_i, not all of them will be active all the time due to, for example, the limited degrees of parallelism in applications. Note that energy is the integral of power over time. Therefore, although running at a lower frequency can reduce the energy consumption due to the frequency-dependent power for active cores on the block B_i, the increased execution time will lead to more energy consumption from the frequency-independent power for the block. That is, similar to the energy-efficient frequency for uniprocessor systems [3, 54], a *block-wide energy-efficient frequency* will exist and the active cores should not run below such frequency as doing so will consume more energy. From Equation 2, we can see that the frequency-dependent power component is associated with cores and depends on the number of active cores that have workload to execute. Intuitively, when more cores are active, we can have a lower energy-efficient frequency as more energy savings can be expected from frequency-dependent power (which can compensate the additional energy consumption due to frequency-independent power) and vice versa.

Recall that there are n_i cores on block B_i. Suppose that the number of active cores is a_i ($\leq n_i$) and each active core has the same workload w_i, which will be executed at the scaled frequency f_i. The time needed for the active cores to finish executing the workload will be $t = \frac{w_i}{f_i}$. After putting the idle cores to power saving sleep states, the active energy consumption for the block B_i to execute the required workload can be given as:

$$E_i(a_i, f_i, t) \quad = \quad (P_{ind,i} + a_i \cdot C_{ef} \cdot f_i^m) \cdot t \tag{4}$$

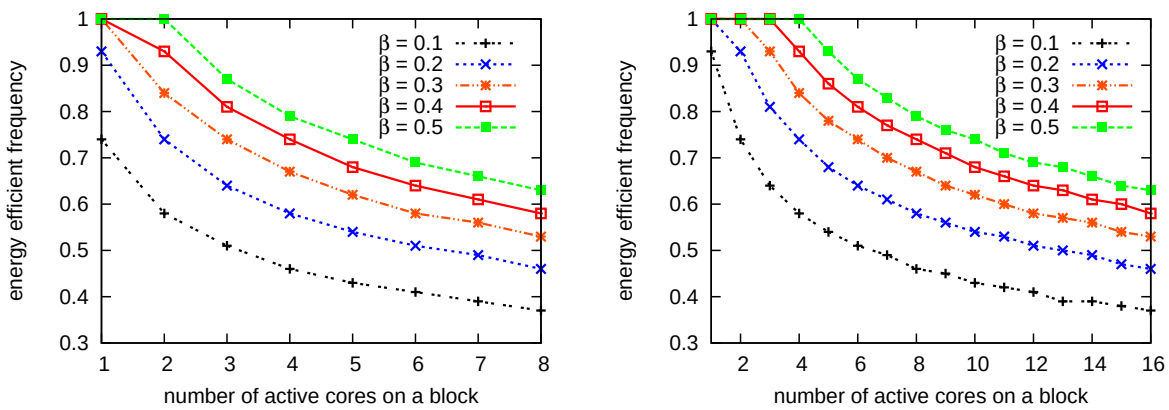
From the above Equation, we can see that the active energy consumption to execute the workload on block B_i is a convex function of f_i. Differentiate $E_i()$ with f_i and set the resulting equation to be zero, we can find out that $E_i()$ is minimized when f_i equals the following *energy-efficient frequency*:

$$f_{ee,i}(a_i) \quad = \quad \sqrt[m]{\frac{n_i \cdot \beta \cdot C_{ef}}{a_i \cdot (m-1)}} \tag{5}$$

Even if there are more available time to execute the workload, we should not have the active cores run at a frequency lower than $f_{ee,i}(a_i)$, as doing so will consume more energy. Moreover, as the number of active cores a_i becomes smaller, the energy-efficient frequency $f_{ee,i}$ for the block B_i will become larger and the active cores need to run faster for the best energy savings. When all cores on the block B_i are active (i.e., $a_i = n_i$), Equation (5) can be simplied as:

$$f_{ee} = \sqrt[m]{\frac{\beta \cdot C_{ef}}{(m-1)}} \tag{6}$$

That is, for any block that have all its cores active, the same energy-efficient frequency f_{ee} can be obtained, which also denotes the **system-wide energy-efficient frequency** when all cores of a multicore processor are active.



(a) 8 cores on a block (b) 16 cores on a block

Figure 2. Energy-Efficient Frequency for Blocks with Different Number of Cores.

In addition to the number of active cores, from Equation (5), the energy-efficient frequency also depends on the frequency-independent active power of a block. For different frequency-independent active power (i.e., different values of β), Figures 2(a) and 2(b) show the energy-efficient frequency of blocks with 8 and 16 cores, respectively, as the number of active cores changes. From the figures, we can see that as the number of active cores decreases, the energy-efficient frequency becomes higher and the cores need to run faster to achieve the best energy savings. If there is only a few (e.g., 1 or 2) active cores, having them run at f_{max} can be the most energy efficient approach, especially when the block's frequency-independent power is large (i.e., for larger values of β). In what follows, the energy-efficient frequency will be utilized to evaluate the energy efficiency of different block configurations for parallel applications.

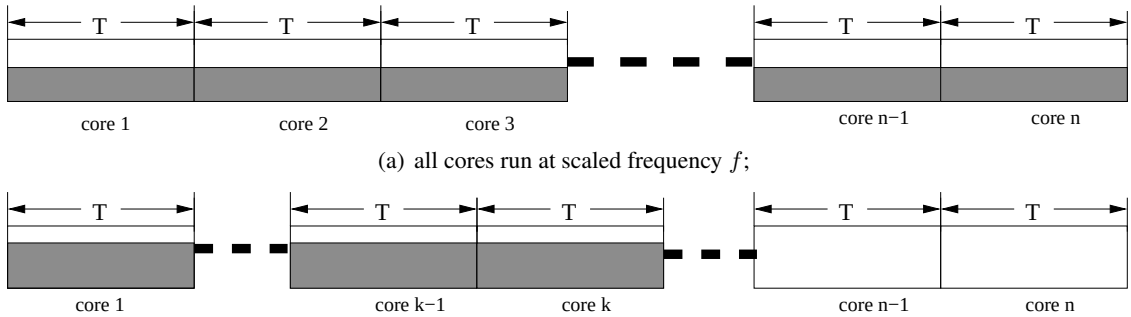
4 Energy Efficiency of BP-Multicore for Parallel Applications

For parallel applications running on a system with a single block-partitioned multicore processor with n cores, we analyze the energy efficiency for different block configurations, which provide different flexibility to scale down the processing frequency and/or power off individual cores to save energy. If there is enough workload and all n cores will be utilized by applications all the time, no power management can be applied and the system will consume the same power (and energy) regardless different block configurations of the cores. However, as applications (or different phases in an application) have different degrees of parallelism, not all cores will be employed all the time and the system may consume different energy with different block configurations. For ease of presentation, normalized energy consumptions are reported and system energy consumption with the *common block configuration* (where all cores are on a single block and share a common power supply) is used as the baseline.

To simplify the analysis, the workload W of the application under consideration is assumed to have the degree of parallelism DP, which is an integer. For applications that have varying degrees of parallelism in different phases, the same analysis can be applied to each phase separately. Moreover, we assume that the workload should be processed within a period of time T, which can be derived from the user specified performance requirements or the inter-arrival time of a repetitive application. The *system load* is defined as $\delta = \frac{W}{n \cdot T \cdot f_{max}}$, which is the ratio of application's workload W over the total computation capacity of the system at the maximum frequency f_{max}. Note that, the system load does not take the workload's parallelism into consideration and the application may not be able to finish in time even when $\delta < 1$ due to limited parallelism. Depending on the degree of parallelism, we focus our analysis on two types of applications: the *embarrassingly parallel* applications and the ones with *limited parallelism* (i.e., $DP \leq n$).

4.1 Embarrassingly Parallel Applications

For applications with embarrassing parallelism, it is assumed that the workload W can be *arbitrarily* divided and executed in parallel on *any* number of cores (e.g., the large number of small requests to be processed in a large interval in web applications) [19]. That is, for the system with n cores, we can *evenly* distribute the workload W among the cores and the minimum amount of time required to process the workload at the maximum frequency f_{max} will be $t_{min} = \frac{W}{n}$. If $t_{min} > T$ (i.e., $\delta > 1$), the workload can not be processed in time. Otherwise (i.e., $t_{min} \leq T$), we can scale down the processing frequency of the cores to save energy. In what follows, we prove that the same amount of energy savings can be obtained under different block configurations for embarrassingly parallel applications with $\delta \leq 1$.



(a) all cores run at scaled frequency f;

(b) k cores are used to run at a higher scaled frequency f';

Figure 3. For the case of system load $\delta \geq f_{ee}$

Theorem 1 *For an embarrassingly parallel application with workload W to be executed on a multicore system with n cores, the same amount of energy will be consumed (i.e., the same energy savings can be obtained) if the system load $\delta \leq 1$.*

Proof

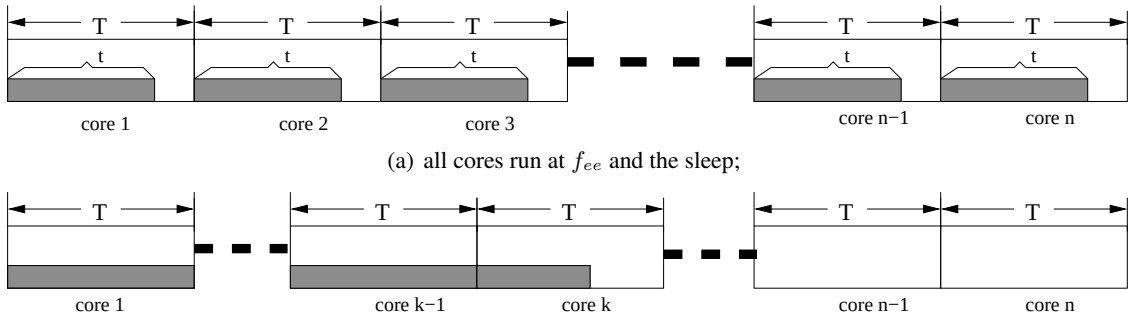
Recall that the system will not be completely powered off and the static power component P_s is always consumed, which will be the same for all block configurations. In what follows, we focus on the energy consumption from the frequency-dependent and frequency-independent active power.

When $\delta \leq 1$, by executing the workload on all n cores at the maximum frequency f_{max}, we can finish the execution at time $t_{min} = \frac{W}{n} \leq T$. For energy savings, we can scale down the processing frequency (and corresponding supply voltage) of all cores as low as $f = \frac{t_{min}}{T} = \frac{W}{n \cdot T} = \delta$. Recall that there is a system-wide energy-efficient f_{ee}, which limits the lowest energy-efficient frequency (see Section 3.3). Depending on the value of f, there are two cases:

Case 1 ($f \geq f_{ee}$): In this case, there is enough workload and we will show that all n cores should be utilized to minimize system energy consumption (thus to maximize energy savings).

When all cores are utilized and execute the workload at scaled frequency f, we can pack the execution on the cores one after each other as shown in Figure 3(a), which can also been seen as executing the workload on one core for time $n \cdot T$. The total energy consumption will be $E(f, n) = P_s \cdot T + (\beta P_d^{max} + C_{ef} f^m) \cdot n \cdot T$.

Suppose that only k ($< n$) cores are utilized to execute the workload W. The scaled frequency will be $f' = \frac{W}{k \cdot T} > f$. Such execution can also be seen as executing the workload on one core for time $k \cdot T$ (as shown in Figure 3(b)). Here, the total energy consumption will be $E(f', k) = P_s \cdot T + (\beta P_d^{max} + C_{ef} f'^m) \cdot k \cdot T$. Note that the idle cores are put to sleep for better energy savings. From the results in [3, 54], we know that $E(f', k) > E(f, n)$ due to the convexity in the system power function. Therefore, all cores should be utilized for the minimum energy consumption, which will be the same for different block configurations.



(a) all cores run at f_{ee} and the sleep;

(b) k cores are used to run at f_{ee} and other cores are in sleep;

Figure 4. For the case of system load $\delta < f_{ee}$

Case 2 ($f < f_{ee}$): In this case, for the best energy savings, we should execute the workload at the energy-efficient frequency f_{ee}.

If all cores are utilized, the workload can be completed at time $t = \frac{W}{n \cdot f_{ee}} < T$. Then all cores can be put to sleep to further save the energy consumption from the frequency-independent active power. Again, if we pack the execution on the cores one after each other, it can be shown as in Figure 4(a). The overall system energy consumption can be calculated as $E(n, f_{ee}) = P_s \cdot T + (\beta P_d^{max} + f_{ee}^m) \cdot n \cdot t = P_s \cdot T + (\beta P_d^{max} + f_{ee}^m) \cdot \frac{W}{f_{ee}}$.

If we execute the workload on only k ($\geq \frac{W}{f_{ee} \cdot T}$) cores while other cores are put to sleep for energy savings, we can get the packed execution as shown in Figure 4(b). Note that, using fewer than k cores will force them to execute at a frequency higher than f_{ee}, which is not energy efficient. Here, the total execution time in the gray area will be $\frac{W}{f_{ee}}$, and the energy consumption will be $E(k, f_{ee}) = P_s \cdot T + (\beta P_d^{max} + f_{ee}^m) \cdot \frac{W}{f_{ee}}$, which is the same as that of executing the workload on n cores.

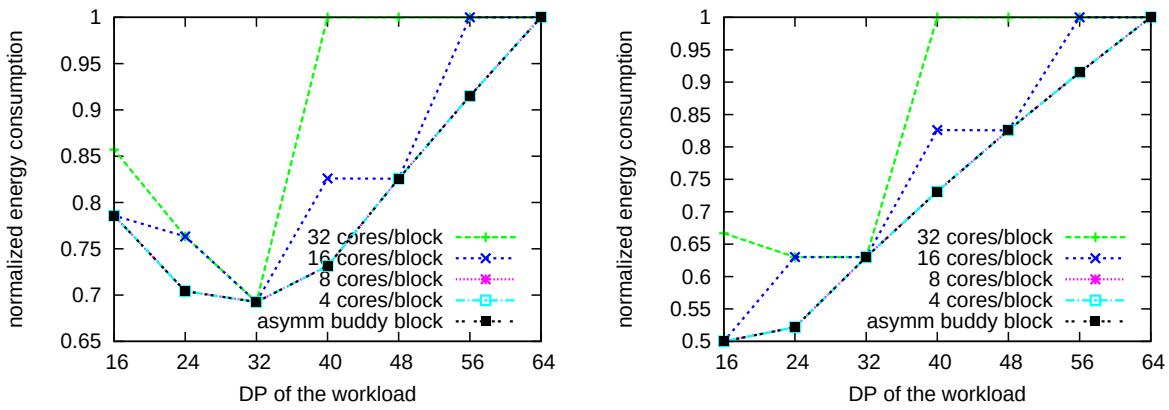
That is, for the case of $f = \delta < f_{ee}$, by enforcing the execution of the workload at f_{ee} (i.e., with no fewer than $\frac{W}{f_{ee} \cdot T}$ cores), the same minimum energy will be consumed.

To conclude, for embarrassingly parallel applications to be executed on a block-partitioned multicore system, the same minimum energy will be consumed regardless of how the cores are grouped into different blocks.

■

4.2 Applications with Limited Parallelism

Due to various dependencies, most applications only have limited parallelism. For applications with the limited degree of parallelism of $DP \leq n$, the number of cores to be utilized will be DP and the workload allocated to each core will be $\frac{W}{DP}$ (that is, we assume that the workload can be evenly distributed among the



(a) $\beta = 0.1$ (b) $\beta = 0.5$

Figure 5. Normalized energy consumption of a 64-core system with different block configurations for applications with limited parallelism.

employed cores). If $\frac{W}{DP} > T$, it is not possible to execute the workload within the given time T. Otherwise (i.e., $\frac{W}{DP} \leq T$), we can scale down the processing frequency of the employed cores to save energy. However, for different block configurations, the flexibility to choose a given number of cores is different, which will lead to different energy consumption.

For symmetric block configurations, we can easily determine the minimum number of blocks needed to provide the required number of cores. Moreover, for asymmetric buddy block configuration, as discussed in Section 3.1.2, it is always possible to select the appropriate blocks to provide a given number of cores. For applications with different degrees of parallelism (thus require different number of cores), we present in what follows the analysis results on the normalized energy consumption of the system with different block configuration.

Here, we consider a system with a 64-core processor. The system load is assumed to be $\delta = 0.25$. That is, the minimum degree of parallelism for the application should be 16, where 16 cores will be employed to run at f_{max} to complete the workload just in time. However, for different block configurations, the result energy consumption will be different as shown in Figure 5. For symmetric block configurations with each block having no more than 16 cores, we can select the blocks that have *exactly* 16 cores while powering off other blocks to save the energy consumption from the frequency-independent active power, which lead to the same normalized energy consumption for symm-4, symm-8 and symm-16 (which correspond to '4 cores/block', '8 cores/block' and '16 cores/block', respectively). Moreover, the asymmetric buddy configuration can also provide exactly 16 cores, which has the same normalized energy consumption as well. However, for

symmetric blocks with 32 cores on each block (symm-32), one block will be needed even for 16 cores and more energy from the frequency-independent active power will be consumed.

From the figure, we can also see that, as the degree of parallelism for the workload increases, the block configurations that can provide the exact required number of cores will have the same lowest normalized energy consumption. Such configurations include asymmetric buddy block configuration and symm-1 (i.e., individual block configuration, which is not shown in the figure). For systems with larger frequency-independent active power (e.g., $\beta = 0.5$) as shown in Figure 5(b), more energy savings can be achieved when compared to that of the common block configuration.

5 Evaluations and Discussions

For applications with limited degree of parallelism, extensive simulations has also been conducted to evaluate and validate the energy-efficiency of different block configurations. In the simulations, the adopted parameters for the system power model are shown in Table 5. We consider synthetic workload with varying degree of parallelism as well as execution trace data from a real-life application.

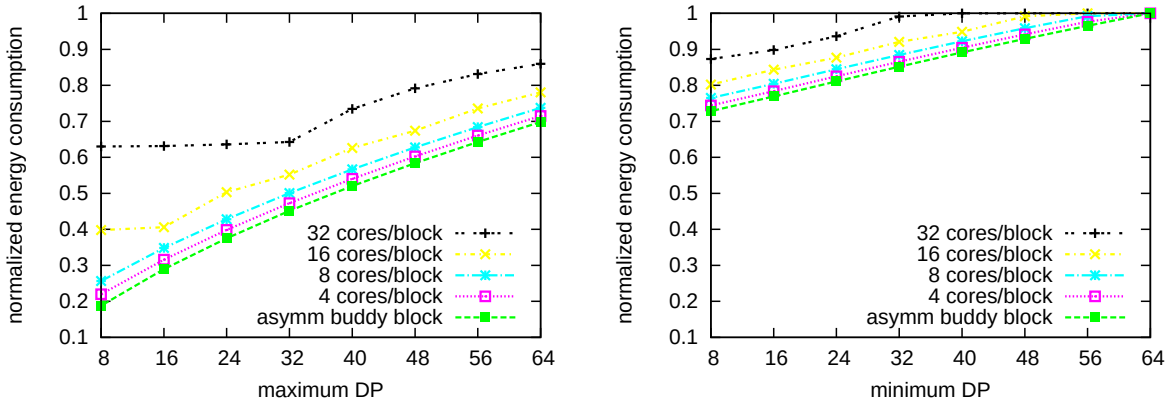
C_{ef}	m	f_{max}	Speed levels	P_s
1	3	1	$0.2, 0.4, 0.6, 0.8, 1.0$	0.01

Table 1. Parameters for Power Model

In the simulations, for a given workload under any block configuration, we select the optimal number of blocks and appropriate scaled frequency that lead to the minimum energy consumption. Moreover, the same as in the analysis, normalized energy consumption is reported and the energy consumption under the common block configuration is used as the baseline.

5.1 Synthetic Workload

To obtain a sequence of synthetic workload, we define DP_{min} and DP_{max} as the minimum and maximum degrees of parallelism, respectively. We assume that the workload will be executed on a 64-core system and the time period is $T = 1000$ time units. For the workload within each time period, its degree of parallelism DP is first randomly generated between DP_{min} and DP_{max}. Then, the workload is generated within the range of $[1, DP \cdot T]$ following the uniform distribution. That is, on average, the scaled frequency of the employed cores will be $f = 0.5$. For each point in the results, we generate the workload for $1,000,000$ time periods and the average result is reported.



(a) $DP_{min} = 1$ (b) $DP_{max} = 64$

Figure 6. Normalized energy consumption of a 64-core system with different block configurations for synthetic workload; $\beta = 0.5$.

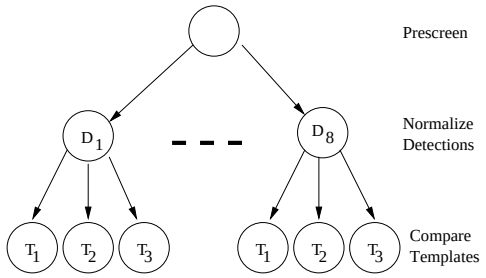
With fixed $DP_{min} = 1$, Figure 6(a) first shows the energy efficiency of different block configurations for varying DP_{max}. Here, we have $\beta = 0.5$. Similar results are obtained for $\beta = 0.1$, which is not shown due to space limitation. From the figure, we can see that, when the parallelism of workload is very limited (i.e., smaller values of DP_{max}), the energy savings for configurations with larger blocks (e.g., 32 cores per block) is rather limited. The reason comes from the fact that at least one block will be select, which limits the energy savings from the frequency-independent active power. However, for configurations with smaller blocks (e.g., 4 cores per block), compared to that of the common block, significant energy savings can be obtained as the unused blocks can be effectively powered off to save the energy consumption from the frequency-independent active power. As PD_{max} becomes larger, the energy savings decrease as more cores will be needed to process the workload and number of blocks that can be powered off for energy savings becomes small. Moreover, due to its flexibility to support various workload, the asymmetric buddy configuration can lead to the maximum energy savings, which is essentially the same as that of the individual block configuration (which is not shown in the figure).

With fixed $DP_{max} = 64$, Figure 6(b) further shows the energy efficiency of different block configurations for varying DP_{min}. Following the same reasoning, as the degree of parallelism increases, less energy can be saved for different block configurations. When $DP_{min} = 64$, the workload always has the degree of parallelism as 64 and all cores will be utilized, which results in the same amount of energy consumption for all block configurations.

5.2 Automated Target Recognition (ATR)

In this section, we use the trace data from automated target recognition (ATR) to show the energy efficiency of different block configurations. ATR searches regions of interest (ROI) in an image frame and tries to match specific templates with each ROI. The dependence graph for ATR is shown in Figure 7a. For military systems (e.g., missiles), ATR is widely used and usually requires multiple processing units to obtain real-time processing [42].

The trace data is gathered by instrumenting the ATR application to record their execution time for each parallel section, which runs on a Pentium-III $500MHz$ with $128MB$ memory. Figure 7b shows the run time information about the tasks in ATR for processing 180 consecutive frames on our platform. Here, we assume that ATR can process up to eight (8) ROIs in one frame and that each ROI is compared with three (3) different templates. Therefore, the maximum degree of parallelism in the template comparison phase will be 24. We use a 32-core system to examine the energy efficiency of different block configuration for the ATR application.



a. Dependence Graph of ATR

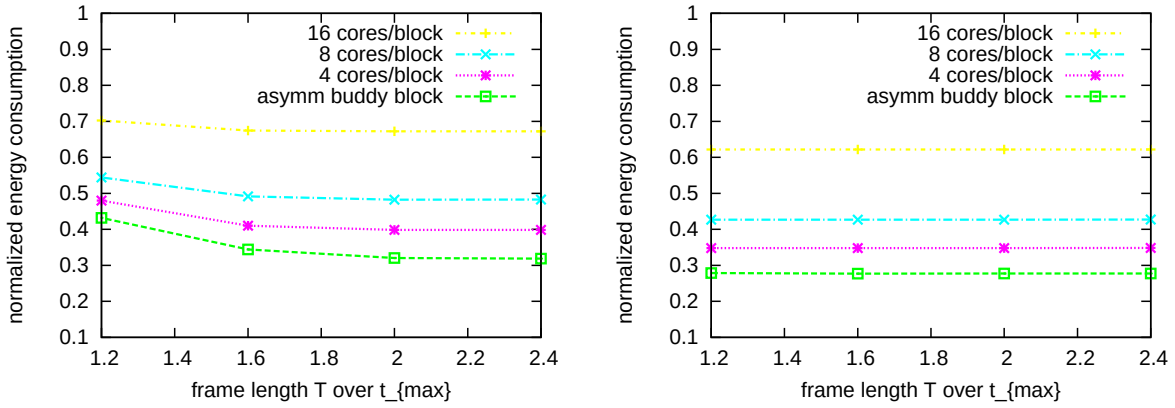
	min(μs)	max(μs)
Prescreen	1146	1299
Norm. Detection	429	748
Template 1	466	574
Template 2	466	520
Template 3	467	504

b. Execution Time for Tasks in ATR

Figure 7. The Dependence Graph of ATR. Assuming up to 8 detections in one frames and 3 templates.

Note that, the maximum processing time of one image frame can take up to $t_{max} = 2.621ms$. In the simulations, we assume that the time period T to process one frame varies from $1.2 \cdot t_{max}$ to $2.4 \cdot t_{max}$, which allows the cores to get different scaled frequencies. Within each frame, all employed cores are scaled down to the frequency of $f = \frac{t_{max}}{T}$. Note that, uniformly scaling down the execution of all phases with different parallelism is not optimal for energy savings [13]. However, exploring the optimal slack/time allocation to the phases with different parallelism for maximizing energy savings is orthogonal to the energy efficiency evaluation of different block configurations.

Figure 8 shows the simulation results based on the trace data for ATR. The results further confirm that



(a) $\beta = 0.1$ (b) $\beta = 0.5$

Figure 8. Normalized energy consumption for ATR on a 32-core system with different block configurations.

the configurations with smaller blocks can achieve better energy efficiency. Moreover, the asymmetric buddy block configuration can achieve the same level of energy efficiency as that of the individual block configuration. Better energy savings can be obtained for larger frequency-independent active power. In addition, the energy performance of different block configurations is rather stable for different amount of available static slack.

6 Conclusion

Energy management has become an important research area in the last decade. As an energy efficient architecture, multicore has been widely adopted. However, with the number of cores on a single chip continuing to increase, it has been a grand challenge to effectively manage the energy efficiency of multicore-based systems. In this paper, based on *voltage island* and *dynamic voltage and frequency scaling (DVFS)* techniques, we investigate the energy efficiency of *block-partitioned* multicore processors, where cores are grouped into *blocks* and each block has a DVFS-enabled power supply. Depending on the number of cores on each block, we study both *symmetric* and *asymmetric* block configurations, which contain the *same* and *different* number of cores on each block, respectively. In particular, a *buddy-asymmetric* block configuration is studied for its flexibility to efficiently support different system workloads. The number of required blocks (i.e., power supplies) in buddy-asymmetric block configuration is *logarithmically* related to the number of cores on the chip.

For such block-partitioned multicore systems, we develop a *system-level power model* that can effectively support various levels of power management techniques and derive both block- and system-level *energy-efficient frequencies*. Based on the power model, we prove that, for *embarrassingly parallel* applications, having all cores on a single block can achieve the same energy savings as that of the individual block configuration (where each core forms a single block and has its own power supply). However, for applications with limited degrees of parallelism, we show the superiority of the *buddy-asymmetric* block configuration, in that it can achieve the same amount of energy savings as that of the individual block configuration. The energy efficiency of block-partitioned multicore systems is further evaluated through extensive simulations with both synthetic as well as real life applications.

References

[1] Tarek A. AlEnawy and Hakan Aydin. Energy-aware task allocation for rate monotonic scheduling. In *RTAS '05: Proceedings of the 11th IEEE Real Time on Embedded Technology and Applications Symposium*, pages 213–223, 2005.

[2] James H. Anderson and Sanjoy K. Baruah. Energy-efficient synthesis of periodic task systems upon identical multiprocessor platforms. In *ICDCS '04: Proceedings of the 24th International Conference on Distributed Computing Systems (ICDCS'04)*, pages 428–435, 2004.

[3] H. Aydin, V. Devadas, and D. Zhu. System-level energy management for periodic real-time tasks. In *Proc. of The* 27th *IEEE Real-Time Systems Symposium (RTSS)*, Dec. 2006.

[4] H. Aydin, R. Melhem, D. Mossé, and P. Mejia-Alvarez. Dynamic and aggressive scheduling techniques for power-aware real-time systems. In *Proc. of The* 22th *IEEE Real-Time Systems Symposium*, Dec. 2001.

[5] H. Aydin and Q. Yang. Energy-aware partitioning for multiprocessor real-time systems. In *Proc. of the* 17th *International Parallel and Distributed Processing Symposiu m (IPDPS), Workshop on Parallel and Distributed Real-Time Systems (WPDRTS)*, 2003.

[6] D. Bautista, J. Sahuquillo, H. Hassan, S. Petit, and J. Duato. A simple power-aware scheduling for multicore systems when running real-time applications. In *IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 1–7, 2008.

[7] P. Bohrer, E. N. Elnozahy, T. Keller, M. Kistler, C. Lefurgy, C. McDowell, and R. Rajamony. *The case for power management in web servers*, chapter 1. Power Aware Computing. Plenum/Kluwer Publishers, 2002.

[8] S. Borkar. Thousand core chips: a technology perspective. In *Proc. of the 44th annual Design Automation Conference (DAC)*, pages 746–749, 2007.

[9] T. D. Burd and R. W. Brodersen. Energy efficient cmos microprocessor design. In *Proc. of The HICSS Conference*, Jan. 1995.

[10] Jian-Jia Chen. Multiprocessor energy-efficient scheduling for real-time tasks with different power characteristics. In *ICPP '05: Proceedings of the 2005 International Conference on Parallel Processing*, pages 13–20, 2005.

[11] Jian-Jia Chen, Heng-Ruey Hsu, Kai-Hsiang Chuang, Chia-Lin Yang, Ai-Chun Pang, and Tei-Wei Kuo. Multiprocessor energy-efficient scheduling with task migration considerations. In *ECRTS '04: Proceedings of the 16th Euromicro Conference on Real-Time Systems*, pages 101–108, 2004.

[12] Jian-Jia Chen, Heng-Ruey Hsu, and Tei-Wei Kuo. Leakage-aware energy-efficient scheduling of real-time tasks in multiprocessor systems. In *RTAS '06: Proceedings of the 12th IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 408–417, 2006.

[13] Sangyeun Cho and Rami G. Melhem. On the interplay of parallelization, program performance, and energy consumption. *IEEE Transactions on Parallel and Distributed Systems*, 21(3):342–353, 2010.

[14] K. Choi, W. Lee, R. Soma, and M. Pedram. Dynamic voltage and frequency scaling under a precise energy model considering variable and fixed components of the system power dissipation. In *Proc. of International Conference on Computer Aided Design (ICCAD)*, pages 29–34, Nov. 2004.

[15] 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 the International Conference on Computer-Aided Design (ICCAD)*, pages 195–202, 2002.

[16] J. Donald and M. Martonosi. Techniques for multicore thermal management: Classification and new exploration. In *ISCA '06: Proceedings of the 33rd International Symposium on Computer Architecture*, pages 78–88, Washington, DC, USA, Jun. 2006. IEEE Computer Society.

[17] J. Dorsey, S. Searles, M. Ciraula, S. Johnson, N. Bujanos, D. Wu, M. Braganza, S. Meyers, E. Fang, and R. Kumar. An integrated quad-core opteron processor. In *IEEE International Solid-State Circuits Conference*, pages 102–103, Feb. 2007.

[18] X. Fan, C. Ellis, and A. Lebeck. The synergy between power-aware memory systems and processor voltage. In *Proc. of the Workshop on Power-Aware Computing Systems*, 2003.

[19] I. Foster. *Design and Building Parallel Programs*, chapter 1.4.4. Addison-Wesley, 1995.

[20] P. Hazucha, G. Schrom, Jaehong Hahn, B. A. Bloechel, P. Hack, G. E. Dermer, S. Narendra, D. Gardner, T. Karnik, V. De, and S. Borkar. A 233-mhz 80%-87% efficient four-phase dc-dc converter utilizing air-core inductors on package. *Solid-State Circuits, IEEE Journal of*, 40(4):838–845, 2005.

[21] Sebastian Herbert and Diana Marculescu. Analysis of dynamic voltage/frequency scaling in chip-multiprocessors. In *Proceedings of the international symposium on Low power electronics and design (ISLPED)*, pages 38–43, 2007.

[22] M. D. Hill and M. R. Marty. Amdahl's law in the multicore era. *IEEE Computer*, 41(7):33–38, 2008.

[23] http://public.itrs.net. International technology roadmap for semiconductors. 2008. S. R. Corporation.

[24] http://www.amd.com/us en/Processors/ProductInformation/, 2008.

[25] http://www.intel.com/products/processor/core2quad/, 2008.

[26] J. Hu, Y. Shin, N. Dhanwaday, and R. Marculescu. Architecting voltage islands in core-based system-on-a-chip designs. In *Proc. of the Int'l symposium on Low power electronics and design (ISLPED)*, pages 180–185, 2004.

[27] Intel. Intel embedded quad-core xeon;http://www.intel.com/products/embedded/processors.htm, 2009.

[28] C. Isci, A. Buyuktosunoglu, C.-Y. Cher, P. Bose, and M. Martonosi. An analysis of efficient multi-core global power management policies: Maximizing performance for a given power budget. In *Proc. of the 39th Annual IEEE/ACM Int'l Symposium on Microarchitecture (MICRO)*, pages 347–358, 2006.

[29] T. Ishihara and H. Yauura. Voltage scheduling problem for dynamically variable voltage processors. In *Proc. of The 1998 International Symposium on Low Power Electronics and Design*, Aug. 1998.

[30] A. Iyer and D. Marculescu. Power efficiency of voltage scaling in multiple clock, multiple voltage cores. In *Proc. of the International Conference on Computer-Aided Design (ICCAD)*, pages 379–386, Nov. 2002.

[31] R. Jejurikar and R. Gupta. Dynamic voltage scaling for system wide energy minimization in real-time embedded systems. In *Proc. of the Int'l Symposium on Low Power Electronics and Design (ISLPED)*, pages 78–81, 2004.

[32] H. Kim, H. Hong, H.-S. Kim, J.-H. Ahn, and S. Kang. Total energy minimization of real-time tasks in an on-chip multiprocessor using dynamic voltage scaling efficiency metric. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 27(11):2088–2092, Nov. 2008.

[33] W. Kim, M.S. Gupta, G-Y. Wei, and D. Brooks. System level analysis of fast, per-core dvfs using on-chip switching regulators. In *Proceedings of the Intl. Symp. on High-Performance Computer Architecture (HPCA)*, pages 123–134, Feb. 2008.

[34] R. Kumar and G. Hinton. A family of 45nm ia processors. In *Proc. of the Int'l Solid-State Circuits Conference*, 2009.

[35] A. R. Lebeck, X. Fan, H. Zeng, and C. S. Ellis. Power aware page allocation. In *Proc. of the* 9th *International Conference on Architectural Support for Programming Languages and Operating Systems*, Nov. 2000.

[36] L. Leung and C. Tsui. Energy-aware synthesis of networks-on-chip implemented with voltage islands. In *Proc. of the 44th ACM/IEEE Design Automation Conference (DAC)*, pages 128–131, 2007.

[37] X. Li, Z. Li, F. David, P. Zhou, Y. Zhou, S. Adve, and S. Kumar. Performance directed energy management for main memory and disks. In *Proc. of the* 11th *Int'l Conference on Architectural Support for Programming Languages and Operating Systems*, 2004.

[38] L. Mosley. Power delivery challenges for multicore processors. In *Proc. of the CARTS USA*, 2008.

[39] C. Neau and K. Roy. Optimal body bias selection for leakage improvement and process compensation over different technology generations. In *Proc. of the International Symposium on Low Power Electronics and Design (ISLPED)*, pages 116–121, Aug. 2003.

[40] K. Olukotun, B. A. Nayfeh, L. Hammond, K. Wilson, and K. Chang. The case for a single-chip multiprocessor. In *Proc. of the Int'l Symp. Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 2–11, Oct. 1996.

[41] P. Pillai and K. G. Shin. Real-time dynamic voltage scaling for low-power embedded operating systems. In *Proc. of* 18th *ACM Symposium on Operating Systems Principles (SOSP'01)*, Oct. 2001.

[42] J. A. Ratches, C. P. Walters, R. G. Buser, and B. D. Guenther. Aided and automatic target recognition based upon sensory inputs from image forming systems. *IEEE Tran. on Pattern Analysis and Machine Intelligence*, 19(9):1004–1019, 1997.

[43] S. Saewong and R. Rajkumar. Practical voltage scaling for fixed-priority rt-systems. In *Proc. of the* 9th *IEEE Real-Time and Embedded Technology and Applications Symposium*, 2003.

[44] E. Seo, J. Jeong, S. Park, and J. Lee. Energy efficient scheduling of real-time tasks on multicore processors. *IEEE Trans. Parallel Distrib. Syst.*, 19(11):1540–1552, 2008.

[45] J. Shin, K. Tam, D. Huang, B. Petrick, H. Pham, C. Hwang, H. Li, A. Smith, T. Johnson, F. Schumacher, D. Greenhill, A. Leon, and A. Strong. A 40nm 16-core 128-thread cmt sparc soc processor. In *Proc. of the International Solid-State Circuits Conference (ISSCC)*, 2010.

[46] A. Silberschatz, P. B. Galvin, and G. Gagne. *Operating System Concepts*. 2002.

[47] A. Sinkar and N. S. Kim. Analyzing potential power reduction with adaptive voltage positioning optimized for multicore processors. In *Proc. of the 14th ACM/IEEE int'l symposium on Low power electronics and design*, pages 189–194, 2009.

[48] R. Springer, D. K. Lowenthal, B. Rountree, and V. W. Freeh. Minimizing execution time in mpi programs on an energy-constrained, power-scalable cluster. In *Proc. of the 11th ACM SIGPLAN symposium on Principles and practice of parallel programming (PPoPP)*, pages 230–238, 2006.

[49] V. Swaminathan and K. Chakrabarty. Pruning-based, energy-optimal, deterministic i/o device scheduling for hard real-time systems. *ACM Transactions on Embedded Computing Systems*, 4(1):141–167, Feb. 2005.

[50] S. Vangal, J. Howard, G. Ruhl, S. Dighe, 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.

[51] Chuan-Yue Yang, Jian-Jia Chen, and Tei-Wei Kuo. An approximation algorithm for energy-efficient scheduling on a chip multiprocessor. In *DATE '05: Proceedings of the conference on Design, Automation and Test in Europe*, pages 468–473, 2005.

[52] F. Yao, A. Demers, and S. Shenker. A scheduling model for reduced cpu energy. In *Proc. of The* 36th *Annual Symposium on Foundations of Computer Science*, Oct. 1995.

[53] 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.

[54] D. Zhu, R. Melhem, and D. Mossé. The effects of energy management on reliability in real-time embedded systems. In *Proc. of the Int'l Conf. on Computer Aidded Design*, 2004.