Ain Shams Engineering Journal (2017) 8, 445–456

Ain Shams University

Ain Shams Engineering Journal

www.elsevier.com/locate/asej
www.sciencedirect.com

ELECTRICAL ENGINEERING

# Dynamic power management techniques in multi-core architectures: A survey study

**Khaled M. Attia** *, **Mostafa A. El-Hosseini, Hesham A. Ali**

*Computers and Control Systems Engineering Department, Faculty of Engineering, Mansoura University, Mansoura, Egypt*

Received 10 May 2015; accepted 3 August 2015
Available online 1 October 2015

**KEYWORDS**

Chip multiprocessors;
Multi-core;
Power management

**Abstract** Multi-core processors support all modern electronic devices nowadays. However, power management is one of the most critical issues in the design of today's microprocessors. The goal of power management is to maximize performance within a given power budget. Power management techniques must balance between the demanding needs for higher performance/throughput and the impact of aggressive power consumption and negative thermal effects. Many techniques have been proposed in this area, and some of them have been implemented such as the well-known DVFS technique which is used in nearly all modern microprocessors. This paper explores the concepts of multi-core, trending research areas in the field of multi-core processors and then concentrates on power management issues in multi-core architectures. The main objective of this paper is to survey and discuss the current power management techniques. Moreover, it proposes a new technique for power management in multi-core processors based on that survey.


## 1. Introduction

The evolution of multi-core processors led to the evolution of many research areas. Before the appearance of multi-core processors, the speed of microprocessors increased exponentially over time. More speed requires more transistors. Moore [1] observed that the number of transistors doubles approximately every two years. With the rapid increase in speed, the number of transistors in processors increased in a way that it can't scale to Moore's law anymore as an extremely huge number of transistors switching at very high frequencies means extremely high power consumption. Also, the need for parallelism increased and the instruction level parallelism [2] was not sufficient to provide the demanding parallel applications. So the concept of multi-core was introduced by Olukotun et al. [3], to design more simple cores on a single chip rather than designing a huge complex one. Now all modern microprocessor designs are implemented in a multi-core fashion. Multi-core advantages can be summarized as follows:

- A chip multiprocessor consists of simple-to-design cores
- Simple design leads to more power efficiency
- High system performance in parallel applications where many threads need to run simultaneously

* Corresponding author. Mobile: +20 1000736160.
E-mail addresses: khaled.m.attia@mans.edu.eg (K.M. Attia), melhosseini@mans.edu.eg (M.A. El-Hosseini), h_arafat_ali@mans.edu.eg (H.A. Ali).
Peer review under responsibility of Ain Shams University.

ELSEVIER Production and hosting by Elsevier

http://dx.doi.org/10.1016/j.asej.2015.08.010

Introducing multi-core processors aroused many related areas of research. Dividing code into threads, each can run independently is very important to make use of the power of the multi-core approach. However, not all code can be divided in such a manner. That issue was described by Amdhal in [4] which concludes that maximum speedup is limited by the serial part and that is called the serial bottleneck. Serialized code reduces performance expected by the processor; it also wastes lots of energy. Also, the parallel portion of the code is not completely parallel because of many reasons such as synchronization overhead, load imbalance and resources contention among cores. The serial bottleneck research led to the evolution of asymmetric multi-core processors [5].

The concept of asymmetric multi-core processors implies that the design would include one large core and many small cores. The serial part of the code will be accelerated by moving it to the large core and the parallel part is executed on the small cores. This accelerates both the serial part by using the large core and the parallel part as it will be executed simultaneously on the small cores and the large core to achieve high throughput. Using asymmetric cores can be more energy efficient too. In [5] Mark et al. described how asymmetry can be achieved. They divided it into static and dynamic methods. For static methods, cores may be designed at different frequencies or a more complex core with completely different micro-architecture may be designed. In dynamic methods, frequencies can be boosted dynamically on demand or small cores may be combined to form a dynamic large core and this is described in detail in [6]. Other research topics related to multi-core processors that emerged include the following: power management, memory hierarchies in multi-core processors, the design of interconnection networks in multi-core processors, heterogeneous computing in multi-core processors, reliability issues in multi-core processors and parallel programming techniques. In power management, the main objective is to reach the maximum performance of the processors without exceeding a given total power budget for the chip. There has been lots of research on power management in chip multiprocessors. Here we are going to discuss most of those techniques [7] and some modern works that try to optimize the efficiency of these techniques. In this paper we examine all popular techniques in detail and how they work to minimize performance losses while saving power. We investigate the suitable technique for each case (workloads, power budget available, critical systems) and how to make these techniques even more suitable for their cases.

This paper makes the following contributions:

- Listing almost all the used techniques for power management in multi-core processors, discussing them in terms of advantages and disadvantages (performance loss, power saving, suitable cases) and providing a comparison between them.
- Examining some of the improvements added to each of these techniques to make them even better.
- Proposing a new adaptive control mechanism for power management in asymmetric multi-core processors.
- Suggesting further research to be done in some of the investigated techniques/scenarios.

The rest of this paper is organized as follow: Section 2 introduces the historical improvements in the microprocessor design, explains how we have reached the multicore era and mentions the main issues associated with multicore processors. Section 3 starts to focus on the power management issue, showing the importance of handling such a problem and providing a proper problem formulation. It continues to explain almost all the current techniques used in the power management field in modern processors, showing the advantages and disadvantages of each one and the research done to try to solve each shortage. Section 4 proposes a new mechanism for power management in asymmetric multicore processors. Finally, we conclude in Section 5 by reviewing the most important ideas that were presented in the paper.

## 2. Background

The performance of microprocessors has increased exponentially over years. Techniques have been devised to achieve parallelism, starting from pipelining, passing by super-scalar architectures and finally the chip multiprocessors or multicore processors. Here we shed light on the various levels of parallelism and how consequent technologies tried to exploit each level.

### 2.1. Levels of parallelism

Each one of these techniques exploits some levels of parallelism which can be listed as follows:

(1) Instruction level parallelism

In this level, architectures make use of independent instructions (the operands of one instruction do not depend on the result of another one) that exist in the instructions streams to execute them concurrently.

(2) Basic Block level

A block can be considered a set of instructions that end with a branch. Modern architectures were able to exploit this level of parallelism among basic blocks with the help of advanced branch predictors

(3) Loop iterations

Some types of loops work on independent data in each iteration of the loop. So, it is possible in these loops to run different iterations concurrently in superscalar architectures for example.

(4) Tasks

A task signifies an independent function extracted from one application. It can also be called a thread. Software developers have to divide their code into independent threads to make use of this level of parallelism in multiprocessors systems, where each thread can run independently on a dedicated core.

### 2.2. Advances in processor microarchitecture

Over the years, there have been many trials to exploit better parallelism as shown in Fig. 1; advances in architecture can be viewed as follows:

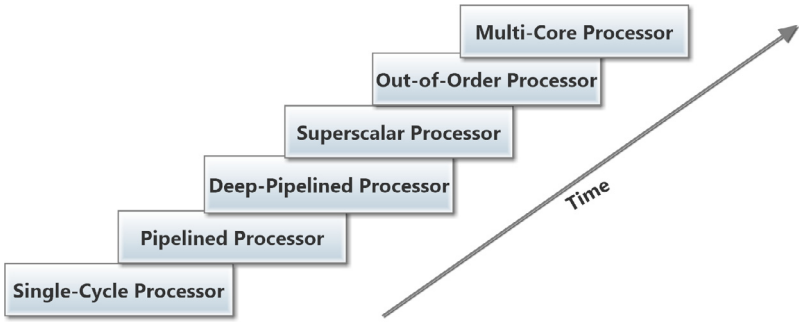


**Figure 1** Advances in microprocessor design over time.

*2.2.1. Single-cycle processor*

This technique was used in very early microprocessors. The key concept is that the whole instruction is executed at once in one clock cycle. Whenever an instruction has started to execute, all other instructions in the instruction stream have to wait until it fully finishes its execution. Of course, some instructions take lots of execution/waiting time which affects the execution of other instructions and degrades overall system performance (see Fig. 2a).

*2.2.2. Pipelining*

Instead of executing the whole instruction at once, pipelining divides the single-cycle processor into many stages; in each stage, a portion of the instruction is executed concurrently with another portion of another instruction. For example, if we have a three-stage pipelined processor that means the single-cycle processor is divided into three stages, let them be, for example, FETCH OPERANDS, DECODE and EXECUTE. Then, we can execute three instructions simultaneously. At clock cycle 3, the first instruction will be in the EXECUTE stage, while the second instruction will be in the DECODE stage and the third instruction would be in the FETCH OPERANDS stage. That obviously diminishes the drawback of long wait times in long instructions (see Fig. 2b). It exploits the instruction level parallelism where multiple instructions can be executed concurrently. On the other hand, pipelining introduces logic overhead in each stage of the pipeline. Also, some data dependency hazards occur when two dependent instructions are executed concurrently. However, many techniques were proposed to overcome such hazards.

*2.2.3. Deep pipelining*

The idea of deep pipelining [8] is to increase the number of pipeline stages significantly. It is obvious from the discussion about the pipelined processor that the more stages you add, the faster execution you get. That is of course valid to a certain extent. Common pipelines have up to 20 stages. The number of stages is greatly limited by many factors such as the existing hazards and the logic overhead. As we mentioned before in the pipelined processor, many techniques has been devised to overcome the data dependency problem. These techniques include, but not limited to, forwarding, stalling and register renaming.

*2.2.4. Super scalar processor*

One of the main bottlenecks in the pipelined processor design is that however many instructions can run at different phases in the same time, the pipeline can only be initiated with one instruction. A superscalar processor is the one that contains multiple copies of the whole datapath (including the ALU) which makes it possible to issue as many instructions as the

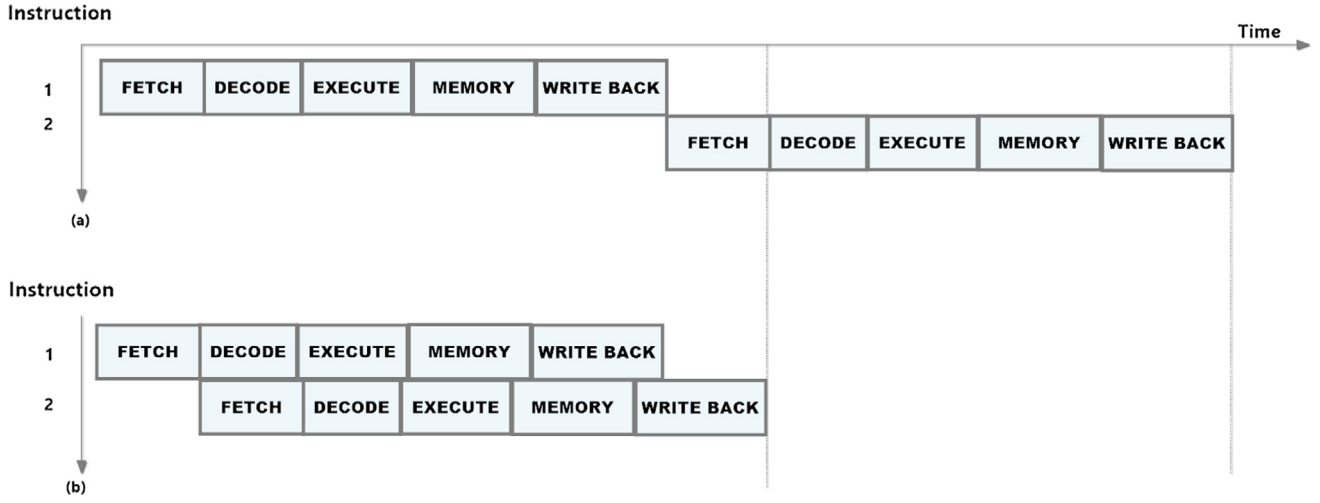


**Figure 2** Difference between (a) single-cycle processor and (b) pipelined processor.

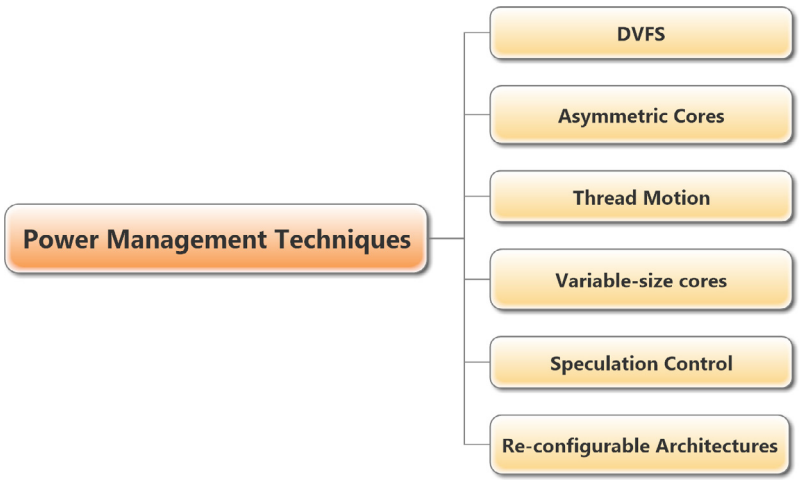


**Figure 3** A tree diagram of the most common power management techniques.

number of the copies is. Each instruction runs almost independently as it has its own dedicated datapath. Superscalar processors concepts have always been combined with pipelined processors concepts to introduce the pipelined superscalar processor which has been commonly used in the 1990s and early 2000s. The basic operation of a superscalar processor includes fetching and decoding a stream of instructions, branch prediction, figuring out whether there are any dependencies among instructions and finally the distribution of instructions to different functional units to be issued [9]. It provides great enhancements in the overall performance/throughput of the system. However, not many instructions can run at the same time because of the dependency problem explained in the pipelined processors. Moreover, the number of issued instructions is limited. Also, it introduces lots of hardware overhead meaning larger areas and more power consumption.

#### *2.2.5. OoO (Out-of-Order) processors*

OoO processors look ahead across instruction window to find independent instructions that can be executed immediately. This means, instructions are not executed in the order they were written in. Once the operands of an instruction are available, the instruction is executed regardless of the sequence of the program. OoO processors solve the problem of dependencies introduced in the pipelined superscalar processor. However, they introduce additional hardware overhead and energy consumption for speculation.

#### *2.2.6. Chip multiprocessors*

Chip multiprocessors or multi-core processors exploit thread level parallelism efficiently. A process is a program currently in execution. Each process consists of one or more threads. For example, a server application would have at least two threads, one for listening to incoming connections and another one for outgoing connections. No thread has to wait for the other to finish as they execute concurrently. In traditional uniprocessor systems, multi-threading is not well utilized. The uniprocessor provides an illusion that threads run concurrently but in fact a fast switch is done between threads of the same process (which is way faster than switching between processes). Multi-core architectures appeared to extract as much parallelism as possible from the thread level parallelism. In a multi-core processor, each thread runs independently on a dedicated core (real parallelism). Hence, great enhancements are made to the overall throughput of the system. However, many issues came up such as the problem of designing the appropriate memory hierarchy, the data locality problem, the design of interconnection networks, maintaining the reliability and validity of the processor and power management. In this paper, we are discussing the power management issue in multi-core processors and the techniques proposed and used in that area.

## 3. Power management techniques

Power management has become a major issue in the design of multi-core chips. There are many negative effects that result from increasing power consumption such as unstable thermal properties of the die and hence affecting the system performance which makes power consumption issue sometimes more important than speed. An important observation is that threads running on different cores do not need the same power all time to execute at high performance. There are some waiting times due to memory read/write operations for example which require saving unnecessary processing power. So, to achieve a good balance between scalar performance/throughput performance and power it is essentially required to dynamically vary the amount of power used for processing according to temporal analysis of the code needs.

Developed power management techniques can be classified into two main categories: reactive and predictive. In reactive techniques, the technique reacts to performance changes in the workload. In other words, a workload may initially have states that need high performance, others of I/O waits and low performance. When the state of the workload changes, the technique reacts to that change accordingly. However, there might be some lag between workload phase changes and power adaptation changes which may lead to states of either in-efficient energy consumption or performance degradation. On the other hand, predictive techniques, for example [10], overcome this issue. Those techniques predict phase changes in the workload before they happen, and hence act immediately before a program phase changes. That leads to optimal energy-saving and performance results. However,

there is no workload that can be fully predicted, so reactive techniques are used for portions that cannot be predicted (which is usually more than 60% of the entire workload). So, reactive techniques are inevitable to use and consequently we concentrate in this study on those techniques. Here, we are examining some of the dynamic techniques as shown in Fig. 3 to achieve the best level of power management in multi-core processors. We also discuss some issues related to each of these techniques and how previous research attempted to handle these issues.

***Problem formulation*** can be viewed as follows: all techniques assume there is an on-chip or on-board hardware controller for power management which contains all the hardware and circuitry required for performing its job. The controller is always supported by some firmware and software that give directives for implementing the specific technique or algorithm. Fig. 4 shows a high-level view of the power management process assuming a global on-chip power management controller. The system-level controller directs the global on-chip controller toward a specific power budget. The global on-chip controller monitors power-performance statistics from all cores and dependently takes the required action. That action depends on the algorithm/technique used (for example, change voltage as in DVFS, cut-off power of specific portions as in power-gating techniques).

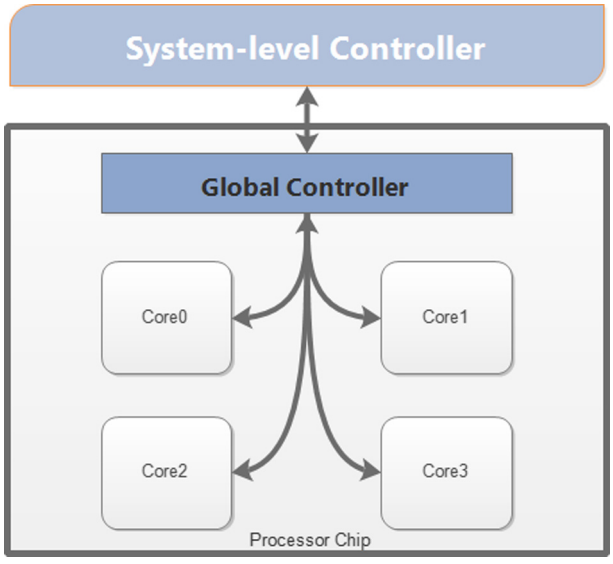


**Figure 4** High-level view of dynamic power management problem.

Techniques can be evaluated in terms of power efficiency. A common metric for the evaluation of power efficiency is energy per instruction (EPI in Watt/MIPS or Joule/Instruction). Other metrics such as energy delay product (EDP), which was initially proposed by Horowitz et al. [11], and $ED^2P$ are used also in latency performance architectures as they assign a weight to the amount of time needed for an instruction to be processed. Obviously, techniques that achieve lower EPI are more energy-efficient. The main objective of almost all techniques is achieving high Instruction per Cycle (IPC) while maintaining low EPI. That balance is the main concern of almost all the research done on power management in microprocessors.

The power management process can be viewed as a feedback closed-loop control system. Power budget is considered the desired input coming from the system-level control system. And there is an on-chip or on-board controller that adjusts some parameters (such as voltage and frequency) based on the monitoring process (feedback) coming from the individual cores of the chip in a closed-loop and so on. Monitoring power consumption has been a hot research topic for many years. For any powersaving mechanism, it needs to monitor consumed power to guide its decision. Mainly, Performance Monitoring Counters (PMCs) are used to obtain power models. Examples of research done on that point are included [12–15] as examples. This representation leads us to another point which is as follows: as power management control systems can be viewed as feedback control system. That implies that they have regions of instability which require in turn providing guarding/security mechanism for power management which is out of the scope of this paper. Fig. 5 illustrates that concept.

### 3.1. Dynamic voltage and frequency scaling

#### 3.1.1. Basic concept

The idea of using dynamic voltage and frequency scaling in power management in microprocessor systems was originally invented by Weiser et al. in 1996 [16]. The power consumption is mainly governed by the following equation:

$$P = CV^2F \tag{1}$$

where $P$ is the power, $C$ is the switching capacitance, $V$ is the supplied voltage and $F$ is the working frequency.

It's very obvious that we can control the amount of consumed power by simply adjusting voltage–frequency pairs. This method has been widely used to achieve different Energy per Instruction (EPI) ratios. It has been commercially introduced under many names such as Intel's SpeedStep technology, AMD's PowerNow! The main idea is to adjust

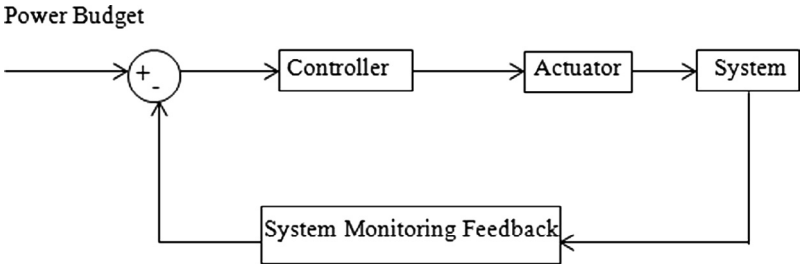


**Figure 5** Feedback closed-loop representation of power management control system.

voltage–frequency pairs within a set of discrete, predefined pairs to achieve the required power/performance level. In other words, for heavy parallel workloads, many cores run at low voltage–frequency pair. However, for scalar workloads which include a big portion of serial code, it is reasonable to run few cores, and boost their frequency to adapt to the required task. Fig. 6 explains this concept. DVFS management system for a dual-core processor can be viewed graphically as in figure.

The system level controller directs the global on-chip or on-board controller with the desired power budget. The global controller monitors voltage, frequency and IPC (power usage) of each core. Depending on these parameters, the controller actuates voltage and/or frequency as required. The same concept is applied through all power management techniques as previously explained, and the main difference lies in the algorithm itself. Machine learning algorithms (especially reinforcement learning) have been recently used to perform DVFS [17–21]. Using these techniques led to even better results on both the performance and energy saving metrics.

DVFS has not been only used in general purpose applications. It is widely applied to almost all modern processors in embedded systems [22]. Also, it can be used in real-time applications. For example in [23], DVFS is used along with a checkpointing technique for consumed power reduction in reliability-guaranteed real-time applications. That study proves that with the use of backward fault recovery technique, DVFS can achieve highest system reliability while consuming minimal amount of energy.

### 3.1.2. Determination of the suitable voltage-frequency setting

As mentioned, the default on demand linux governor chooses voltage–frequency pairs from a set of predefined, discrete values. That's not very power efficient as the required voltage–frequency pair may be not exactly one of the predefined values. Kamga et al. [24] proposed an approach for precise determination of the required frequency for current workload. Kamga suggests a method to precisely determine the required frequency based on the high and low threshold and number of occurrences of each of them. The method ends up with the required frequency to be

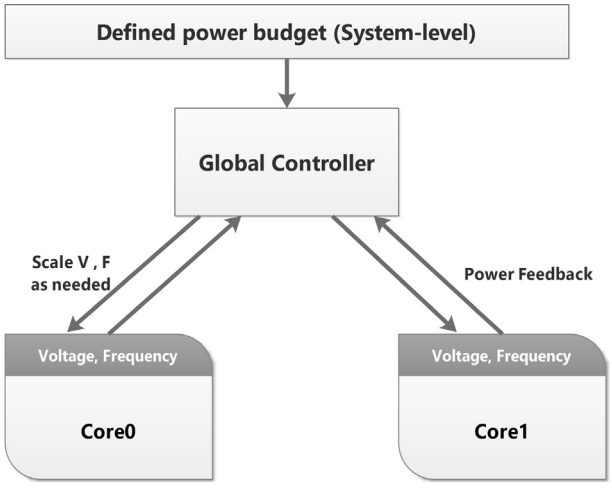


**Figure 6** High-level graphical view of DVFS applied to a dual-core processor.

$$f_{host} = \frac{(f_{high} \times t_{high}) + (f_{low} \times t_{low})}{t_{high} + t_{low}} \quad (2)$$

where $f_{host}$ is the required frequency, $f_{high}$ is the high threshold frequency, $t_{high}$ is the number of occurrences of that frequency and similarly for the low threshold.

### 3.1.3. DVFS levels of granularity

DVFS can be applied either per chip or per core. Applying DVFS per core introduces much flexibility as each core would have its own voltage–frequency pair. However, that incurs at the expense of a large number of on-chip voltage regulators. On the other hand, applying DVFS on the chip level reduces that expense but limits flexibility as the same voltage would be applied to all cores regardless of the special needs of each individual core. It is extremely difficult to determine a single Voltage–Frequency setting that satisfies all cores needs simultaneously. In [25] Kolpe et al. proposed an intermediate technique called "clustered DVFS" which clusters the cores into different DVFS domains and implements DVFS on a per-cluster basis. The algorithm of this approach can be summarized in three main steps: (1) find the optimal voltage/frequency setting for each core individually, (2) find similarities between cores (for example, the cores with similar voltage/frequency setting from the first step over a certain number of clock cycles are similar) and cluster similar cores together and finally (3) evaluate the solution by finding the optimal voltage/frequency setting for each cluster and compare it with the actual setting of the cluster. This approach proved to have significant results, compared to per-core DVFS but it returns diminishing results when the number of clusters increases.

### 3.1.4. Time to vary voltage and frequency

Scaling voltage and frequency takes some latency to wait for the voltage/frequency reach the desired level. However, frequency scaling is much faster than voltage scaling. Consequently, the processor can be in dangerous states where the current voltage cannot support the frequency. In these cases, hard faults would occur and cause the CPU to stop operating. Fig. 7 shows the relationship between voltage and frequency during DVFS. A boundary can be drawn to divide the voltage–frequency space into three areas: (1) area above the boundary which contains dangerous power states because the voltage cannot support the frequency, (2) area under the boundary is not energy efficient, and (3) the boundary which contains power-safe states. For example, if we want to scale from s0 to s3, the frequency will scale faster which leads to reaching the dangerous state s2.

The traditional method to overcome this issue was to scale voltage first and stall the running application until the voltage scaling is done, then scale frequency. It is very obvious that this method introduces lots of latency. Some research has been done to address the latency resulted from applying DVFS. Lai et al. [26] proposed an algorithm that reduces latency by avoiding unnecessary aggressive power states transitions. Also, Lai et al. [27] proposed the Retroactive Frequency Scaling (RFS) technique which suggests not stalling the execution of the application during voltage scaling, but running it at the previous frequency setting until voltage scaling is done. Although that eliminates much of the latency, it comes to the cost of running at power inefficient state during voltage scaling.

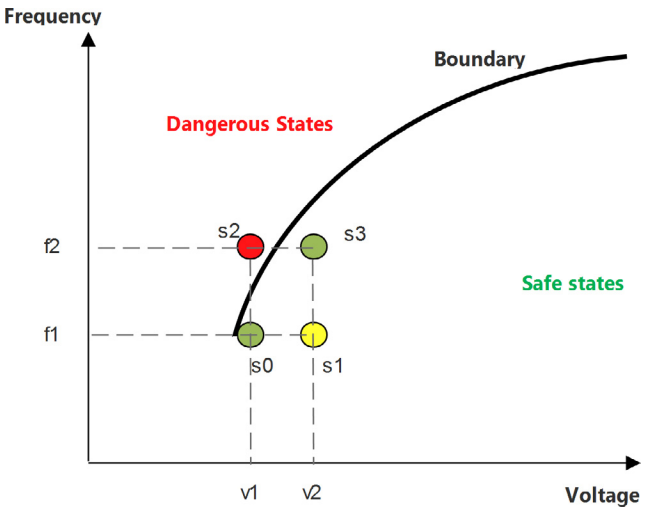


**Figure 7** Relationship between voltage and frequency during dynamic scaling.

### 3.1.5. Limitations of DVFS

We can summarize the limitations of DVFS from the above discussion as follows:

*3.1.5.1. DVFS domain.* Based on the level to which DVFS is applied, some issues are represented. If DVFS is applied per-core, that would mean higher cost and less scalability when applying it to a processor with a large number of cores. When DVFS is applied per-chip, it would ignore the individual needs of each core separately. Also, in the clustered solution, it would return diminishing results when the number of clusters increases to a certain extent.

*3.1.5.2. Large state transition delay.* As we mentioned, voltage change takes a while to complete. That makes the transition from one power state to another slow and sometimes dangerous as showed in the hard faults issue. Slow transition is a big problem as the required setting may also change in the period of transition which makes it useless.

## 3.2. Asymmetric cores

### 3.2.1. Basic concept

The concept of designing a heterogeneous, single-ISA multi-core processor [5] in which each core differs in performance and power consumption has been very useful in the field of power management. As previously mentioned, two types of cores are designed: large (or complex) cores which is usually out-of-order, superscalar, deep-pipelined core to support heavily scalar workloads and small (or simple) core which, on the other side, is usually in-order, scalar, short-pipelined core that participates in parallel workloads.

One of the most popular commercial implementations of asymmetric chip multiprocessor is ARM's big.LITTLE architecture [28]. In 2011, ARM announced that in this architecture, out-of-order superscalar Cortex-A15 cores are combined with the simple, in-order Cortex-A7 cores as shown in Fig. 8. They both implement the ARM v7A. The Cortex-A15 support for high-performance, energy-hungry scalar workloads while the Cortex-A7 support the parallel workloads. Later in 2012, ARM announced the Cortex-A53 and Cortex-A57 that implement the ARMv8-A instruction set and are compatible to be combined together in a big.LITTLE architecture. Later in 2014, ARM announced the Cortex-A17 which can be combined with Cortex-A12 in the big.LITTLE architecture.

Samsung implemented ARM's big.LITTLE architecture in its octa-core processor Exynos 5 [29] which consists of 8 cores: 4 small, power-efficient Cortex-A7 cores and 4 big, high-performance Cortex-A15 cores. Also, NVidia implemented its own heterogeneous 5-core multiprocessor Tegra3 [30] which has 4 large, high-performance cores and one small, power-efficient core. Asymmetric cores have been widely used recently in most modern mobile and embedded systems [31] to reduce power consumption of these devices. Using DVFS in asymmetric cores is very popular and can be used in both single-ISA and multiple-ISA heterogeneous architectures [32]. It is expected to see more architectures and implementations of the heterogeneous multi-core architecture as they have proven usability, accommodating to various, special purpose applications and of course power efficiency needed for almost all modern mobile and embedded devices.

### 3.2.2. Critical sections and thread scheduling

There are many issues associated with asymmetric cores. One important issue is thread scheduling, in other words, the problem of assigning the thread to the proper core that would be most power/performance efficient for its workload. Different hardware platforms require different scheduling/resource allocation techniques as they differ in performance/energy trade-off spaces. So, choosing the wrong technique may give inverse, negative results of power management as demonstrated in [33].

There has been lots of research on this topic, for example, Lakshminaryana et al. [34] proposed a scheduling scheme which schedules a task with longer remaining time to a faster core. Also, Becchi et al. [35] used Instruction per Cycle (IPC) as a metric for the assignment of threads to cores. Srinivasan et al. [36] used performance prediction model to predict application behavior on different cores and hence assign thread to the proper core.

Some works used Last-Level Cache (LLC) misses as a metric of scheduling threads as in [37]. LLC miss rate provides an indication of the intensity of off-chip memory accesses which contribute with a big part to the overall latency and in turns can be a good metric for identifying the nature of each core/thread. Manakkadu et al. [38] proposed a technique for identifying critical sections in threads based on a scoring mechanism. Based on the score for each thread, best optimization decisions can be made. Their study claims that by applying asymmetric frequency scaling directed by the proposed metric, 28.13% savings in average power consumption were achieved with a maximum of 7.1% performance loss. The approach used in that paper depends on the IPC for each thread in different time intervals. It assumes that execution time is divided into equal time intervals. At each interval, each thread has a specific IPC. The score at a specific interval for a thread is found by dividing the thread's IPC at that period by the summation of all threads IPCs at the same period. The final score of the thread is obtained by summing up all the thread's scores at all intervals. The formula can be written as follows:

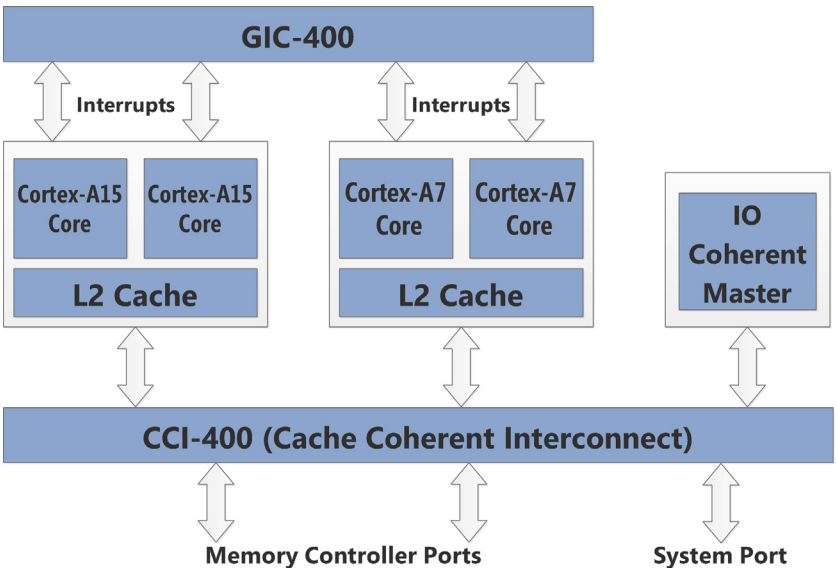


**Figure 8** ARM big.LITTLE architecture.

$$\sum_t \left( \frac{\mathrm{IPC}_i^t}{\sum_i \mathrm{IPC}_i^t} \right) \tag{3}$$

where is the IPC of the $i$th thread at interval $t$. The thread with the highest score is the most critical thread among other threads and hence speeding it up would reduce the overall execution time.

In [39], Petrucci et al. proposed an optimization approach that includes an Integer Linear Programming model [40] (rather than concentrating on a single performance metric) and a scheme to dynamically solve the problem of thread-to-core assignment. It uses a regression model to predict at runtime how thread would perform on different types of cores available and hence map that thread to the core that suits it best. Their stated results show an EDP improvement over other scheduling techniques of 10–40% depending on the workload. However, that approach works by reallocating a thread to another core when the performance profile changes. This means that a thread may execute on the wrong core for a while before its performance profile changes and hence allowing many inefficient power states. Also, the action of thread relocating may degrade performance significantly especially in the case of shared LLC as contention may exist.

Moreover, frameworks for dynamic power management in such asymmetric architectures were purposed. In [41] one is presented. The framework proposed is price theory based that tries to exploit all energy saving opportunities. It employs DVFS, load balancing and thread re-allocation to achieve as high energy savings rates as possible. That study designed and implemented that framework within Linux OS on ARM big.LITTLE architecture. We found that framework very promising as it uses more than a single power saving opportunity and it is distributed and scalable. However, it divides cores into clusters. Each cluster have a specific V–F setting and all cores within a cluster are constrained to be symmetric. This reduces flexibility and wastes some good power saving opportunities. Also, the framework takes into consideration the priority of the task which is assumed to be assigned by the user. That introduces some level of user overhead. Moreover, the framework is implemented within the linux kernel which means it requires modifications in the kernel itself and also requires the modified version of the kernel to be re-built to make use of the framework.

#### 3.2.3. Many-type asymmetric cores

In addition to using only two types of cores in asymmetric chip multiprocessor, Kumar et al. [42] proposed single-ISA heterogeneous multi-core architectures that assume a single chip contains different kinds of cores (not only two) based on different power/performance requirements. The study decides which core is best for a certain application execution and then ship the application to that core. It assumes example architecture with five cores with different architectures and claims that initial results show a 3× power reduction in the cost of only 18% performance loss.

#### 3.2.4. Limitations of asymmetric cores

From the above discussion, it is obvious that asymmetric chip multiprocessor architectures suffer from some limitations. One of the most important limitations is that the number of large and small core is fixed at design time and cannot be modified. This reduces the flexibility of accommodating to the software diversity.

Another important limitation of asymmetric CMPs is the overhead and latency introduced during the process of thread migration between cores. Along with that latency, another data locality issue is presented. Each core has one or more levels of private cache. When threads are migrated from one core to another, data must also be migrated from one cache to another or must be fetched again from main memory. That introduces another communication/delay overhead and limits performance improvements.

### 3.3. Thread motion

#### 3.3.1. Basic concept

Based on [43], this technique was proposed to enhance the well-known DVFS. It proved that 2 levels of voltage–frequency domains are sufficient to improve performance. The

idea of thread motion is to have small cores, running at two different levels of voltage–frequency levels. When applications are executed, the algorithm decides which core has the best voltage–frequency setting to execute that application and moves it to that core instead of changing voltage–frequency pair for that core which introduced more latency. Thread Motion enables applications to migrate to cores with higher or lower voltage/frequency settings depending on the current workload of the program. For example, if one application could benefit from a higher voltage/frequency setting on some core while the application on that core is stalled for I/O operation for example, thread motion swaps the two applications between the cores.

#### 3.3.2. Limitations of thread motion

The limitation of this technique is that it was proposed for simple, homogeneous cores and it's also limited to power-constrained multi-core systems. The results show that it provides up to 20% better performance than coarse-grained DVFS.

### 3.4. Variable size cores

#### 3.4.1. Basic concept

The basic idea is to design a complex, large core that is able to degrade later into a small core [44]. This can be done through dynamically disabling execution units and even pipestages. This idea is based on the classic power gating [45] technique. Power gating algorithms typically operate by turning off the resource if it has been idle for a specified number of clock cycles. In cases of high scalar workloads (low parallelism), run a few cores that would fully-operate to support the scalar performance. However, when dealing with highly parallel workloads, it would be power/throughput performance efficient to run many cores using fewer resources on each core.

#### 3.4.2. Limitations of power-gating

It is obvious that power-gating (and consequently variable-size cores) has some serious limitations:

*3.4.2.1. Mis-prediction.* As we viewed, power-gating algorithms depend on turning off a resource that is reported idle for a specified number of clock cycles. Hence, the controller may turn off some resource which was idle just before the application needs that resource again, giving negative power savings and degrading performance significantly.

*3.4.2.2. Small power savings.* While turning off some portions/resources of the systems saves power consumption, it is not of that great impact. Power savings resulting from this technique are very minimal compared to other techniques such as DVFS.

### 3.5. Speculation control

#### 3.5.1. Basic concept

Some energy is wasted on mis-speculated instructions for example, instructions after a mis-predicted branch. The results of a mis-speculated instruction are more likely to be discarded but energy has been wasted anyway to execute that instruction. Speculation reduction technique suggests that in cases of highly parallel workload, it is power efficient to run many cores with little speculation on each core. In scalar workloads, it's advisable to run a few cores with as much speculation as possible.

#### 3.5.2. Limitations of speculation control

Regarding limitations of this technique, it is not very useful in cases of parallel workloads. Parallel workloads do not suffer a lot from mis-speculated instructions. Also, latency introduced by the pipeline degrades performance significantly.

### 3.6. Core fusion

#### 3.6.1. Basic concept

Core Fusion [6] is re-configurable chip multiprocessor architecture that starts with small simple cores which are able to dynamically fuse into a larger core to support scalar performance when needed. It neither requires special programming effort nor specialized compiler support. Core Fusion can accommodate to software diversity and variations of workloads. When the workload is extremely parallel, distribute the workload among the simple cores. When the workload is heavily scalar, the simple cores dynamically fuse into a larger, more powerful single core. Full details of hardware implementation of this architecture can be found in [6]. Many re-configurable architectures used Core Fusion as the foundation [46–48].

#### 3.6.2. Limitations of core fusion

Limitations of Core Fusion according to [49] include that the fused large core consumes lots of power and is slower than a traditional out-of-order core because there are additional latencies among the pipeline stages of the fused core. Also, mode switching between small cores and fused core comes at the cost of flushing instruction cache and moving data between caches.

## 4. Proposed technique

Based on the above discussion and referring to the comparison provided by Table 1, we were able to propose a technique we think it will provide best balance between consumed power reduction and overall performance/throughput. This technique would make use of clustered DVFS [25] with Retroactive Frequency Scaling (RFS) [27] in Asymmetric [5] Many-Type Multicore Processor [42] which schedules critical sections threads using the scoring mechanism [38]. Power gating technique [44] may also be used in cases of very low CPU utilization. The technique will be controlled via an adaptive control mechanism which decides based on many parameters (workload style, current cores utilization, available amount of parallelism, current performance, e.g.) how to use that technique efficiently.

For example, when the initial workload is highly-parallel, small cores frequency will be fixed while the frequency of the large cores will be scaled down and all cores will be used to execute that parallel code. If the workload contains lots of sequential code, large cores will be used at maximum frequency. Our technique is currently subject to further research and validating it using simulation is our future work.

**Table 1** Comparison between different power management techniques in multi-core processors.

| | DVFS | Asymmetric Cores | Thread Motion | Variable-size cores | Speculation Control | Core Fusion |
|---|---|---|---|---|---|---|
| Idea | Change voltage and frequency according to the required performance/throughput | Having small cores for parallel code execution and a large core for scalar code execution | Having almost 2 voltage/frequency domains, migrating threads between cores of different domains | Turning off/on resources as needed to save power | Reduce speculation to save power | Having small cores that execute parallel code and can dynamically fuse into a large core |
| Advantages | – very effective power savings with minimal performance degradation<br>– easy to implement on many scales | – effective results in power savings<br>– accommodates well to software diversity | – fast change between different VF settings<br>– effective results in power savings (rely on DVFS) | – very simple to implement<br>– can be used in conjunction with other techniques | Future work suggests the evaluation of such hybrid, compatible techniques/improvements to get further better results in both performance and energy terms. | – low area cost<br>– excellent accommodation to software diversity<br>– no extra programming effort needed |
| Shortcomings | – long transition time between power states<br>– the level of granularity to which DVFS is applied affects cost and performance | – number of small/big cores is fixed at design time<br>– latency introduced during thread migration<br>– data locality issue during migration<br>– choosing appropriate scheduling/mapping technique | – limited to homogenous CMP architectures | – very low power saving when used alone<br>– mis-prediction of resources to be turned off may result in negative power savings | – not very useful in cases of parallel workloads as they do not suffer a lot from mis-speculated instructions<br>– latency introduced by the pipeline | – more latencies in the fused core than a traditional OoO core.<br>– cache flushing and data migration delay during mode switching |

## 5. Conclusion

In this paper we investigated the concept of multi-core processors, research trends in that field and focused on the power management issue. We reviewed most of the used techniques, their advantages and disadvantages and the research done for each technique to address its problems. Finally, we proposed a new technique that makes use of the gathered results. It is very clear from the discussion that there is no absolute perfect way for power management in chip multiprocessor architecture. It depends on whether or not you are open to changes in the architecture itself, how much you can sacrifice performance and the amount of workload you expect on your chip. However, the combination of some techniques with solutions made to improve those techniques is an excellent choice to think of. For example, applying thread scoring in many-type asymmetric cores seems very promising. Future work suggests the evaluation of such hybrid, compatible techniques/improvements to get further better results in both performance and energy terms.

## References

[1] Moore Gordon E. Cramming more components onto integrated circuits, vol. 12; 1965.
[2] Wall DW. Limits of instruction-level parallelism. WRL Research Report 93/6. Digital Western Research Laboratory, Palo Alto (CA); 1993.
[3] Olukotun Kunle et al. The case for a single-chip multiprocessor. ACM Sigplan Notices 1996;31(9):2–11.
[4] Amdahl Gene M. Validity of the single processor approach to achieving large scale computing capabilities. In: Proceedings of the April 18–20, 1967, spring joint computer conference. ACM; 1967.
[5] Hill Mark D, Marty Michael R. Amdahl's law in the multicore era. IEEE Comput 2008;41(7):33–8.
[6] Ipek Engin et al. Core fusion: accommodating software diversity in chip multiprocessors. ACM SIGARCH computer architecture news, vol. 35(2). ACM; 2007.
[7] Grochowski Ed et al. Proceedings IEEE international conference on computer design: VLSI in computers and processors, 2004. ICCD 2004. IEEE; 2004.
[8] Sprangle Eric, Doug Carmean. Increasing processor performance by implementing deeper pipelines. In: Proceedings 29th annual international symposium on computer architecture, 2002. IEEE; 2002.
[9] Smith James E, Sohi Gurindar S. The microarchitecture of superscalar processors. Proc IEEE 1995;83(12):1609–24.
[10] Bircher William Lloyd, John Lizy. Predictive power management for multi-core processors. In: Computer architecture. Berlin, Heidelberg: Springer; 2012.
[11] Horowitz Mark, Indermaur Thomas, Gonzalez Ricardo. Low-power digital design. In: Low power electronics, 1994. Digest of technical papers. IEEE; 1994.
[12] Bertran Ramon et al. A systematic methodology to generate decomposable and responsive power models for CMPs. IEEE Trans Comput 2013;62(7):1289–302.
[13] Basmadjian Robert, de Meer Hermann. Evaluating and modeling power consumption of multi-core processors. In: 2012 third international conference on future energy systems: where energy, computing and communication meet (e-Energy). IEEE; 2012.
[14] Rethinagiri Santhosh Kumar et al. System-level power estimation tool for embedded processor based platforms. In: Proceedings of the 6th workshop on rapid simulation and performance evaluation: methods and tools. ACM; 2014.
[15] Walker Matthew J et al. Run-time power estimation for mobile ad embedded asymmetric multi-core CPUs; 2015.

[16] Weiser Mark et al. Scheduling for reduced CPU energy. Mobile computing. US: Springer; 1996, p. 449–71.
[17] Khan UA, Rinner B. Online learning of timeout policies for dynamic power management. ACM-TECS 2014;13(4):25 96.
[18] Das Anup et al. Reinforcement learning-based inter-and intra-application thermal optimization for lifetime improvement of multicore systems. In: 2014 51st ACM/EDAC/IEEE design automation conference (DAC). IEEE; 2014.
[19] Ye Rong, Xu Qiang. Learning-based power management for multicore processors via idle period manipulation. IEEE Trans Comput-Aided Des Integr Circ Syst 2014;33(7):1043–55.
[20] Shen Hao et al. Achieving autonomous power management using reinforcement learning. ACM Trans Des Autom Electr Syst (TODAES) 2013;18(2):24.
[21] Otoom Mwaffaq et al. Scalable and dynamic global power management for multicore chips. In: Proceedings of the 6th workshop on parallel programming and run-time management techniques for many-core architectures. ACM; 2015.
[22] Chao Seong Jin, Yun Seung Hyun, Jeon Jae Wook. A powersaving DVFS algorithm based on operational intensity for embedded systems. IEICE Electr Exp 2015;0.
[23] Li Zheng, Shangping Ren, Gang Quan. Energy minimization for reliability-guaranteed real-time applications using DVFS and checkpointing techniques. J Syst Architect 2015.
[24] Kamga Christine Mayap. CPU frequency emulation based on DVFS. ACM SIGOPS Operating Systs Rev 2013;47(3):34–41.
[25] Kolpe Tejaswini, Zhai Antonia, Sapatnekar Sachin S. Enabling improved power management in multicore processors through clustered DVFS. In: Design, automation & test in europe conference & exhibition (DATE), 2011. IEEE; 2011.
[26] Lai Zhiquan et al. Latency-aware dynamic voltage and frequency scaling on many-core architectures for data-intensive applications. In: 2013 international conference on cloud computing and big data (CloudCom-Asia). IEEE; 2013.
[27] Lai Zhiquan, Zhao Baokang, Su Jinshu. Efficient DVFS to prevent hard faults for many-core architectures. Information and communication technology. Berlin, Heidelberg: Springer; 2014, p. 674–9.
[28] Greenhalgh Peter. Big.little processing with arm cortex-a15 & cortex-a7. ARM White Paper; 2011.
[29] Chung Hongsuk, Kang Munsik, Cho Hyun-Duk. Heterogeneous multi-processing solution of Exynos 5 Octa with ARM® big. LITTLE™ Technology.
[30] Rajovic Nikola et al. Experiences with mobile processors for energy efficient HPC. In: Proceedings of the conference on design, automation and test in Europe. EDA Consortium; 2013.
[31] Kihm J, Guimbretière FV, Karl J, Manohar R. Using asymmetric cores to reduce power consumption for interactive devices with bistable displays. In: Proc 32nd annu ACM conf hum factors comput syst – CHI'14; 2014. p. 1059–62.
[32] Marowka A. Maximizing energy saving of dual-architecture processors using DVFS. J Supercomput 2014;68:1163–83.
[33] Imes Connor, Hoffmann Henry. Minimizing energy under performance constraints on embedded platforms; 2015. p. 12.
[34] Lakshminarayana Nagesh B, Lee Jaekyu, Kim Hyesoon. Age based scheduling for asymmetric multiprocessors. In: Proceedings of the conference on high performance computing networking, storage and analysis. ACM; 2009.
[35] Becchi Michela, Crowley Patrick. Dynamic thread assignment on heterogeneous multiprocessor architectures. In: Proceedings of the 3rd conference on computing frontiers. ACM; 2006.
[36] Srinivasan Sadagopan et al. HeteroScouts: hardware assist for OS scheduling in heterogeneous CMPs. In: Proceedings of the ACM SIGMETRICS joint international conference on measurement and modeling of computer systems. ACM; 2011.
[37] Koufaty David, Reddy Dheeraj, Hahn Scott. Bias scheduling in heterogeneous multi-core architectures. In: Proceedings of the 5th European conference on computer systems. ACM; 2010.
[38] Manakkadu Sheheeda, Dutta Sourav, Botros Nazeih M. Power aware parallel computing on asymmetric multiprocessor. In: 2014 27th IEEE international system-on-chip conference (SOCC). IEEE; 2014.
[39] Petrucci Vinicius et al. Energy-efficient thread assignment optimization for heterogeneous multicore systems. ACM Trans Embed Comput Syst (TECS) 2015;14.1:15.
[40] Wagner Harvey M. An integer linear-programming model for machine scheduling. Naval Res Logist Quart 1959;6(2):131–40.
[41] Somu Muthukaruppan T, Pathania A, Mitra T. Price theory based power management for heterogeneous multi-cores. In: Proc 19th int conf archit support program lang oper syst – ASPLOS'14; 2014. p. 161–76.
[42] Kumar Rakesh et al. A multi-core approach to addressing the energy-complexity problem in microprocessors. In: Workshop on complexity-effective design; 2003.
[43] Rangan Krishna K, Wei Gu-Yeon, Brooks David. Thread motion: fine-grained power management for multi-core systems. ACM SIGARCH computer architecture news, vol. 37(3). ACM; 2009.
[44] Efthymiou Aristides, Garside Jim D. Adaptive pipeline depth control for processor power-management. In: Proceedings 2002 IEEE international conference on computer design: VLSI in computers and processors, 2002. IEEE; 2002.
[45] Hu Z et al. Microarchitectural techniques for power-gating of execution units. In: Proc int'l symp on low power electronics and design, ISLPED, Aug. 2004.
[46] Boyer M, Tarjan D, Skadron K. Federation: boosting per-thread performance of throughput-oriented manycore architectures. In: ACM trans archit code optim (TACO); 2010.
[47] Pricopi M, Mitra T. Bahurupi: a polymorphic heterogeneous multi-core architecture. In: ACM TACO, January 2012.
[48] Gibson D, Wood DA. ForwardFlow: a scalable core for power-constrained CMPs, in ISCA; 2010.
[49] Khubaib K et al. Morphcore: an energy-efficient microarchitecture for high performance ilp and high throughput tlp. In: 2012 45th annual IEEE/ACM international symposium on microarchitecture (MICRO). IEEE; 2012.

**Khaled M. Attia** is a teaching assistant at Computers and Control Systems Engineering Department, Mansoura University. He received his B.Sc. in 2013 with an overall grade of excellent with honors from Mansoura University. His main research interests include Computer Architecture and Organization, Heterogeneous Multi-Core Architectures, Power-aware computing and Heterogeneous Parallel Programming.

**Mostafa A. El-Hosseini** is an Ass. Professor at Computers Engineering and Control systems Dept.—Faculty of Engineering—Mansoura University, Egypt. He received the B.Sc. from the Electronics Engineering Department, M.Sc. and Ph.D. from Computers & Systems Engineering, all from Mansoura University, Egypt. His major research interests are Artificial Intelligence such as Genetic Algorithms, Neural Networks, Particle Swarm Optimization, Simulated Annealing, and Fuzzy Logic. Also he is interested in the application of AI in Machine Learning, Image Processing, access control and Optimization. The Applications of Computational Intelligence CI and Soft Computing tool in Bioinformatics is also one of his interests. He served as a member of the international program committees of numerous international conferences.

**Hesham Arafat** ali is a Prof. in Computer Eng. & Sys. and an assoc. Prof. in Info. Sys. and computer Eng. He was assistant prof. at the Univ. of Mansoura, Faculty of Computer Science from 1997 up to 1999. From January 2000 up to September 2001, he joined as Visiting Professor to the Department of Computer Science, University of Connecticut. From 2002 to 2004 he was a vice dean for student affair the Faculty of Computer Science and Inf., Univ. of Mansoura. He was awarded with the Highly Commended Award from Emerald Literati Club 2002 for his research on network security. He is a founder member of the IEEE SMC Society Technical Committee on Enterprise Information Systems (EIS). He has many book chapters published by international press and about 150 published papers in international (conf. and journal). He has served as a reviewer for many high quality journals, including Journal of Engineering Mansoura University. His interests are in the areas of network security, mobile agent, Network management, Search engine, pattern recognition, distributed databases, and performance analysis.