or

Published by John Wiley & Sons, Inc., Hoboken, New Jersey.
Published simultaneously in Canada.

Library of Congress Cataloging-in-Publication Data:

Title, etc
Printed in the United States of America.

10 9 8 7 6 5 4 3 2 1

CONTENTS

1 The Future in Mobile Multicore Computing **1**
Blake Hurd, Chiu C. Tan, and Jie Wu

1.1 Introduction 1
1.2 Background 3
1.2.1 GPGPU Implementation 3
1.2.2 Intelligent Power Scaling Implementation 4
1.2.3 Multi-Tasking Implementation 4
1.3 Hardware Initiatives 5
1.3.1 Chipset Support 5
1.3.2 Impact 7
1.4 Software Initiatives 7
1.4.1 Language Support 8
1.4.2 Impact 8
1.5 Additional Discussion 9
1.5.1 Company-specific Initiatives 9
1.5.2 Embedded Computing Research Initiatives 10
1.6 Future Trends 10

1.7 Conclusion 11
References 11

CHAPTER 1

THE FUTURE IN MOBILE MULTICORE COMPUTING

BLAKE HURD, CHIU C. TAN, AND JIE WU

Temple University, Philadelphia, Pennsylvania

1.1 INTRODUCTION

Mobile computers are with us everywhere, allowing us to work and entertain ourselves at any venue. Due to this, mobile computers are replacing desktops as our personal computers. Already, we see signs of smartphones becomming more popular than traditional desktop computers [1]. A recent survey of users reveals that email, Internet access, and a digital camera are the three most desirable features in a mobile phone, and the consumers wanted these features to be as fast as possible [2]. The increasing sales of more powerful phones also indicate consumer demand for more powerful phones [1, 3].

There are two ways to improve mobile computing. The first way is to execute the computation *remotely*, where the mobile phone transfers the processing to a remote platform, such as a cloud computing environment, to perform the computation and

title, edition. By author

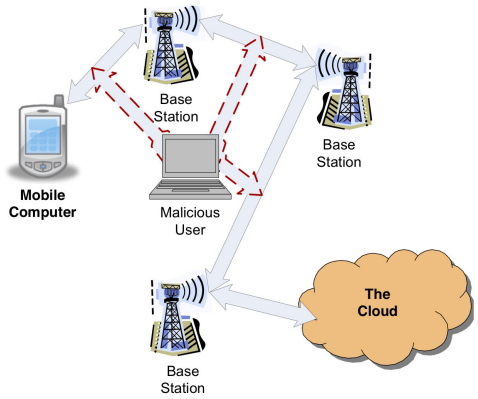

Figure 1.1 Remote computation requires greater security, energy, and latency consideration than local computation.

then retrieves the output. The alternative is for the mobile device to execute the computation *locally* using its own hardware. The following three factors make remote computation less ideal than local computation.

1. **Security.** Remote computation requires outsourcing data to a third party, which increases the security risks since the third party may not be trustworthy. For instance, the third party may utilize the data to violate the user's privacy. Local computation, on the other hand, does not have this problem.

2. **Efficiency.** Transmitting data to a remote server may incur a higher energy cost [4] due to the large communication overhead of the wireless transmission. Furthermore, remote computation requires utilizing more bandwidth, which can be more expensive in environments where bandwidth is metered. Local computation can avoid the high bandwidth charges, and, as we will show in subsequent sections, may be more power efficient.

3. **Timeliness.** It is difficult to guarantee timeliness when using remote computation due to the unpredictability of wireless communications under different environmental conditions, such as traveling on a subway. Local computation is not affected by this issue.

In this paper, we are concerned with scenarios where local computation is better than remote computation. Local computation has its own set of challenges, specifically to increase its energy efficiency while improving its timeliness. The two requirements are somewhat contradictory since reducing the processor speed is a important component of improving the energy efficiency; however this, will result in a longer computation time and will decrease timeliness. This is the case in a single core architecture. In this paper, we will show that multicore architectures do not have this limitation.

1.2 BACKGROUND

The adoption of general purpose GPUs (GPGPUs) and multicore CPUs into mobile devices allows these devices to perform powerful local computations. GPGPUs allow parallelized programs to run on the GPU. These programs run on both the CPU and GPU as needed, and this level of hardware flexibility allows software to present a better user experience. We define Mobile Multicore Computing (MMC) as a mobile computer computing with a GPGPU and/or multicore CPU. MMC is quickly becoming a reality: tablets are available, and smartphones will be available during the first quarter 2011. Projections suggest that by 2013, most mobile platforms, 88%, will have MMC architecture [5].

We explore three technical issues related to MMC: implementation of GPGPUs, intelligent power scaling, and multi-tasking applications. There are more issues, but we expect these three to be the most important currently and in the future. For each issue, we will examine its importance to MMC as well as the challenges. We divide our discussion into hardware and software components. We analyze why each is difficult to solve and why each is beneficial. We discuss hardware and software support and the impact of this support on these issues.

GPGPU, power scaling, and multi-tasking, enabled by MMC hardware and software support, will allow for increased performance and increased energy efficiency to be possible. For example, there are advantages in using a multicore CPU. Single-core CPUs increase performance linearly at the expense of an exponential increase in power; multicore CPUs can increase performance linearly for a linear increase in power. In other words, if we get similar throughput from two 400mhz cores versus one 800mhz core, we save power by using the two 400mhz cores. Nvidia tests this concept and demonstrates a 40% power improvement to achieve the same performance benchmark on their latest mobile chipset [6].

1.2.1 GPGPU Implementation

GPGPU uses the GPU's synchronized group of cores to process small, parallelized, non-graphic tasks in parallel to run certain tasks faster and more efficiently than CPUs. GPGPU processing is suitable for tasks that can be split into parallel data, such as matrices or arrays of data where each sector needs the same instruction executed. These highly parallel tasks are best run on the GPGPU instead of the CPU. For example, video encoding and decoding is more efficient when run via the GPGPU.

The GPGPU cannot be widely implemented until an agreed standard between hardware and software is achieved. Each chipset needs to incorporate a GPGPU that supports a language for sending tasks to the GPU and retrieving results. The language supported must be a determined standard; otherwise, programs must be re-developed based on what device the programs are running on.

Mobile computers have to calculate a multitude of massively parallel problems like video processing, wireless baseband processing, Fast Fourier Transform (FFT),

and packet routing. If these calculations are run via the GPGPU rather than the CPU, the performance gained and power efficiency is highly beneficial.

1.2.2 Intelligent Power Scaling Implementation

All current chipsets allow Dynamic Voltage Frequency Scaling (DVFS) and static power domains. This allows complex, robust implementations for intelligently scaling chipset power consumption. Static power domains allow the CPU to move between frequency boundaries, and DVFS allows dynamic voltage and CPU frequency tweaking. A sufficiently intelligent power scaling implementation wields these features optimally.

When all applications are running on one core, finding the optimal frequency is simple, and thus a power scaling implementation is simple. However, in a multicore environment, cores manage different workloads, and some environments require each core to run at the same frequency. Overall, this problem, optimal power management for DVFS-enabled multicore processing, is proven to be NP-hard [7].

If each processor core runs at a frequency that meets the user's requirements and no faster, then the mobile device may conserve power. The benefits are substantial: with hundreds of millions of smartphones (there are supposedly 170 million [8] sold every year), improving phone power consumption efficiency by 5% will save the amount of energy equivalent of 8.5 million smartphones.

1.2.3 Multi-Tasking Implementation

Multi-tasking refers to running as many tasks at the same time as possible. The goal is to give the user the perception of complete parallelization - that one can run as many applications as one wants to without any limitations. Multi-tasking is essentially multi-threading; each application is separated into *threads*, or *tasks*, and then each thread/task is scheduled in proper balance to multi-task.

The difficulty is developing an intelligent scheduler that balances as many processes/threads as possible while providing a satisfactory experience. Mobile multi-tasking is more challenging due to limited memory and power. Limited memory requires applications to be small. In addition, code reuse is also necessary if multiple applications are kept in memory at the same time. If the foreground application needs memory that other applications are using, it will slow down and more power is spent. Background applications also drain the battery when the ongoing workload causes a measurable task switching overhead. Users may launch applications without ending any running applications and inadvertently drain the limited power. Finally, because most mobile phones use ARM-designed CPUs, the scheduler must consider the design's slower task/process switching.

If multi-threading is enabled, the OS may redesign their scheduler to share independent threads across multiple cores once available. Non multi-tasking OSs only maintain the thread(s) allocated to the application in-use with the thread(s) used by the OS itself. Such OSs can only schedule the OS thread(s) on a different core, and there would be trouble with scheduling the thread(s) allocated to the application in

Table 1.1 Current Smartphone Architectures

Phone	CPU	GPU	GPGPU	Power Scaling	Multi-tasking
Apple iPhone 3G	ARM11 MPCore	PowerVR MBX Lite	No	Yes	No
Apple iPhone 3GS	ARM Cortex-A8	PowerVR SGX535	No	Yes	Partial
Apple iPhone 4	ARM Cortex-A8	PowerVR SGX535	No	Yes	Partial
HTC Nexus One	Qualcomm Scorpion	Qualcomm Adreno 200	No	Yes	Partial
HP-Palm Pre	ARM Cortex-A8	PowerVR SGX530	No	Yes	Partial
HP-Palm Pixi	ARM11 MPCore	Qualcomm Adreno 200	No	Yes	Partial
HP-Palm Pre 2	ARM Cortex-A8	PowerVR SGX530	No	Yes	Partial
HTC Evo 4G	Qualcomm Scorpion	Qualcomm Adreno 200	No	Yes	Partial
Microsoft Kin One & Two	ARM11 MPCore	Nvidia ULP GeForce	No	Yes	Partial
Motorola Droid	ARM Cortex-A8	PowerVR SGX530	No	Yes	Partial
Samsung Galaxy S	ARM Cortex-A8	PowerVR SGX540	No	Yes	Partial

use; the application may require an order to commence thread execution and may lock up the cores, or the application may have a single thread.

1.3 HARDWARE INITIATIVES

Hardware initiatives create a new hardware architecture foundation for solving our three technical issues. *Chipsets* are the main hardware initiative; a new chipset allows software to utlize more capabilities and to present a stronger device. The latest chipsets mostly rely on the latest ARM designs, which are licensed to most companies releasing chipsets. The chipsets are created by combining a CPU, a GPU, specialized processing units, and memory.

1.3.1 Chipset Support

We discuss the progress of multicore CPUs and the progress of GPGPUs; then, we discuss the state of current and future generation smartphone architectures in utilizing these MMC components.

Table 1.2 Upcoming Chipsets

Chipset	CPU	GPU	GPGPU	Power Scaling	Multi-Tasking
Apple A5 ARM	Cortex-A9	PowerVR SGX543MP2	Yes	shared	Yes
Qualcomm Snapdragon QSD8x50A	Qualcomm Scorpion	Qualcomm Adreno 205	No	per-core	Partial
Qualcomm Snapdragon MSM8x60	Qualcomm Scorpion	Qualcomm Adreno 220	No	per-core	Yes
Qualcomm Snapdragon QSD8x72	Qualcomm Scorpion	Qualcomm Adreno 220	No	per-core	Yes
Nvidia Tegra 2 T20	ARM Cortex-A9	Nvidia ULP GeForce	No	shared	Yes
Nvidia Tegra 2 AP20H	ARM Cortex-A9	Nvidia ULP GeForce	No	shared	Yes
Nvidia Tegra 2 3D T25/AP25	ARM Cortex-A9	Nvidia ULP GeForce	No	shared	Yes
TI OMAP4430	ARM Cortex-A9	PowerVR SGX540	No	shared	Yes
TI OMAP4440	ARM Cortex-A9	PowerVR SGX540	No	shared	Yes
Samsung Orion	ARM Cortex-A9	ARM Mali-400	No	shared	Yes
ST-Ericsson U8500	ARM Cortex-A9	ARM Mali-400	No	shared	Yes

Most mobile CPUs are based on ARM's design licenses. There are four relevant generations of mobile ARM CPU designs: the ARM11 MPCore, the ARM Cortex-A8, the ARM Cortex-A9, and the ARM Cortex-A15. The ARM11 MPCore is the oldest and the least energy-efficient, but it is the cheapest to produce. The Cortex-A8 only allows single core. Otherwise, the other three generations allow multicore, up to four cores, at improved energy efficiency and performance between generations.

Various mobile GPUs are available to pair with these CPUs. PowerVR's SGX series is the most popular, and a GPGPU was released that is available for the future generation of smartphones. The early 2011 generation added dual-core CPUs, but later 2011 models should include GPGPUs, starting with Apple's newest smartphone.

Every CPU in the 2010 generation smartphone architectures is single-core (see Table 1.1). All current phones support DVFS for power scaling, and almost all current phones have multicore sequels in development. There is a popular mobile GPU that

introduces GPGPU support [9]; most future generation mobile phones should use this GPU and thus implement GPGPU.

Multicore chipsets (see Table 1.2) were available in smartphones and other mobile computers in early 2011, and multicore is common in future smartphone architecture. The ARM Cortex-A9 CPU supports shared DVFS and four static power domains, whereas the Qualcomm Scorpion CPU supports a per-core DVFS. Shared DVFS means that each core in the same power domain must run at the same frequency, and each power domain provides an upper and lower bound for modifying the frequency. In per-core, each core can always run at a frequency independent of other cores. Per-core DVFS thus allows more flexible power scaling solutions.

1.3.2 Impact

As discussed in chipset support, all chipsets in the current generation do not implement GPGPU. Chipsets in future generations will. Incorporating the GPGPU into the chipset allows software initiatives to be designed to take advantage of the GPGPU.

Power scaling has hardware support in the previous generation but is more important in this next generation. With two CPU cores to manage, either with shared or per-core DVFS, algorithms must handle more conditions, but the potential energy efficiency achieved is more beneficial. The industry is counting on implementations of intelligent power scaling algorithms to take advantage of parallelism to lower the system's power consumption.

Multi-tasking was limited in the previous generation. With one CPU core to run on and no GPGPU support, multi-tasking was largely about swapping running tasks fast enough to give the user the perception of multi-tasking. With two CPU cores available for general purpose programming and with GPGPU support upcoming, tasks can be distributed across the available hardware and can be executed at virtually the same time. This requires additional research to better perform load balancing of the various requirements. Load balanced multi-tasking is also related to power scaling; if CPU cores are multi-tasking the workload at the best balance possible, we can measure the minimum frequency needed to maintain that balance and set the CPU to that frequency.

1.4 SOFTWARE INITIATIVES

Software initiatives use the foundation laid by the hardware initiatives to solve the issues that MMC presents. Languages are the main software initiative, specifically streaming languages. Streaming languages provide the software side of implementing GPGPU on a mobile computer and also assist power scaling algorithms. Streaming languages allow for better parallelization of each application. A higher degree of parallization will allow power scaling algorithms to better balance the energy, and thus lower the overall power consumption. The streaming languages may also support load balancing algorithms for multi-tasking and balancing programs over CPUs and the GPGPU.

1.4.1 Language Support

Streaming languages allow fine-grained parallelization to weave into serial programming. A set of data, called a *stream*, is presented with a list of operations, called *kernel functions*, which are then applied to every element in the data stream. These streams can illustrate a graph and allow the stream language compiler to parallelize the program. The most popular languages are OpenCL and CUDA. OpenCL is the future standard, but CUDA remains available due to the limited hardware and software support for OpenCL. In addition to these, we briefly discuss Brook and StreamIt, which are popular in academia.

OpenCL is an open-source project that is supported by the mobile industry as the future of parallel programming. AMD and Nvidia support OpenCL on their GPUs, so OpenCL will be supported on most, if not all, GPUs in the future. OpenCL provides a framework for managing CPUs and other processing units through a sublanguage that is used beside higher level languages like C++ or Java. Serial programming is still allowed by programming in C++ and Java without the bindings, and weaves into parallel programming through OpenCL bindings. OpenCL will be available to mobile chipsets before CUDA, as the first mobile GPGPU supports OpenCL [9].

GPGPU is possible if executing on an Nvidia GPU that supports CUDA. However, the latest mobile Nvidia chipsets do not support CUDA, so a mobile GPGPU through CUDA is currently not possible. Furthermore, CUDA only controls GPGPUs; CUDA does not affect the programming framework of different types of processing units and multicore CPU management. Nvidia continues to support CUDA, but also supports OpenCL, releasing APIs for converting code from CUDA to the OpenCL framework.

Brook and StreamIt are two academic languages for parallel programming. Brook is similar to OpenCL and CUDA, predating both languages. StreamIt, alternatively, is unique and remains popular in research, presenting a high-level language for designing highly parallel programs. StreamIt depends on its own compiler, and its syntax restricts its capacity for handling serial code. However, StreamIt remains popular and papers continue to be published, as it is effective for creating highly parallel programs.

1.4.2 Impact

GPGPUs need a streaming language, otherwise programs cannot communicate to it. Every major GPU developer has chosen to support OpenCL, releasing an OpenCL driver to support OpenCL on their GPU. Writing OpenCL then allows programs to execute in parallel on the GPU, thus completing GPGPU implementation.

Power scaling may be addressed by software support. Application developers can parallelize their programs by using the tools available to develop more power efficient and scalable algorithms. Power scaling algorithms can also account for the differences of a CPU execution versus a GPGPU execution.

Multi-tasking solutions can be created with the help of streaming languages. Streaming languages allow programs to be designed as parallel as possible. These parallel components can be run on the MMC hardware to multi-task.

1.5 ADDITIONAL DISCUSSION

Now, we discuss new advances in industry and related academic research that are relevant to MMC.

1.5.1 Company-specific Initiatives

There are many different initiatives geared towards MMC among companies, and some are collaborative projects between them. We mention leading companies in the mobile marketplace and how they approach the technical issues of MMC.

- *Google.* Google's Android platform does not support GPGPUs yet and has not taken direct steps to improve power scaling. However, Google licensed the WebM codec, also known as the VP8 codec, which is designed as a complex encoding process with a simple, low-power decoding process. Both the encoder and decoder algorithms are parallelizable for MMC. When fully implemented, the codec may provide a multimedia experience that is more power efficient than current methods. Google's Android platform continues to develop its scheduler algorithm to multi-task better. When multicore becomes the standard, multi-tasking will be further pursued on the platform.
- *Texas Instruments.* Texas Instruments also does not support GPGPUs yet. Texas Instruments' latest platform supports power scaling like other platforms, but requires more intelligent algorithms due to multicore availability. Texas Instruments is also supporting SMP (Symmetric Multi Processing) on their latest platform; SMP means that each processor core shares memory and can run any task presented by the OS, given that the task is not being run on another core, allowing for multi-tasking.
- *ST-Ericsson/ARM/Google.* ST-Ericsson does not support GPGPUs. However, ST-Ericsson and ARM are working on a joint development on the Android OS to take advantage of SMP when executing their latest platform, which contains a multicore CPU. Like Texas Instruments' SMP support, this can lead to power scaling and multi-tasking improvements.
- *Apple.* Apple is prioritizing support for GPGPUs; their OS introduced support for GPGPU [9]. Their latest smartphone will contain a dual-core OpenCL-enabled GPGPU [10], and better power scaling and multi-tasking efforts may be attainable.
- *Microsoft.* Microsoft does not discuss implementing GPGPUs or power scaling on its mobile platform. However, on multi-tasking, Microsoft's Barrelfish research project considers a distributed multikernel OS where each application is assigned to a set of cores, is completely independent of other applications, and can asynchronously communicate with other applications.
- *Nokia.* Nokia contributes to OpenCL and further develops GPGPU implementations. Nokia's Research Center also works on power scaling under the Nokia

Research High Performance Mobile Platforms Project, but no publications in this topic have been released yet. Nokia has not announced any initiatives towards improving multi-tasking on mobile computers.

- *HP-Palm.* HP-Palm does not discuss GPGPU for Pre/WebOS. An implementation of intelligent power scaling was added via patching WebOS. WebOS partially supports multi-tasking, but it does not run on non-MMC hardware; applications run in parallel from the user's perspective, but the OS schedules intelligently on a single core. With new funding from HP, the Pre/WebOS may improve solutions to all three issues.

1.5.2 Embedded Computing Research Initiatives

Embedded computing research is tied to MMC, as mobile computers and smartphones are a subset of embedded computing.

There is research on using GPGPUs on mobile/embedded computers, but presently most GPGPU research is about running applications on desktops where the applications are potentially useful in a mobile environment, such as wireless processing, packet routing, and network coding.

In recent years, research focused on developing an intelligent power scaling scheduling algorithm that load balances and unbalances intelligently, maintaining the same performance while saving power via controlled usage of DVFS. There are many contributions in this area [11, 12, 13, 14], and we expect that more will follow, as scheduling for optimal power management on multicore is an NP-hard problem [7]. Some papers propose strategies that are alternative approaches and do not compare to previous strategies. We expect that combinations of these methods will be a future research topic, as a more robust implementation, capable of handling more workload scenarios optimally, may be possible. These scheduling algorithms must create even load balancing to save power; thus, such algorithms also contribute to the topic of improving multi-tasking.

1.6 FUTURE TRENDS

As MMC research progresses, there are several trends we observe.

Trend 1. *Software driven energy efficiency.* More advanced software solutions will be designed given the upcoming influx of MMC hardware that makes MMC possible. Current software solutions must be re-developed to balance the energy efficiency and the performance while taking advantage of multiple cores; balancing GPGPU execution versus CPU execution.

Trend 2. *Adding cores.* Hardware development will add cores for both CPUs and GPUs. For CPUs, dual-core is available, and quad-core is not far behind. The first mobile GPGPU supports 2 to 16 GPU cores, allowing future generations to scale as necessary. Adding cores adds valuable granularity and flexibility for improved energy efficiency and improved performance. Research is unclear on when adding

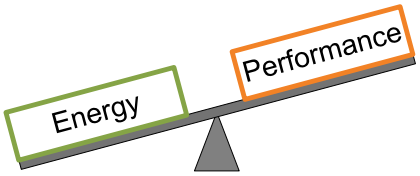

Figure 1.2 Future Trend I: Software in MMC will push energy efficiency; the energy vs. performance trade off.

cores will no longer offer any advantages in a mobile environment; that will be more important in the future.

Trend 3. *GPGPU implementation and its future in MMC.* GPGPUs will be implemented on every mobile computer with a GPU, and we expect GPGPUs to be widespread within the next few years. All MMC smartphones are slated to release in the first half of 2011 with a non-GP GPU. However, the latest of these GPUs in their respective families have GPGPU support through OpenCL support. When GPGPUs are fully implemented, increasingly many algorithms will adopt OpenCL, allowing for easy parallelization across CPUs and GPUs. We expect the implications of using a GPGPU on a mobile device to be a burgeoning topic of research.

1.7 CONCLUSION

In conclusion, we observed the progress of mobile computing, its trend toward MMC, and three important technical issues: GPGPUs are not yet being implemented, intelligent power scaling algorithms are needed, and multi-tasking algorithms are needed. We discussed hardware and software support for solutions to these issues. We additionally discussed some leading companies in the mobile marketplace, and their solutions, and different research initiatives to these issues. We discussed the field of embedded computing research, its relation to MMC, and its contributions toward these issues, specifically its contributions to an intelligent power scaling algorithm. Discussing these issues and initiatives reveal three future trends of the MMC industry that we discussed briefly: software will be developed to use the hardware more effectively to provide more energy efficiency; hardware will to add more cores, allowing for more flexibility and granularity to software solutions for improved energy efficiency; GPGPUs will take the forefront as a valuable tool for improving both energy efficiency and performance for solutions to many types of problems in mobile computing.

REFERENCES

1. D. McGrath, “IDC: Smartphones out shipped PCs in Q4,” in *EE Times*, 2011.
2. O. R. C. News, “Consumers want smarter, faster phones.” 2010.

3. E. Woyke, "The Most Powerful Smart Phones," in *Forbes.com*, 2009.
4. J. Baliga, R. Ayre, K. Hinton, and R. Tucker, "Green cloud computing: Balancing energy in processing, storage, and transport," 2011.
5. Softpedia, "In-stat research predicts multi-core cpus in 88% of mobile platforms by 2013." 2009.
6. NVIDIA, "The benefits of multiple cpu cores in mobile devices." 2010.
7. C.-Y. Yang, J.-J. Chen, and T.-W. Kuo, "An approximation algorithm for energy-efficient scheduling on a chip multiprocessor," in *IEEE Design, Automation and Test in Europe (DATE)*, 2005.
8. Gartner, "Gartner Says Worldwide Mobile Phone Sales to End Users Grew 8 Per Cent in Fourth Quarter 2009; Market Remained Flat in 2009." 2010.
9. ImaginationTechnologies, "Imagination Technologies Extends Graphics IP Core Family with POWERVR SGX543." 2009.
10. AppleINsider.com, "Apple to pack ultrafast, dual core SGX543 graphics into iPad 2, iPhone 5." 2011.
11. X. Huang, K. Li, and R. Li, "A energy efficient scheduling base on dynamic voltage and frequency scaling for multi-core embedded real-time system," in *International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP)*, 2009.
12. X. Wu, Y. Lin, J.-J. Han, and J.-L. Gaudiot, "Energy-efficient scheduling of real-time periodic tasks in multicore systems," in *IFIP international conference on Network and parallel computing (NPS)*, 2010.
13. 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*, 2008.
14. E. Seo, J. Jeong, S. Park, and J. Lee, "Energy efficient scheduling of real-time tasks on multicore processors," in *IEEE Transactions of Parallel Distributed Systems*, 2008.