M.S. THESIS

Cooperative Power Management for Chip Multiprocessors using Space-Shared Scheduling

Space-Shared 스케줄링을 사용한 칩 멀티프로세서를 위한 협업 전력 관리

August 2015

DEPARTMENT OF ELECTRICAL ENGINEERING & COMPUTER SCIENCE
COLLEGE OF ENGINEERING
SEOUL NATIONAL UNIVERSITY

Seungyul Lee

M.S. THESIS

Cooperative Power Management for Chip Multiprocessors using Space-Shared Scheduling

Space-Shared 스케줄링을 사용한 칩 멀티프로세서를 위한 협업 전력 관리

August 2015

DEPARTMENT OF ELECTRICAL ENGINEERING & COMPUTER SCIENCE
COLLEGE OF ENGINEERING
SEOUL NATIONAL UNIVERSITY

Seungyul Lee

Cooperative Power Management for Chip Multiprocessors using Space-Shared Scheduling

Space-Shared 스케줄링을 사용한 칩 멀티프로세서를 위한 협업 전력 관리

지도교수 Dr. Bernhard Egger

이 논문을 공학석사학위논문으로 제출함

2015 년 06 월

서울대학교 대학원

전기 · 컴퓨터 공학부

이 승 열

이 승 열의 석사학위논문을 인준함

2015 년 08 월

Abstract

Cooperative Power Management for Chip Multiprocessors using Space-Shared Scheduling

Seungyul Lee
Department of Electrical Engineering & Computer Science
Collage of Engineering
The Graduate School
Seoul National University

Nowadays, many-core chips are especially attractive for data center operators to provide cloud computing service models. The trend in operating system designs, furthermore, is changing from traditional time-sharing to space-shared approaches to support recent many-core architectures. These CPU and OS changes make power and thermal constraints becoming one of most important design issues. Additional power management methods and core re-allocation techniques are necessary to overcome the limitations of traditional dynamic voltage and frequency scaling (DVFS).

In this thesis, we present a cooperative hierarchical power management for many-core systems running a space-shared operating system. We consider two levels of space-shared system resources: space in the form of cores and physical memory. Recent chip multiprocessors (CMPs) provide group-level DVFS in which the voltage/frequency of cores is managed at the level of several cores instead of every single core. Memory is also allocated by a coarse-grained resource manager to isolate space partitions. Our research reflects these characteristics of CMPs.

We show how to integrate core re-allocation and DVFS techniques through cooperative hierarchical power management. The core re-allocation technique considers the data performance in dependence of the core location. In addition, two important factors are performance loss caused by DVFS and the benefit of core re-allocation. We have implemented this framework on the Intel Single Chip Cloud Computer (SCC) and achieve a 27-32% better performance per watt ratio than naïve DVFS policies at the expense of a minimal 1-2% overall performance loss. Furthermore, we have achieved a 5-11% higher performance than previous research with a migration technique that uses a naïve migration algorithm that does also not consider the migration benefit and data locality.

Keywords: Many-core Architecture, Scheduling, DVFS, Energy Efficiency
Student Number: 2013-23128

Contents

List of Figures

List of Tables

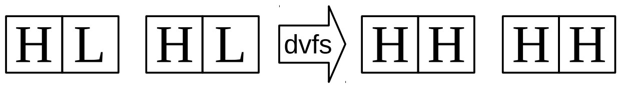
Chapter 1

Introduction

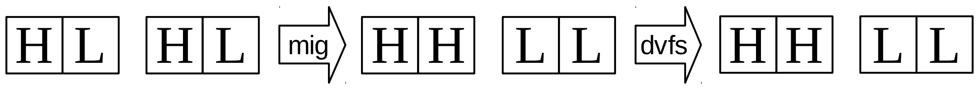
In the past decade, we have seen amazing improvement in transistor integration techniques. The recent trend of CPU architecture has changed from single- or dual-core to multi- or many-core and integrate more and more cores onto one processor die called chip multiprocessors (CMPs) [1, 2, 3, 4]. The CMPs led to chip-level power and thermal constraints to become one of most important design issues and performance limitations. A lot of cores increase energy cost and higher die temperature adversely affect chip reliability and lifetime [5].

Most processors include CMPs provide the dynamic voltage and frequency scaling (DVFS) technique to handle the voltage and frequency levels. The operating system (OS) periodically monitors the load of the processor, the voltage and frequency are scaled to use energy more efficiently. For the multi-core system, each core can be controlled individually but it is too costly in the CMPs [6]. To reduce these overheads, cores are clustered into voltage and frequency domains that leading to multiple-voltage/multiple-frequency (MVMF) designs [7, 8, 5]. All cores are clustered a specific domain have the same power properties; it can reduce the hardware overhead and increase the performance.

Furthermore, the trend in OS designs is changing to space-shared approaches from traditional time-sharing to support recent many-core architectures [9, 10, 11, 12, 13]. This model divides the role of the OS to a coarse-grained resource manager and a runtime library. It could reduce the complexity of runtime scheduling so that it guarantees scalability. The coarse-grained resource manager provides cores and memory allocation and chip-wide power management. The runtime library, on the other hand, provides scheduling of the processes and threads. These two designs isolate applications so they do not interfere with each other.



(a) DVFS



(b) DVFS after migration

Figure 1.1: Simple migration effect to DVFS

This thesis proposes a hierarchical power management architecture for CMPs that runs completely isolated applications such as the OS. Existing power management techniques for CMPs do not suite MVMF designs and clustered domains because the OS could not move to other compositions. We propose that it is possible to change the allocation of the physical core to the clustered domain with zero copy migration on CMPs and could combined with existing DVFS policies. Figure 1.1 (a) shows the result of applying DVFS with considering the clustered frequency domain. Even though there are two high and two low cores, DVFS sets all the cores to high in order to provide the required performance. On the other hand, Figure 1.1 (b) shows the result of DVFS combined with the migration technique. These figures show simplified migration technique. The heuristic core re-allocation algorithm that is based on a cost-benefit buyer-seller model is presented in the thesis. It includes several techniques such as performance loss to get lower energy consumption, migration benefit ratio by evaluating power and migration overhead, and data performances as core location. We have implemented this technique in the Linux operating system running on the Intel Single-chip Cloud Computer (SCC) [14]. As a result, the proposed technique achieves 27-32% better performance per watt ratio than naïve DVFS policies at the expense of a minimal 1-2% overall performance loss. Furthermore, we have achieved 5-11% higher performance than previous research [15] that used a simple migration technique without heuristic re-allocation algorithm and migration evaluation.

The remainder of this thesis is organized as follows: Chapter 2 discusses the related work. Chapter 3 discusses the many-core architecture and gives detailed information. Chapter 4 describes how zero-copy OS migration works. Chapter 5 describes the cooperative hierarchical power management. Chapter 6 discuss the core re-allocation and the DVFS policies. The experimental setup and the results are presented in Chapter 7. Finally, Chapter 8 concludes the thesis.

Chapter 2

Related Work

Recent researches on the power management techniques for CMPs suggest various manner with or without additional hardware. Some of them have considered heterogeneous CMP designs to reduce energy consumption with minimal performance loss. Kumar et al. [16] and Ghiasi [17] proposed power management technique to improve power consumption and thermal management. But these two techniques need hardware design change or additional hardware support. Meisner et al. proposed PowerNap [18] and DreamWeaver [19] that assume hardware support for quick transitions between on- and off-states focused on exploiting idle periods. In case of a frequently changing state, these techniques would lead to longer execution time which in turn reduces the potential to sleep. Our research interested in orthogonal phase that lower frequency by DVFS technique occurs longer execution time. They should consider the performance effect.

A number of researchers have proposed another power management techniques for CMPs [20, 21, 22, 23, 24, 25]. Isci et al. [22] apply different DVFS policies controlled globally under a given power budget. Their best performing policy achieves as good as an oracle policy within very low performance degradation. Ma et al. [23] propose grouping cores with running same application and partition the power budget to these groups. The groups divide power budget to each core and cores scale their frequencies. Meng et al. [24] propose multi-optimization power saving strategy for multi-core power management through run-time adaptation of highly configurable processor cores. Rangan et al. [25] propose ThreadMotion that is fine-grained power-management to migrate applications between cores in a multi-core system. They also need hardware support to quickly migrate threads to another cores.

The work most closely related to ours has been presented by Ioannou et al. [20] and Qiong et al. [21]. Ioannou et al. [20] suggest a hierarchical power manager to scaling the voltages and frequencies for the SCC. Qiong et al. [21] propose the thread shuffling that migrates critical threads to same DVFS domain and scales frequencies for non-critical threads. This method of combining DVFS features and thread migration could reduce the energy of non-critical threads in one operating system. We show that core re-allocation technique regardless of the number of OSes significantly improves the performance per watt ratio than only considered the DVFS and thread migrations without additional hardware support.

Our previous research [15] considered similar hierarchical architecture for migration and DVFS on CMPs, but it handled only CPU overhead and didn't evaluate the result of migrations. In this thesis, we propose the heuristic core re-allocation algorithm that evaluates the migration cost and benefit combined with CPU overhead and memory performance according to core location.

Chapter 3

Many-core Architectures

Many-core architectures exhibit a number of typical characteristics [26] in order to effectively manage and utilize the large number of cores. First of all, global memory addressing to access memory by all cores via memory controllers. Secondly, shared memory to share data or communicate with other cores. Finally, an interconnection network to transmit data to/from the memory or I/O devices. In this chapter, we provide overviews of two kinds of many-core architectures; the Intel single-chip cloud computer (SCC) and the TILE-Gx8036 processor (Tilera). In this thesis, we focus on the SCC architecture and implement our cooperative power management on it.

3.1 The Intel Single-chip Cloud Computer

3.1.1 Architecture Overview

Intel Labs has created the experimental SCC for many-core research. They integrated 48 cores consist of Intel P54C Pentium® on a silicon CPU chip that interconnected by network-on-chip (NoC). Each core has 16KB *L1 caches* and 256KB *L2 caches* that there is no cache coherence. SCC incorporates tech-

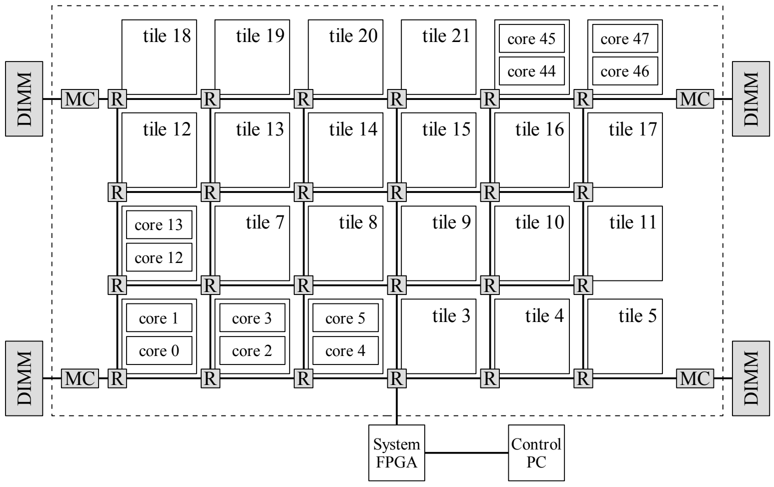


Figure 3.1: Intel SCC block diagram

nologies such as *advanced power management* and additional support for 16KB *message-passing buffer*. Two cores are grouped together to form a *tile* that share network resource, clock management technique, interrupt handling and other system features. Each tile connected router mesh network with 256GB/s bit-section bandwidth. Four memory controllers located in the four corners of the mesh network to access to maximum 64 GBs of memory. The FPGA is the bridge between the SCC and the management-console PC (MCPC) provides allow the environment setup and communication each other. Figure 3.1 shows the SCC block diagram.

3.1.2 Memory Addressing

There are four memory controllers on the SCC mesh could access physical memory, and each core uses the 32-bit physical address that is not enough to addressing the entire 64 GBs system memory. To access system memory via memory controllers, it needs additional address translation.

Each core has 256-entry lookup table (LUT) that provide additional information to access memory controllers and system configuration registers. The top 8-bit of core-level address indicates LUT index could get 22-bit information; a `bypass` bit, an 8-bit `destID`, a 3-bit `subdestID`, and a 10-bit `system address extension`. The `bypass` bit is provided for local tile memory buffer access. The 8-bit router destination ID (`destID`) designates one of the four memory controllers (MC). The 3-bit intra-tile sub-ID (`subdestID`) determines selecting the memory controller, the voltage regulator controller, or the system interface. The 10-bit `system address extension` is prepended to the remaining 24 bits of the core address to form a 34-bit address. Figure 3.2 illustrates the core-to-system address translation.

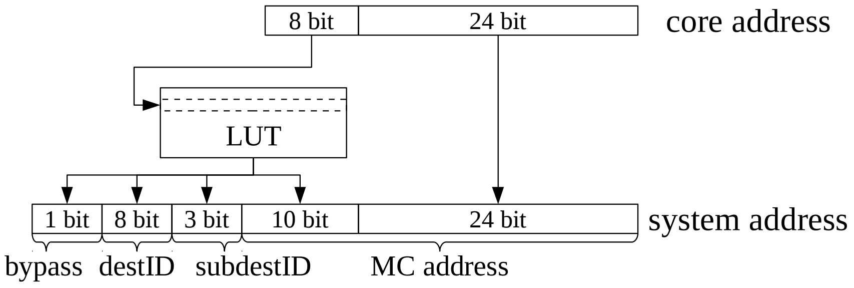


Figure 3.2: Core-to-system address translation on the SCC

In addition to address translation, there is another characteristic of LUT entries. Front 8-bit of core address makes 256 LUT entries and one entry maps to 16 MB of memory. In our configuration, the 8 GBs per memory controller are divided to the 12 cores located closest to the controller; i.e., the physical memory space of one core is mapped using 42 LUT entries by default. We use these 42 entries to migrate memory information in the Chapter 4.

3.1.3 DVFS Capabilities

The SCC provides voltage and frequency scaling technique for the domains and the NoC. Scaling frequency for *tile* that is clock domain, all cores in the same tile run at the same frequency. The *voltage domain* that is a group of four tiles control the voltage and it affect to all tiles inside of voltage domain. Figure 3.3

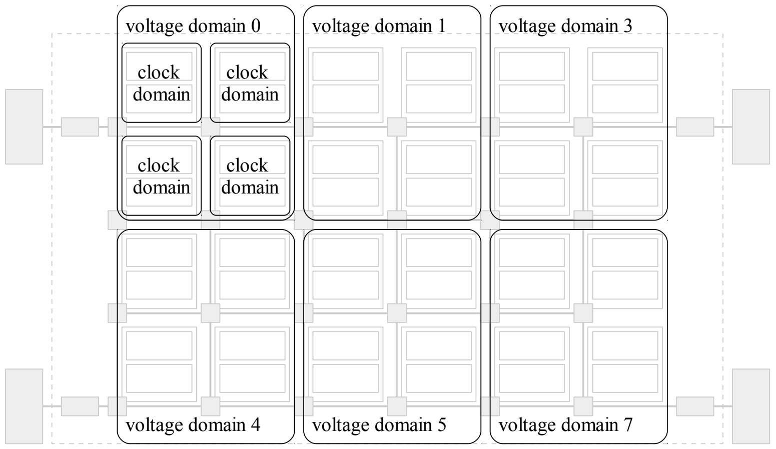


Figure 3.3: Voltage and clock domains on the Intel SCC

illustrates the clock and voltage domains on the SCC. Voltage domains 2 and 6 are not shown in this figure because they work for the NoC and the system interface.

To control the voltage and frequency, the SCC uses specific registers: a voltage regulator controller (VRC) register and a system configuration registers. The system configuration registers change frequencies in few clocks. Voltage change, by the way, may take up to 30ms and several voltage changes must be serialized because it uses only one VRC register.

The SCC supports seven different supply voltage levels from $1.1V$ to $0.7V$, however, only four are of practical interest: $1.1V$ to run at a frequency of 800MHz, $0.9V$ to run at 533MHz, $0.8V$ for 400MHz, and $0.7V$ for frequencies below 400MHz. The frequency is set by writing a divisor between 2 and 16 for the 1600MHz clock resulting in core frequencies from 800MHz to 100MHz. Table 3.1 shows the available voltage and frequency settings.

Voltage (V)	Frequency (MHz)
1.1	800
0.9	533
0.8	400
0.7	320, 266, 228, 200, 178, 160, 145, 133
	123, 114, 106, 100

Table 3.1: Voltage and frequency settings on the SCC

3.2 Tilera

3.2.1 Architecture Overview

Tilera's TILE-Gx processor [27] is based on the MIT alewife project containing cache-coherent, distributed shared memory and user-level message-passing concepts. Tile-Gx8036 processor is one of TILE-Gx processors consisting of thirty-six 64-bit RISC cores. Each core runs at 1.2GHz and three way VLIW process with 12 Mbytes 3-level coherent on-chip cache architecture. There are two 72-bit DDR3 DRAM controllers with ECC. And it provides 40Gbps of integrated Ethernet I/O. Figure 3.4 shows the Tilera block diagram. Tilera's TILE-Gx processors archive isolation through the so-called multicore hardwall [28].

3.2.2 Memory Architecture

The memory architecture of Tilera defines a flat, globally shared 64-bit physical and virtual address space. Tile-Gx processors implement a 40-bit subset physical address and provide the mechanism by which software running on different tiles, and I/O devices, share instructions and data. Page tables are used to translate virtual-to-physical addresses. The virtual address is architecturally 64-bit, but is used 42-bit in the Tile-Gx processor. Virtual addresses consist of three parts, which are two legal VA regions, lower and upper, and an illegal region in the middle, as shown in Figure 3.5.

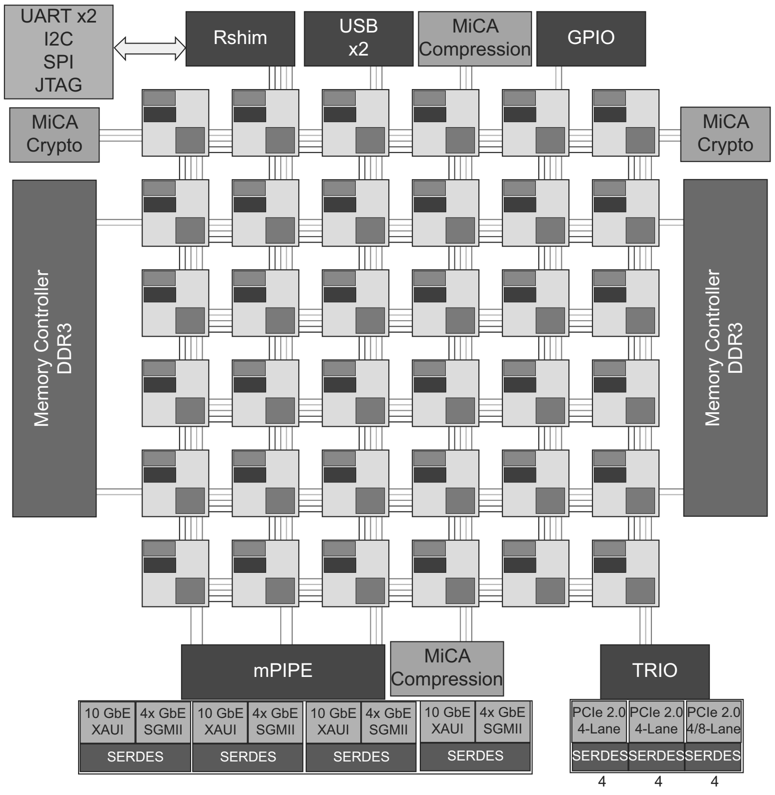


Figure 3.4: TILE-Gx36 SoC block diagram (source: Tilera Architecture Overview [29])

The translation process verifies the protected regions of memory and designates the page of physical address; coherent, non-coherent, uncacheable, or memory mapped I/O (MMIO). Recently-accessed values are stored in cache location in each tile for coherent and non-coherent pages. However, uncacheable and MMIO addresses are never stored into a tile cache.

3.2.3 Switch Interface and Mesh

Tilera inter-network communication within the tile array takes place over the iMesh^{TM} Interconnect shown in Figure 3.6. The iMesh^{TM} Interconnect provides two classes of networks that support low and high bandwidth commu-

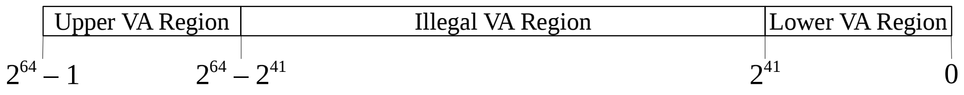


Figure 3.5: Virtual Address Space on the Tilera

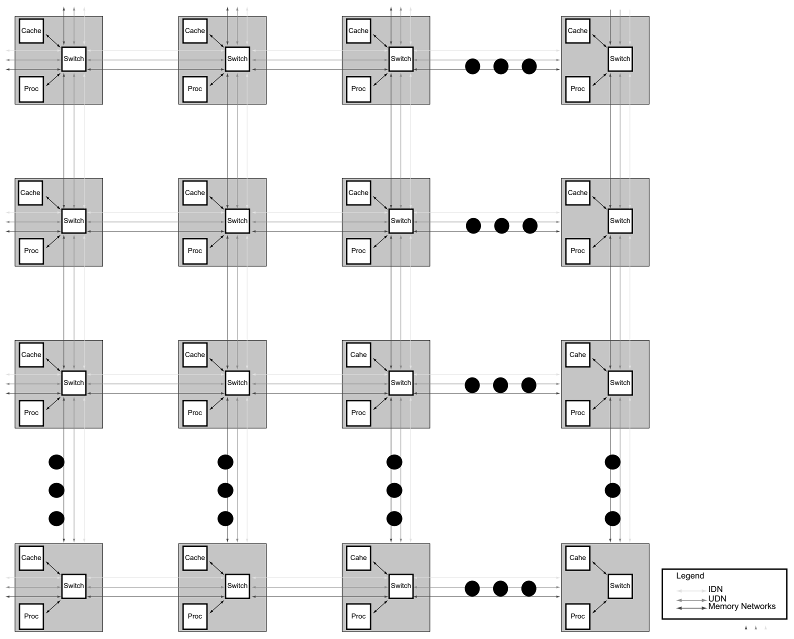


Figure 3.6: TILE-Gx Switch Interfaces (source: Tilera Architecture Overview [29])

nication. The first class comprises a set of software visible networks for application level streaming and messaging. Another class used by the memory system to handle memory requests, exchange cache coherency commands and support high performance shared memory communication. Dedicated switches are used to implement the iMesh Interconnect, allowing for a complete decoupling of data routing from the processor.

Chapter 4

Zero-copy OS Migration

Variety workloads on cores could make inefficient use of the CMP. In order to optimize the execution of workloads towards a specific goal, often need to move the workloads to another core. Such goals include, but are not limited to even heat dissipation and adherence to a given power budget. For the heat reduction, workloads of high temperature cores move to other idle cores. The second case is motivated by the need to cluster workloads with similar performance requirements in voltage and/or frequency domains to achieve optimal power usage during operating DVFS. The task migration techniques on the operating system also provide the similar effect, but the space-shared environment on CMPs needs another workload management manner because task migration could not handle the workload over the operation system. We suggest, therefore, OS migration technique on the CMPs and describe the necessary steps in the following sections.

4.1 Cooperative OS Migration

Moving an OS to another physical core can be implemented two types that are with or without the cooperation of the migration. The migrated OS should enter a safe state to move to the newly assigned core and resume with the co-operative setting. Transparent migration, on the other hand, does not need interactions and knowledge of the OS.

The main risk of migration is how to deal with the volatile state, i.e., the assigned memory and last values of registers. If the CMP uses only global shared memory address, it does not need to consider memory migration. However, memory migration is still critical issue because each core use the one of memory controllers depending on core location. Section 4.3 would discuss these volatile state.

4.2 Migration Steps

In the hierarchical architecture, zero-copy OS migration is orchestrated by a migration manager that is part of the *chip controller*. The steps are illustrated in Figure 4.1. It reveals that migration is, in fact, rather a *circular swap* of two (or more) OSes rather than a unidirectional migration from one core to another. Since we require a minimal amount of cooperation from all involved cores, we assume that a cooperative OS runs on all (including the currently unused) cores. (1) The migration manager sends the signal on the form of an interrupt to the cores who involved in the migration. (2) This interrupt is handled by the cooperative OS' interrupt handler which saves the necessary registers into a per-core designated memory area. (3) After all registers have been saved, the affected cores signal completion to the migration manager and completely flush their caches. (4) The migration manager then stops all cores involved in the migration by gating their clock, and (5) swaps the cores' register values and the memory mappings. (6) The migration manager signals completion of the

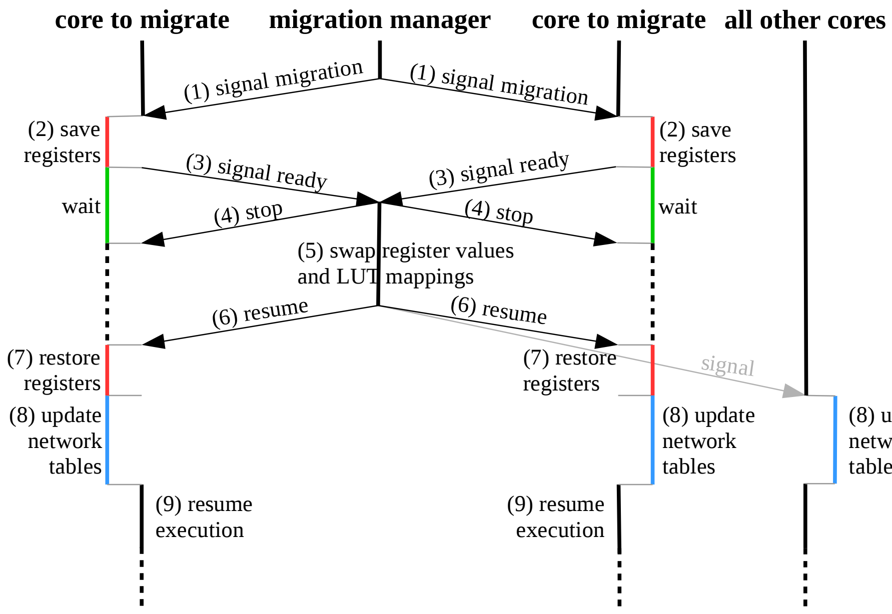


Figure 4.1: OS Migration Steps

migration by resuming the clock on the migrated cores. (7) The cores proceed by restoring the (new) register values from memory, exit the interrupt handler and (8) update internal network routing tables. Finally, (9) each core who involved in migration resume their operation. In addition, all cores who do not involved in migration need to update internal network routing tables to reflect the new locations of the cores (see Section 4.4).

4.3 Migration Volatile State

The cores who are involved in the core re-allocation phase save and restore their register values to a designated memory area. After saving registers, the migrated cores flushing all caches and enter a busy loop for migration. It is impossible to set the program counter to the exactly correct instruction after resume cores because we don't know what instruction was executing in the busy loop when the clock was gated. However, it can be ignored by assuring the busy loop.

The memory of the migrated cores also need copying to another cores, but its costs are too huge. Instead of copying all memory area, copy the corresponding entries in the LUT that translates physical memory address to system memory address. The migrated cores could access their private memory by copying 42 LUT entries as mentioned Chapter 3.1.2.

4.4 Networking

There are two types of network exist on the SCC. For the on-chip network, the SCC uses interrupt and MPB. The network sender put data to MPB then send the interrupt signal to network receiver. The network receiver fetches the data from the sender's MPB in the interrupt service routine. Instead of changing this MPB location, migration phase updates IP-to-coreID mappings. To send data, sender make interrupt according to the IP-to-coreID mapping table. Therefore, it does not need complex MPB change because every cores update their mapping table on last step of migration.

The subnet for communication with the MCPC, on the other hand, used eMAC interface. Internally all eMAC modules have a 4Kbyte FIFO buffer for the TX and RX directions and a connected to the FPGA router via the client interface. The sending core writes its frames into the buffer, informs the FPGA hardware that new frames are present. The hardware then pulls the frames from the buffer and transfers them to the external ethernet ports. When the HW receives frames on the external ethernet port, it puts them into the corresponding memory buffer, tells the core that new frames are present and the receiving core starts pulling the frames from the buffer. Signaling can be done sending interrupts to the core. The cores involved in the migration need to change the configurations of eMAC. When the migration is finished, they read the mac address from eMAC general configuration registers then change their configurations.

Chapter 5

Cooperative Hierarchical Power Management

A global power management technique for CMPs is hard to controlling entire chips because there are different frequency and voltage domains. If the policy is simple, it could get low energy effect with small calculation and vice versa. The hierarchical structure is an alternative management technique for these domains characteristics on CMPs. This chapter describes the organization of the cooperative hierarchical power manager in detail.

5.1 Cooperative Core Re-Allocation

Space-sharing provides more opportunities for application-specific runtimes to allow efficient user-level scheduling within an application container. This design, however, hinders chip-level optimizations such as global load balancing or power management since the coarse-grained OS has no control over which core an application task is executed on. If orchestrated properly, the architecture of CMPs, however, allows for dynamic resource (core) re-allocation with no or very little application-runtime involvement. The idea is to reclaim a core from an application container and assign it to another partition without the application-specific scheduler knowing about it.

Core re-allocation combined with DVFS allows the power manager to group cores with similar performance requirements into the same frequency/voltage domains and then select a close-to-optimal frequency/voltage for these domains. The re-allocation operation is a circular re-assignment of cores $c_0, c_1, \ldots, c_{k-1}$ belonging to the application containers $a_0, a_1, \ldots, a_{k-1}$ with $k \geq 2$ where core c_i is assigned from $a_i \rightarrow a_{(i+1) \mod k}$. That is, the number of cores per application does not change and can be done transparently to the application-specific runtime. Re-assigning a core c_i from application container $a_i \rightarrow a_{(i+1) \mod k}$ requires flushing the local caches, save the volatile state of the core c_i, and finally restore the state of core $c_{(i+1) \mod k}$.

The volatile state includes only the core registers; physical memory does not need to be copied since CMPs provide a global memory address space. Without explicit hardware support, it is impossible to read a core's volatile state from outside a core. With a minimal level of cooperation by the application-specific runtime executing on each core it is possible to emulate the missing hardware support by explicitly saving and restoring the volatile state of a specific core to global shared memory as discussed in the Section 4.3.

5.2 Hierarchical Organization

The logical structure of the hierarchical power manager reflects the structure of the CMP with separate frequency and voltage domains (Figure 5.1). At the lowest level in the hierarchy are the *core controllers* that represent a single core. The second level, the *frequency controllers*, represents a frequency domain with m individual cores all running at the same frequency. The *voltage controllers* at next level constitute a voltage domain. At this level, voltage changes are initiated. The top level in the hierarchy, finally, is represented by the *chip controller* and models the entire chip. Each level only communicates directly

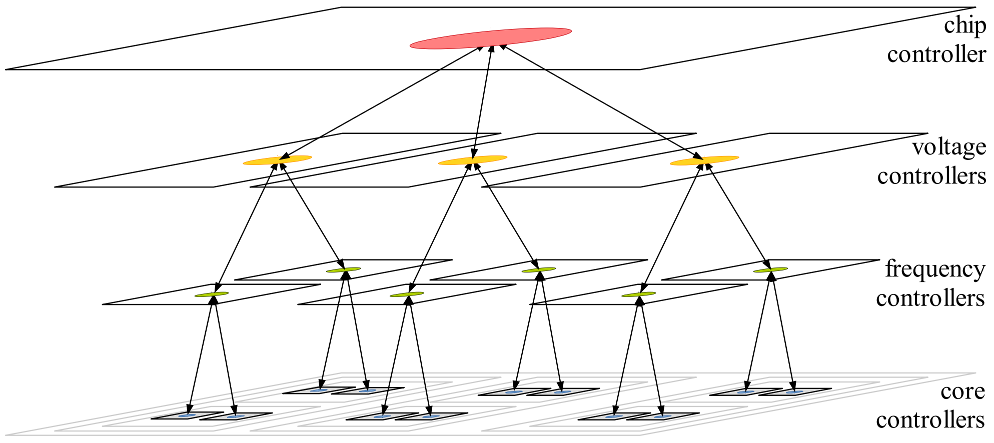


Figure 5.1: Logical abstraction of the hierarchical power manager and the mapping to the physical structure.

with the level above or below, i.e., the clock domain manager interacts with the voltage domain manager.

- **Core Controller:** The task of the core controllers is to monitor and predict the performance of the associated core. Without explicit hardware support to externally read performance counters of single cores, a locally-running software agent is required on each core. This is the role of the *core controller* that monitors the current performance of its core through periodically querying the core's performance monitoring units (PMU). The core controllers also predict the required computational performance based on extrapolated measured data as is common in DVFS policies. At regular intervals, the core controllers communicate with their frequency controllers. Data received from the frequency controller includes the additional information to using in DVFS policies. Data sent upstream comprises the estimated required frequency of the core. Depending on the load factor, the core controller requests a higher, the same, or lower frequency from the frequency controller.

- **Frequency Controller:** For each frequency domain, the *frequency controller* gathers data about the requested frequencies from its core controllers, and processes and forwards that data to the voltage controller. The frequency controllers also compute and set the operation frequency of the domain. The clock frequency is constrained by the current voltage level of the corresponding voltage domain and computed based on the requested frequency levels reported by the core controllers and the currently active DVFS policy (see Section 6.4). Voltage and frequency changes require careful coordination: if the voltage of a voltage domain is to be lowered, the frequencies of all cores within that domain need to be lowered below the maximal supported frequency at that voltage *before* the voltage change is performed. In the opposite case, the frequencies can only be increased *after* the voltage level of the superior voltage domain has been raised.

- **Voltage Controller:** The *voltage controller* gathers data from its frequency controllers, and forwards it to the global chip domain. The voltage controller also computes and sets the operating voltage of its domain, in close collaboration with the frequency controllers as discussed in the *frequency controller.*

- **Chip Controller:** The *chip controller* uses the processed frequency and voltage requests from the subordinate controllers to compute a core assignment that allows more optimal DVFS settings at the voltage and frequency domain level. The core assignment is then translated into a series of circular core re-assignment units, and the cores are migrated. Once migration has completed, the chip controller notifies the voltage controllers which then proceed and orchestrate the frequency/voltage changes.

Chapter 6

Core Re-Allocation and DVFS Policies

This thesis focuses on optimizing the *performance per watt* ratio of the overall chip through the combination of the core re-allocation and DVFS policies. The *core re-allocation* policy uses the *core migration* technique as discussed in Chapter 4. Other policies, such as, heat dissipation or adhering to the given power budget, can also be implemented within the framework.

Without core re-allocation, cores are pinned to their applications. For voltage domains containing both very busy cores and idle cores there is no optimal voltage setting: if the voltage is too low, the idle cores run at the optimal frequency but the performance of the busy cores is severely affected because the low voltage prevents the frequency domain controller from selecting the required frequency. On the other hand, if the voltage is set high enough to satisfy the performance needs of the busy cores, the idle cores waste energy because they operate at a higher than necessary voltage.

In this chapter, we describe the core re-allocation considerations and algorithm, employed DVFS techniques, and other considerations.

6.1 Core Re-Allocation Considerations

The core re-allocation should occur before applying DVFS techniques because migrated cores affect to frequencies and voltages. The probability of migration occurrence, voltage, and frequency changes is determined by the cost of these operations; the time for migration is largely unaffected by the number of cores begin migrated because all involved cores can store/restore the volatile state in parallel. Migrated cores are stopped and have their caches flushed while unaffected cores continue to run during the migration process. Voltage changes are quite an expensive operation because the clock of all affected cores (i.e., whole cores in one voltage domain) is stopped during the rather long voltage adjustment. Frequency changes, on the other hand, are almost instantaneous and can thus be often performed. On the SCC specifically, it has measured latencies of each operation: $\leq 20ms$ for migration and $\leq 30ms$ for voltage changes. On this particular architecture, migration is cheaper than voltage changes. In addition, the SCC only supports one voltage change at a time: i.e., different domains cannot change the voltage in parallel.

Core re-allocation enables consolidation of cores with similar performance requirements into one voltage/frequency domain. This allows setting the voltage/frequency of the domain to a value that is close to the optimal value for most involved cores. As an application's computational requirements change during execution, the core re-allocation is invoked at periodical intervals.

A naïve algorithm would be to sort the cores by their performance requirements and then assign them in order to the voltage and frequency domains. While the resulting allocation of cores to domains is optimal for CPU-bound applications and one time quantum, this algorithm fails to consider the overhead of core re-allocation. The re-allocation of a core itself is very quick (measurements on a real system yield an overhead of $\leq 3ms$), each time a core is re-allocated the application's processes executing on the newly assigned core experience a

Frequency	**Dist 0**	**Dist 1**	**Dist 2**	**Dist 3**	**Dist 4**	**Dist 5**	**Dist 6**	**Dist 7**	**Dist 8**
800MHz	100	95.7	88.1	84.7	81.6	76.0	73.5	71.0	66.7
533MHz	87.9	81.5	78.5	75.8	73.2	69.9	66.6	63.8	61.1
400MHz	78.6	73.3	73.2	68.7	66.6	64.7	61.1	59.4	56.4
320MHz	73.2	69.3	66.0	62.8	62.8	58.7	56.7	54.9	53.3
200MHz	57.7	54.9	52.4	50.0	50.0	47.7	47.7	45.7	45.7
100MHz	36.5	34.4	34.4	34.4	32.2	32.2	32.2	30.6	30.6

Table 6.1: Normalized memory throughput according to frequencies and distances from memory controller

lot of cold misses in the local instruction and data caches which will lead to both a performance loss as well as increased memory traffic. A good core re-allocation algorithm must thus also consider the current core assignments and minimize the number of migrations.

Another consideration is the location of the core on the grid. The effect of re-allocation a core to an application container is in fact that the application container executes its tasks on the newly assigned cores since the location of the physical cores on the CMP is immutable. Such a re-assignment can have significant effects on the access latency and bandwidth of memory accesses, namely, if a memory-intensive application originally executing on a core close to the memory controller holding the required data is moved to a core located far away from said memory controller. Table 6.1 shows memory throughput depend on the frequencies and the distances from the memory controller. As this result, the performances according to the distance from the memory controller are quite different on the same frequency; i.e., if a core running the memory-intensive application with 800MHz frequency migrates from besides of memory controller to corner of opposite position, its performance might drop down to 66.7%. Migrating to near the memory controller, furthermore, could get higher memory performance than raise frequency of running core.

6.2 Core Re-Allocation Algorithm

We employ a buyer-seller heuristic where, for a given target frequency f_{target}, the domain controllers put up for sale (by adding to their respective `Sell` list) (a) all frequency domains which request a frequency that can be run at a lower voltage than v_{target} that set the highest voltage value in the first iteration and reduce 1 voltage level for each iteration, i.e., $v_{req}(tile) \leq v_{target}$, and (b) all single cores that require a voltage smaller than v_{target} but are co-located with other cores that requests a higher voltage, $v_{req}(core) \leq v_{target}$. $v_{req}(core) > v_{target}$ means that this core already migrated or kept in previous iteration. On the `Keep` list frequency domains and cores that request voltage v_{target}, that is $v_{req}(tile/core) = v_{target}$, are included. As an illustrative example on the Intel SCC, consider Table 6.2 showing the `Keep` and `Sell` lists for the configuration shown in Figure 6.1 (a). Note that the `Keep` list does not list the core which requests a lower frequency but instead the co-located core from the same frequency domain. Consider `vdom0`: the two frequency domains at the top are expected to require the frequencies 8 and 5. The `Keep` list of `vdom0` then contains two single core entries $(5), (5)$, expressing that it contains two tiles with one core each already running at the target frequency that it wants to keep. It offers to buy two single cores running at v_{target} and can in return offer two single cores running at frequency 5. On the `Sell` side, on the other hand, the actual cores are listed. `Sell` for `vdom0` contains the two single core entries $(5), (5)$, representing the fact that `vdom0` is offering two single cores expected to run at frequency 5. The reason for this somewhat inconsistent notation is that it is then straightforward to match single cores on the buyer list with those from the seller side by comparing the requested/offered voltage levels.

After initializing the `Keep/Sell` lists, the buyer-seller algorithm runs. The algorithm repeatedly selects two frequency domains or single cores to swap based on the information in the `Keep/Sell` lists until no further changes occur.

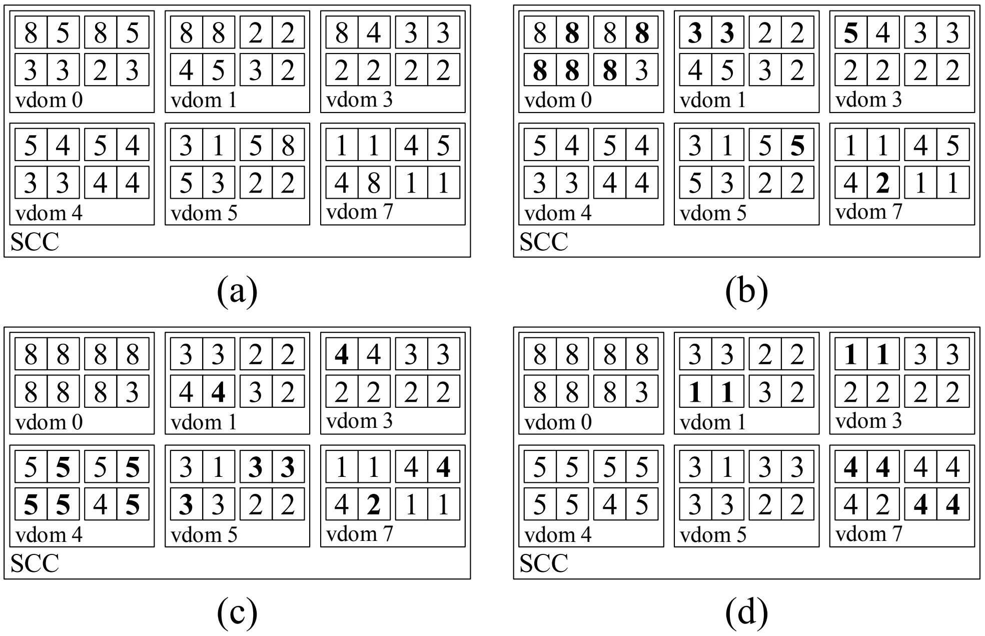


Figure 6.1: Buyer-seller algorithm: (a) initial configuration, (b) configuration after running the first round for $v = 8$, (c) configuration after running the second round for $v = 5$, (d) final configuration after running the last round for $v = 4$. Bold values represent frequency domains/cores migrated in that iteration.

In each repetition, the voltage domain that offers the fewest tiles for sale and as its counterpart the domain that tries to keep the fewest tiles are selected. In the first round for $v_{target} = 8$ the domains `vdom0` and `vdom1` are chosen (Table 6.2). `Vdom0` offers only two tiles for sale which means that it tries to keep the other two. `Vdom1` is the only domain containing a tile running at the target frequency v_{target}. The algorithm pairs the two domains up with `vdom0` representing the buyer and `vdom1` the seller. When selecting a tile to exchange on the `Sell` side, the tile that most closely matches the average frequency of the seller *after* selling the tile running at v_{target} is chosen. In the example at hand, `vdom1` contains the frequencies $2, 2, 4, 5, 3, 2$ after giving tile $\{8, 8\}$ away with an average of 3.2. Tile $\{3, 3\}$ in `vdom0`'s seller list is closest to this value and

vdom	`Keep`	`Sell`
0	$(5),(5)$	$\{3,3\},\{2,3\},(5),(5)$
1	$\{8,8\}$	$\{2,2\},\{4,5\},\{3,2\}$
3	(4)	$\{3,3\},\{2,2\},\{2,2\},(4)$
4		~~$\{5,4\},\{5,4\},\{3,3\},\{4,4\}$~~
5	(5)	$\{3,1\},\{5,3\},\{2,2\},(5)$
7	(4)	$\{1,1\},\{4,5\},\{1,1\},(4)$

Table 6.2: `Keep/Sell` lists for the configuration given in Figure 6.1 (a). `vdom4` will not participate in this round of the algorithm because it does not try to keep any frequency domains/cores at all.

is thus selected and exchanged with tile $\{8,8\}$ from `vdom1`. This operation also updates the domains' `Keep/Sell` lists. This process is repeated until no more tiles can be exchanged, then the algorithm proceeds to swap single cores. First, again the domain with the fewest frequency domains to sell is chosen, then the `Keep` lists of all other domains are searched for a matching value. Keep in mind that the actual core to be exchanged is a core running at frequency v_{target} and the entry in the `Keep` list represents the frequency of the sibling in the same frequency domain. Again, this process is repeated until no further cores can be exchanged. Cores of frequency domains on the `Keep` list can be split up into single core entities if there are single cores to be sold but the `Keep` lists only contain entire domains. In our running example, again `vdom0` is chosen as the seller domain since it contains the fewest tiles to be sold (one after the tile exchange). The core to be sold runs at frequency 5; `vdom5` is offering a core running at frequency 8 and would like to get one running at 5 in return. The two are thus exchanged, and the `Keep/Sell` lists updated.

Figure 6.1 (a) displays the estimated frequencies for each core before the buyer-seller algorithm starts. Figures 6.1 (b) - (d) show the layout after each repetition for $v_{target} = 8, 5$ and 4, respectively; (d) represents the final configuration.

6.3 Evaluation of Core Re-Allocation

The last step of the core re-allocation is to compute the expected benefit of the migration plan. The energy for the next time quantum t of the status quo is computed as

$$E_{status\_quo} = P_{status\_quo} \cdot t \tag{6.1}$$

where P_{status_quo} can be obtained from the on-chip sensors or, in the absence of such, from power measurements obtained offline for each frequency. The expected energy consumption if the migration is performed is given as follows

$$E_{migrated} = P_{migrated} \cdot (t + O_{migration} + O_{memory}) \tag{6.2}$$

$$O_{migration} = t_{migration} + t_{cache\_fill}(f_{target}) \tag{6.3}$$

$$O_{memory} = t \cdot \frac{throughput(status\_quo)}{throughput(migrated)} \tag{6.4}$$

where $P_{migrated}$ is computed based on the power prediction formula with offline profiled chip capacitance values for each frequency level on the SCC as below

$$P_{dynamic} = K_{activityfactor} * C_{chipcapacitance} * V_{voltage}^2 * f_{frequency} \tag{6.5}$$

The migration overhead, $O_{migration}$ is the overhead incurred by the actual migration and the (worst-case) time required to re-fill the entire caches at the target frequency f_{target}. The memory overhead, O_{memory} captures the sensitivity of an application to the location of the assigned core(s) on the CMP. The maximum memory throughput at each frequency and core location is profiled once offline shown in Table 6.1; the required throughput of an application based on the core's last-level cache misses (as obtained by the core controllers).

The migration plan is only executed if the following equation holds

$$E_{status\_quo} > E_{migrated} \cdot (1 + \Delta m) \tag{6.6}$$

that is, the expected benefit of migration has to be above a certain threshold Δm.

The buyer-seller algorithm returns the instructions to perform the actual migration in form of several circular lists of cores that are to be migrated. This list is then processed by the migration manager as discussed in Section 4.2.

6.4 DVFS Policies

We employ two DVFS policies in the hierarchical power manager for CMPs proposed by Ioannou *et al.* [20]. Their work has been implemented and evaluated on the same hardware and thus provides a good reference point.

- `Allhigh`: set clock frequency of all cores within a voltage domain to the highest requested frequency.
- `Tile`: each clock domain chooses higher requested frequency within involved cores. A voltage domain sets own voltage level for the highest clock frequency. Note: in [20] this policy is denoted `Simple`.

We do not employ the `Alllow` and `Allmean` policies since they sacrifice too much performance in return for power savings.

Chapter 7

Experimentation and Evaluation

7.1 Experimental Setup

All experiments were conducted on the Intel Single-chip Cloud Computer. The chip controller and other OS services such as monitoring logging, run on dedicated cores in voltage domain 3. The applications containers run modified versions of sccLinux in one of the remaining domains. We chose this separation in order to separate the power consumption of the core OS from the application containers, voltage domain 3 does not participate in core re-allocation. However, the SCC only allows measuring the *total chip power*; the power consumption of the OS services are therefore also included in all results.

The benchmark scenarios consist of a varying number of single-core application containers executing different profiled workloads. The potential of the proposed technique is demonstrated on a synthetic benchmark; profiled workloads by monitoring desktop computers of 20 users over a period of several months and webserver access logs from the soccer World Cup in France 1998 [30].

The baseline of the experiments is obtained by running the benchmark scenario on the SCC at full speed (800MHz) with no power management enabled.

Unlike the work in [20] we do not use a phase-detector based on message passing since we are aiming at independent OSes running on a CMP. Instead, we apply the workload prediction method based on a weighted average. We compare the DVFS policies of [20], `Allhigh` and `Tile`, against the same policies with core re-allocation. In addition, we consider the additional techniques that the lower performance for lower power consumption and migration benefit threshold to get higher migration effects. The next section discusses about these loss and benefit.

The SCC provides a number of voltage and ampere meters on-board. The total power consumed by the SCC chip is obtained by multiplying the (constant) supply voltage with the supply current for the entire SCC chip. The power consumption of individual voltage domains cannot be computed because only the per-domain supply voltage is available but not the current consumed by the domain. We thus always report the total chip power in our experiments in Section 7.3.

The sensors and meters can be read by directly querying the system FPGA from a core in the SCC. The management console also can be read through a board management microcontroller (BMC) that is connected by PCI-Express cable to the SCC. We choose the BMC approach because it does not affect the SCC core's operation.

7.2 Power Management Considerations

To get the more efficient effect of the DVFS and the migration, we consider performance loss for lower power consumption and migration benefit for better performance per watt ratio. Memory performance according to the core location, furthermore, is the important factor for core re-allocation. In this section, we describe how obtain these thresholds and evaluate core re-allocation with or without data-location aware technique.

BM	Acceptable performance loss (Δp)											
	0%			10%			20%			30%		
	Power	Perf	PPW	Power	Perf	PPW	Power	Perf	PPW	Power	Perf	PPW
Profiled1	98.9	99.8	101.0	97.7	99.8	102.1	92.3	99.6	107.9	87.8	98.7	112.4
Profiled2	93.3	98.9	106.0	92.0	98.7	107.3	92.6	98.5	106.4	87.8	97.4	110.9
Profiled3	93.2	99.0	106.2	92.2	98.8	107.1	91.2	98.4	107.8	87.0	97.4	112.0
Average	**95.1**	**99.2**	**104.4**	**94.0**	**99.1**	**105.5**	**92.0**	**98.8**	**107.4**	**87.5**	**97.8**	**111.8**

Table 7.1: Result for performance penalty with Allhigh DVFS policy

BM	Acceptable performance loss (Δp)											
	0%			10%			20%			30%		
	Power	Perf	PPW	Power	Perf	PPW	Power	Perf	PPW	Power	Perf	PPW
Profiled1	92.5	97.3	105.1	86.8	96.8	111.5	86.8	96.4	111.1	87.6	94.7	108.0
Profiled2	91.6	96.4	105.2	89.5	95.7	107.0	90.0	95.0	105.5	87.1	93.8	107.7
Profiled3	86.4	94.2	109.1	84.7	93.7	110.6	84.7	92.3	109.0	81.5	89.9	110.2
Average	**90.2**	**96.0**	**106.5**	**87.0**	**95.4**	**109.7**	**87.2**	**94.6**	**108.5**	**85.4**	**92.8**	**108.6**

Table 7.2: Result for performance penalty with Tile DVFS policy

7.2.1 DVFS Performance Loss

One of DVFS' purposes is how to use energy more efficiently. There is a trade-off between the performance and the power consumption. If we allow some performance loss, we can get the energy benefit. To determine Δp as performance loss threshold with this trade-off, we have experimented with the performance penalty with 3 profiled benchmark scenarios that would be explained in Section 7.3. The performance penalty means DVFS policies request lower frequency than they really need. Even though we set Δp as 30%, it does not mean performance drop to 30% because requested frequency is set same level in many cases; i.e., DVFS policies require 800MHz even if they need 560MHz with 30% loss that is higher than 533MHz (Table 3.1 shows frequency levels). We have experimented with this lower rate (Δp) as 0%, 10%, 20% and 30%.

Table 7.1 and table 7.2 show results of these experiments and it obviously appears trade-off. If we choose large performance penalty, it might use less

BM	**Minimal migration benefit (Δm)**																			
	$< 5\%$				**$\geq 5\%$**				**$\geq 10\%$**				**$\geq 15\%$**				**$\geq 20\%$**			
	Pow	Perf	PPW	Mig	Pow	Perf	PPW	Mig	Pow	Perf	PPW	Mig	Pow	Perf	PPW	Mig	Pow	Perf	PPW	Mig
Profiled1	78.6	99.0	126.0	600	78.8	99.5	126.2	273	83.1	97.46	117.2	177	80.5	97.3	120.8	69	82.9	96.5	116.5	47
Profiled2	73.1	98.1	134.1	331	72.1	98.3	136.3	220	73.7	97.5	132.3	150	75.6	98.2	130.0	79	81.2	97.4	120.0	44
Profiled3	71.0	99.2	139.7	300	66.4	97.5	146.8	188	71.5	97.6	136.6	154	70.7	98.5	139.2	127	88.1	97.9	111.1	78
Ave	**74.2**	**98.8**	**133.3**	**410**	**72.4**	**98.4**	**136.4**	**277**	**76.1**	**97.5**	**128.7**	**160**	**75.6**	**98.0**	**130.0**	**92**	**84.1**	**97.3**	**115.9**	**56**

Table 7.3: Result for migration benefit with Allhigh DVFS policy

BM	**Minimal migration benefit (Δm)**																			
	$< 5\%$				**$\geq 5\%$**				**$\geq 10\%$**				**$\geq 15\%$**				**$\geq 20\%$**			
	Pow	Perf	PPW	Mig	Pow	Perf	PPW	Mig	Pow	Perf	PPW	Mig	Pow	Perf	PPW	Mig	Pow	Perf	PPW	Mig
Profiled1	74.0	84.4	114.2	682	77.3	92.7	120.0	462	75.6	90.4	119.6	394	78.3	93.2	119.1	170	81.0	95.2	117.5	98
Profiled2	68.9	86.6	125.8	404	69.7	92.9	133.2	288	72.0	93.5	129.8	218	74.6	95.1	127.4	119	74.6	93.4	125.2	64
Profiled3	64.8	93.9	144.9	332	67.3	94.3	140.2	233	69.9	94.3	135.0	200	69.7	95.9	137.6	123	78.5	96.9	123.4	112
Ave	**69.2**	**88.3**	**128.3**	**473**	**71.4**	**93.3**	**131.1**	**328**	**72.5**	**92.7**	**128.1**	**271**	**74.2**	**94.7**	**128.0**	**137**	**78.0**	**95.2**	**122.0**	**91**

Table 7.4: Result for migration benefit with Tile DVFS policy

energy but deteriorating performance. The result of 30% performance penalty shows the lowest power consumption but performance degradation is highest. We decide 10% as Δp that drops performance as little as possible and keep higher the performance per watt ratio in these results.

7.2.2 Migration Benefit

There are a number of core re-allocation effects that mentioned previous sections. One of these effects is gathering high frequency cores that need the same voltage. Distributed high frequency cores make several voltage domains to the high value. So this effect reduces the energy wastes because only gathered voltage domains use high voltage. But sometimes this benefit is too small to reduce energy; even migration costs that contain migration overhead, performance degradation by core location and so on are higher than energy reduction in some cases. The overhead of migration depends on the migration count and the migration distance as discussed in Section 6.1 are also important facts that affect the migration cost. The core re-allocation algorithm, therefore, obtains high migration benefit with minimum cost.

BM	AH+M			AH+M+D			T+M			T+M+D		
	Power	Perf	PPW	Power	Perf	PPW	Power	Perf	PPW	Power	Perf	PPW
Mem Synthetic1	74.29	91.80	123.56	73.47	92.95	126.52	76.0	92.63	121.87	74.88	93.78	125.25
Mem Synthetic2	61.46	96.89	157.78	59.41	96.99	163.23	61.67	97.99	158.88	60.65	98.27	162.03

Table 7.5: Normalized result with or without data-location aware technique

To determine Δm as migration benefit threshold, we have experimented with the same 3 scenarios from Section 7.2.1 that applied several Δm as no limit, 5%, 10%, 15% and 20%. Table 7.3 and table 7.4 show the relation between migration benefit and migration count. If set higher Δm, it decreases migration count but uses more energy. We decide 5% as Δm that could get the highest performance per watt ratio within 1-2% performance loss.

7.2.3 Data-location Aware Migration

Table 6.1 shows memory throughput depend on the frequencies and the distances from the memory controller. It does not mean that core re-allocation always affects significant performance changes even though cores are allocated differently on every migration phase. Because lower memory performance sufficiently satisfy cpu-bound and weak memory-bound applications.

However, memory-bound applications that need huge memory access might raise or drop their performance according to the distance from the memory controller. It is possible to get a little improvement on memory performance by migrating to a location closer to the memory controllers within same frequency cores. We have experimented with two kinds of memory synthetic benchmark are listed in Appendix A.2. Table 7.5 shows results of a data-location aware technique combined with core re-allocation that gets 3-5% higher performance per watt improvement with 0.1-1.2% higher performance than a technique without data-location. This approach consistently enhance core re-allocation for memory-bound applications.

7.3 Results

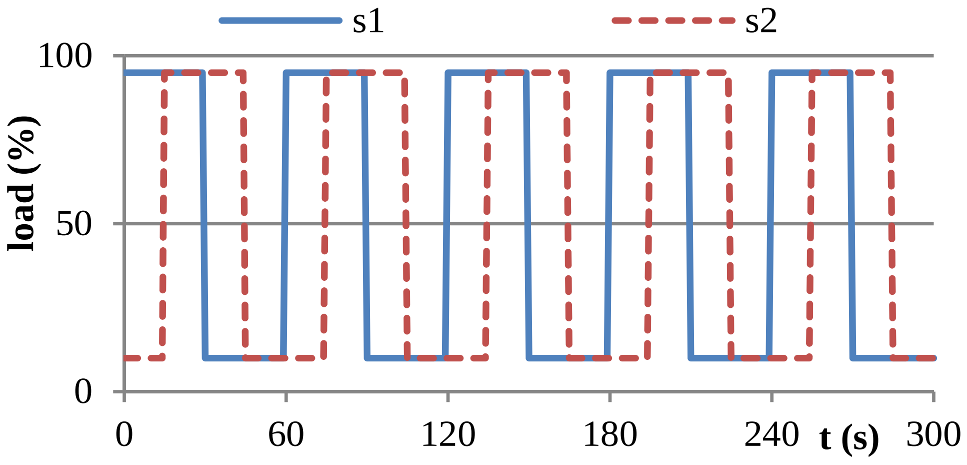
We have conducted a wide range of experiments with the proposed cooperative hierarchical power management technique for CMPs. To show the effect of core re-allocation with DVFS, we first present the results of a synthetic periodic workload before presenting the results for profiled benchmarks and the workload of the webservers of the France 1998 Soccer World Cup. We conclude this section with the overall results over all benchmark scenarios.

7.3.1 Synthetic Periodic Workload

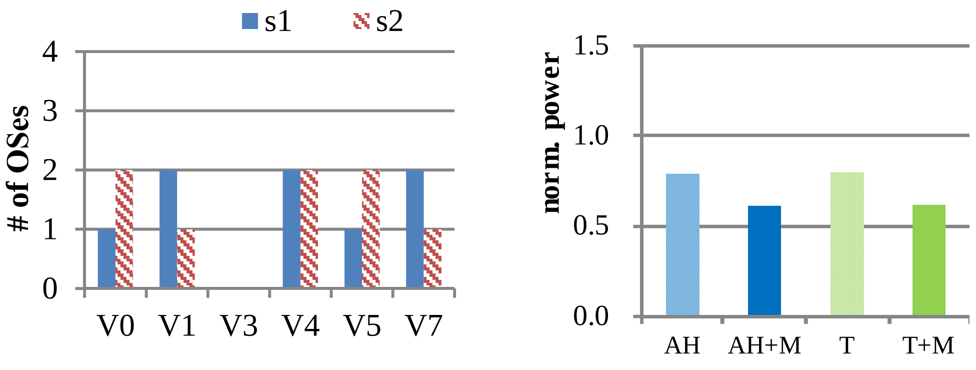
Figure 7.1 shows the setup and the result of synthetic periodic workload patterns. Figure 7.1 (a) shows the load patterns that alternate between 10% and 90%. These two load patterns have the same pattern but 15 seconds time-shifted. We distributed these two patterns to every voltage domain.

Figures 7.1 (b) and (c) show the results of this synthetic benchmark. The right-hand of Figure 7.1 (b) and Figure 7.1 (c) show the normalized power consumption, performance and performance per watt ratio for the `AH` and the `T` that means the `Allhigh` and the `Tile` DVFS policy. We have experimented each DVFS policies without and with (postfix `+M`) core re-allocation.

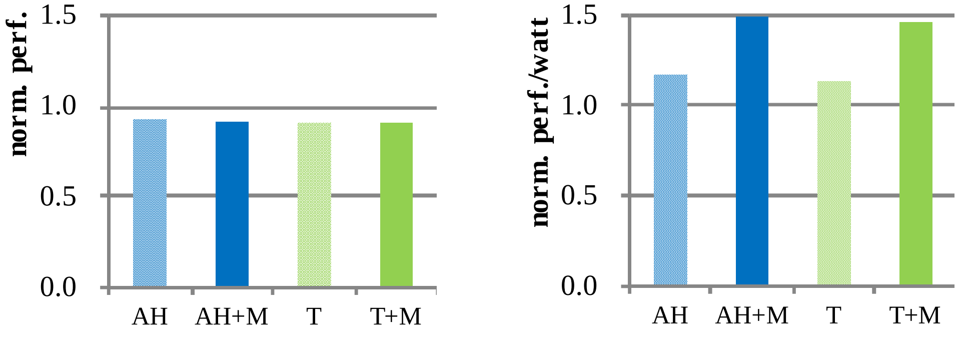
We observe that both DVFS only and DVFS+migration stay within the allowed performance loss of $\Delta p \leq 10\%$. Even though core re-allocation incurs additional overhead, this overhead is correctly reflected by the migration cost-benefit model (Chapter 6). The trade-off between performance and energy is reflected in the normalized power consumption shown in Figure 7.1 (b) on the right-hand side, the reduction in power by far outweighs the loss in performance. In terms of performance per watt ratio (right-hand of Figure 7.1 (c)), both DVFS only and DVFS+migration outperform the base case. In particular, the proposed method of combining DVFS with migration achieves about a 30% improved performance per watt ratio compared to DVFS-only policies.



(a) Load pattern

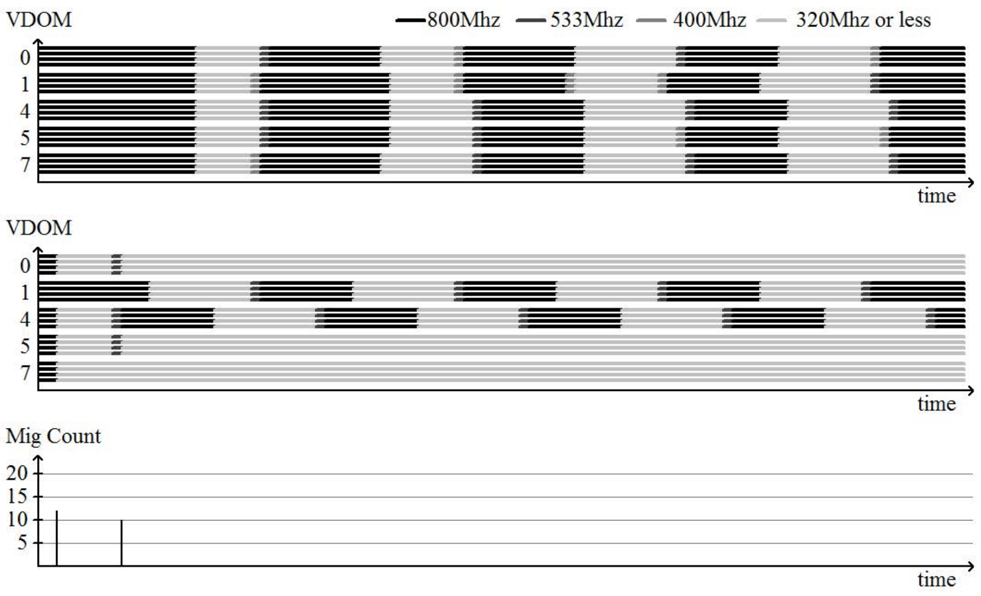


(b) Load distribution & power consumption

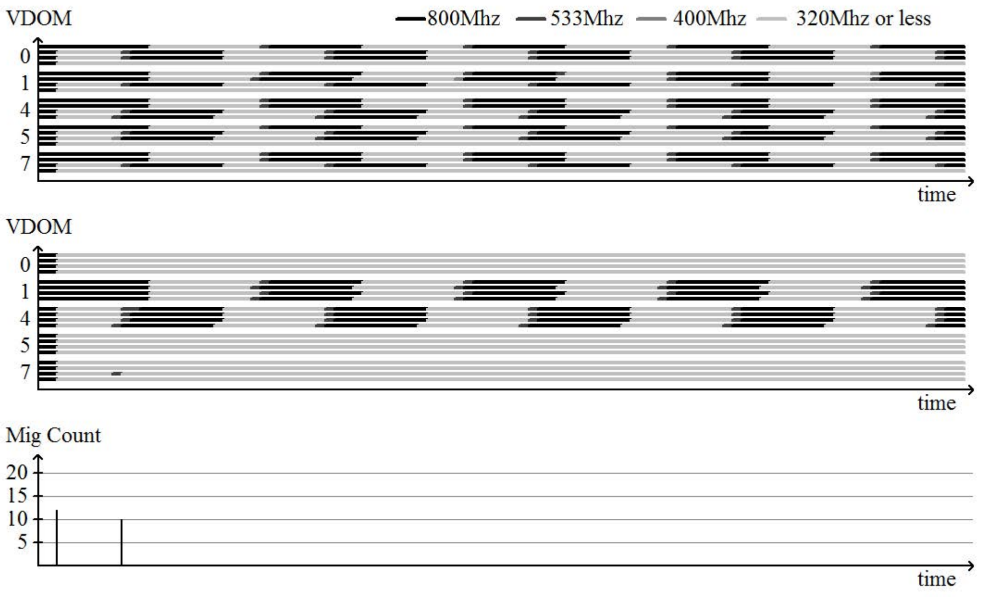


(c) Performance & performance per watt

Figure 7.1: Simple alternating synthetic load



(a) Requested frequency map - `Allhigh`



(b) Requested frequency map - `Tile`

Figure 7.2: Frequency map for simple alternating synthetic load

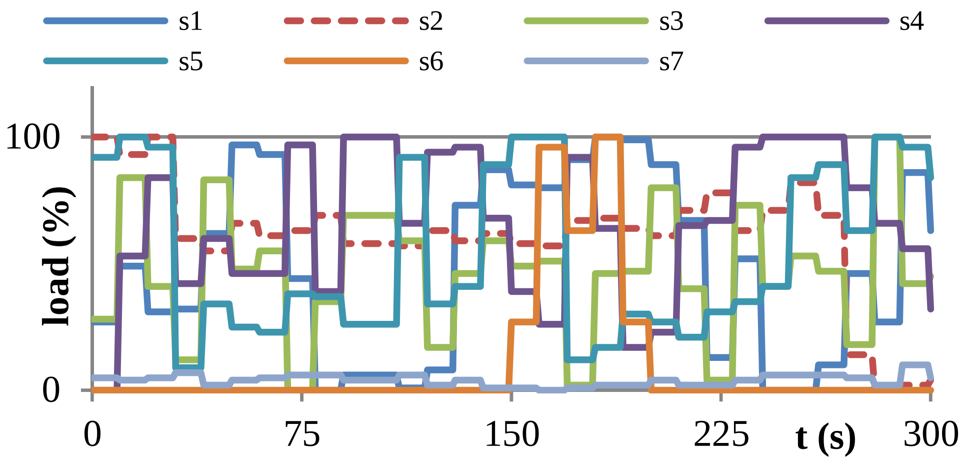
Figure 7.2, finally, visualizes the effect of DVFS only and DVFS+migration on the individual voltage domains' frequency settings for the two DVFS policies `Allhigh` and `Tile`. The frequency over time is shown for each voltage domain for DVFS only (upper part) and DVFS+migration (middle part of the figure). Higher frequency (and thus voltage) settings are represented by darker levels of gray. The lower part of the chart shows the number of migrations over time. We observe how the cooperative core re-allocation technique manages to group cores with similar performance requirements together, thereby allowing more optimal DVFS settings.

7.3.2 Profiled Workload

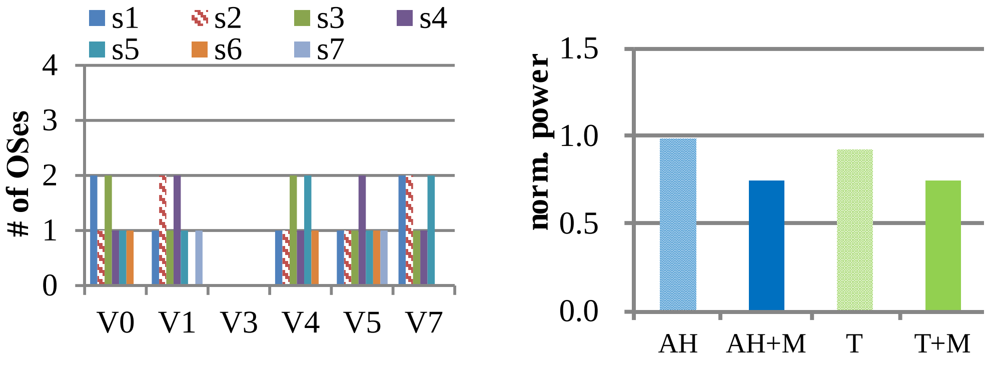
Figure 7.3 shows the results of a scenario based on actual, measured workload patterns. Seven different load patterns obtained from profiling data of university staff and graduate students' computers, `s1` to `s7`, have been selected and are assigned to a total of 40 application containers and initially placed onto the different voltage domains as shown in left-hand of Figure 7.3 (b).

Compared to the synthetic workload, the performance loss (left-hand in Figure 7.3 (c)) is less severe (0.2% and 1.6% for `AH` and `AH+M`, and 2.7% and 3.7% for `T` and `T+M`, respectively). This is because profiled workloads exhibit smoother workload changes; the performance prediction is thus more accurate. The DVFS-only policies cannot group cores with similar workload characteristics together, and all voltage domains run at maximal voltage during most of the benchmark (upper-hand VDOM charts in Figure 7.4). As a consequence, only minimal total energy savings are obtained (1.1% and 7.5% for `AH` and `T`).

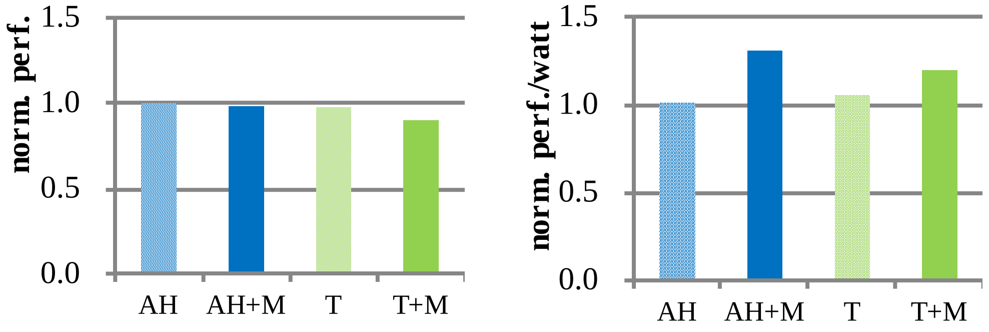
With OS migration, the scheduler is able to group workloads exhibiting similar load patterns into voltage domains as shown by the lower-hand VDOM charts in Figure 7.4. The total energy savings are significant (24.2% and 25.5% for `AH+M` and `T+M`) and lead to a much better performance per watt ratio increase compared to DVFS only (0.9% and 5.1% for DVFS only, 29.8% and 29.3% for



(a) Load pattern

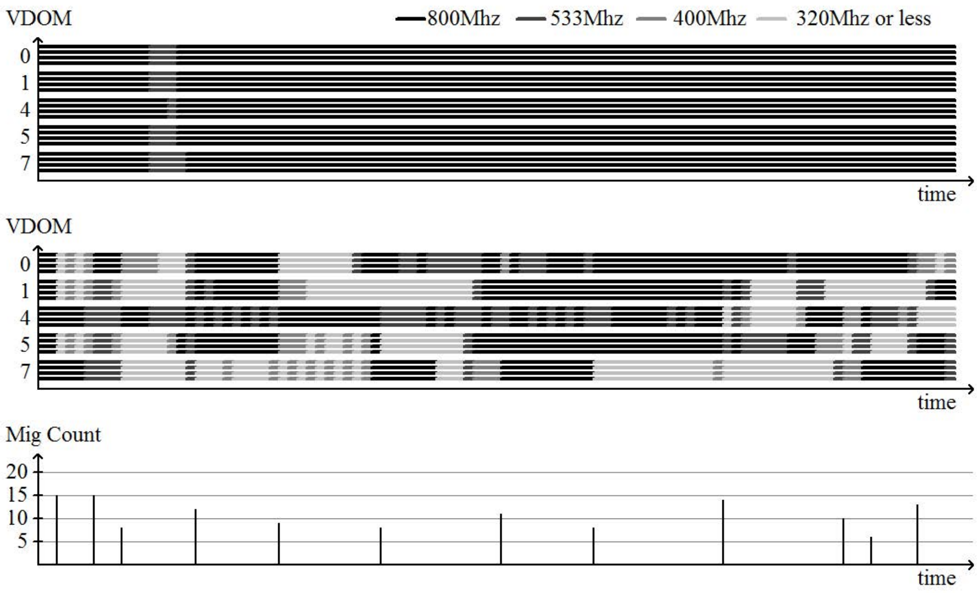


(b) Load distribution & power consumption

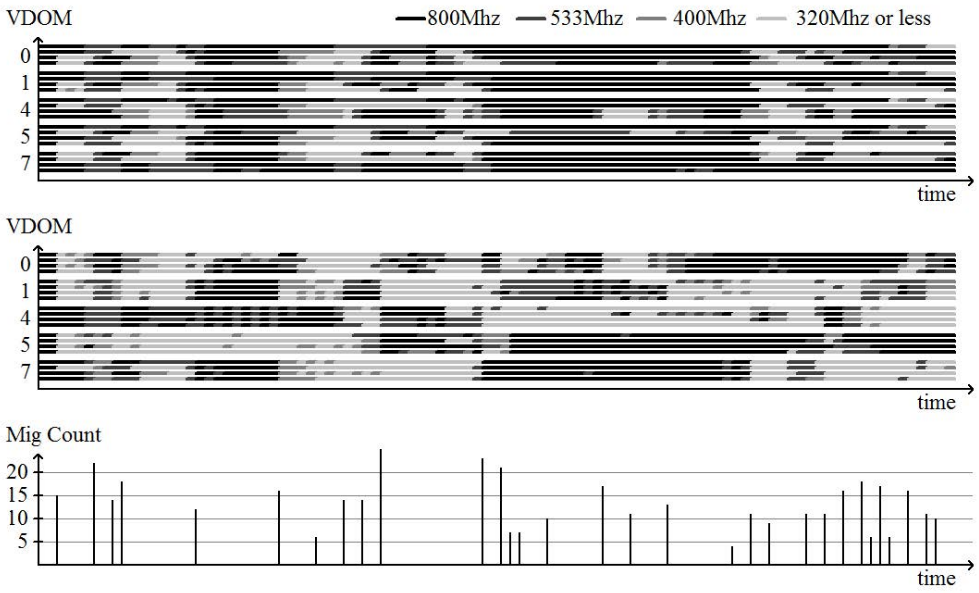


(c) Performance & performance per watt

Figure 7.3: Frequency map for profiled load patterns



(a) Requested frequency map - `Allhigh`



(b) Requested frequency map - `Tile`

Figure 7.4: Results for profiled load patterns

DVFS+migration).

7.3.3 World Cup Workload

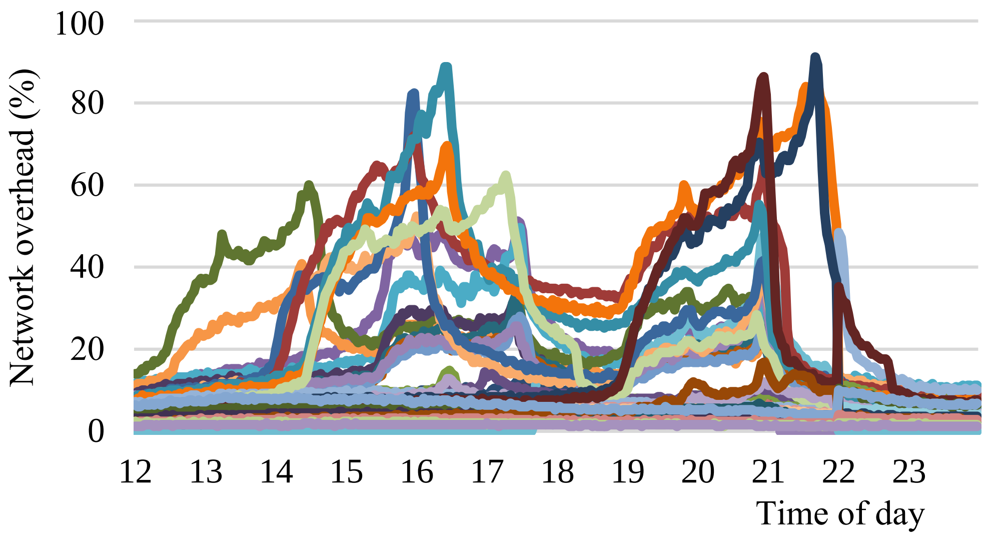
The World Cup workload obtained from [30] is fed into a request generator on the MCPC. On the SCC, we run 40 independent application containers, each of which runs an HTTP server. In the absence of load balancing, we feed data from 40 different days to each of the 40 HTTP servers.

Figure 7.5 shows the workload patterns and the normalized power, response time, and performance per watt ratio. The World Cup scenarios have a long-term idle time in the morning and late night. Also, these network overheads peak before and after playing the game. DVFS policies, therefore, could get a huge reduction of power consumption and improve a lot of performance per watt ratio. An interesting observation is that core re-allocation *reduces the average response time* of the benchmark compared to a DVFS-only solution (Figure 7.5 (b) right-hand side). This is caused by the fact that core migration allows the DVFS algorithm to select frequencies closer to the optimum value than without migration.

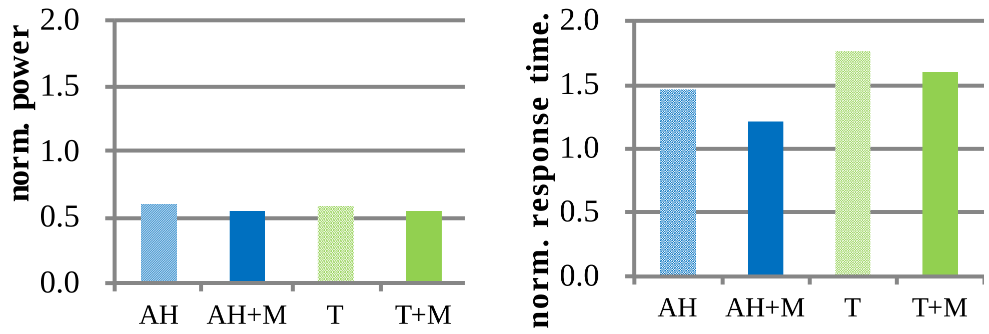
The right-hand of Figure 7.5 (c) shows the sensitivity of the core location and the accessed data for the `Tile` DVFS policy. `T` denotes the DVFS only policy, `T+M` is the data-location aware DVFS+migration algorithm, and `T+M w/o data location` shows the results if the location of the cores' data is ignored. Comparing `T+M` with `T+M w/o data location`, we note that ignoring the data location causes a 1% decrease in the performance/watt and a 6% faster response time.

7.3.4 Overall Results

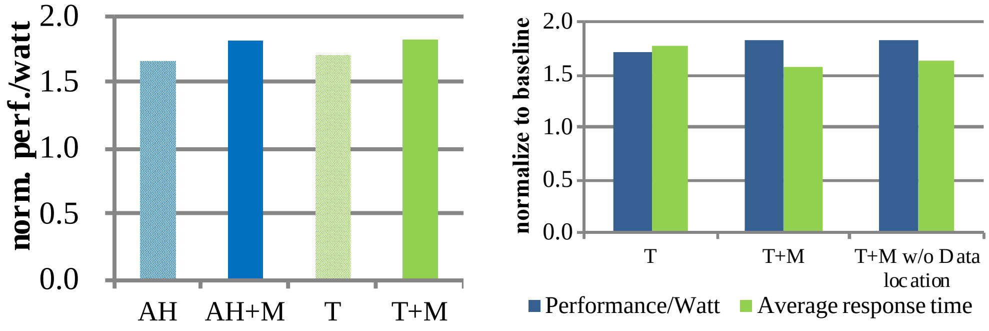
Table 7.6 displays the normalized power, performance, and performance per watt ratio over the baseline, respectively, for the `Allhigh` and the `Tile` policy, denoted `AH` and `T`, without and with (appended `+M` postfix) OS migration for



(a) Load pattern



(b) Power & performance



(c) Performance per watt & location sensitivity

Figure 7.5: Results for the France 1998 World Cup load pattern.

BM	AH			AH+M			T			T+M		
	Power	Perf	PPW	Power	Perf	PPW	Power	Perf	PPW	Power	Perf	PPW
Synthetic1	79.1	92.7	117.1	63.8	92.4	144.9	85.7	90.6	105.7	64.2	90.9	141.6
Synthetic2	91.9	99.5	108.3	65.0	98.9	152.3	84.3	97.7	115.9	61.2	93.8	153.1
Profiled1	98.9	99.8	100.9	75.8	98.4	129.8	92.5	97.3	105.1	74.5	96.3	129.3
Profiled2	90.3	99.0	109.7	71.4	98.8	138.3	88.8	96.7	108.9	68.2	90.5	132.7
Profiled3	93.7	98.5	105.1	66.9	98.6	147.4	86.2	96.7	112.1	67.4	95.1	141.2
WorldCup	60.2	99.9	166.1	54.9	99.8	181.9	58.6	98.7	170.6	54.7	98.9	182.9
Average	**85.7**	**98.2**	**117.9**	**66.3**	**97.8**	**149.1**	**82.7**	**96.3**	**119.7**	**65.0**	**94.3**	**146.8**

Table 7.6: Normalized power, performance, and performance per watt (PPW)

two synthetic workload patterns, three profiled workload patterns, and the simulated webserver for the soccer World Cup in France 1998. The details for two synthetic benchmark are listed in Appendix A.1 and three profiled benchmark are listed in Appendix A.3. The web server access logs for the soccer World Cup in France 1998 are available in [31].

Independent of the workload at hand, re-allocating cores before applying a DVFS policy results in a significantly reduced power consumption at the expense of a very moderate performance degradation. Taking the DVFS-only policy as the baseline, `Allhigh+Migration` achieves a 35% better power-per-watt energy efficiency than `Allhigh` at a relative performance loss of only 0.8%. Similarly, `Tile+Migration` outperforms `Tile` by 25% at a performance loss of 2.9%. We observe that `Tile` outperforms `Allhigh` without migration whereas with migration they achieve similar performance. The reason is that OS migration is able to group OSes with similar performance requirements into voltage domains such that the superior `Tile` DVFS policy has less effect.

Chapter 8

Conclusion

We have presented a cooperative hierarchical power management technique for existing and future many-core systems running a space-shared operating system. We show that the combination of core re-allocation and DVFS techniques archive significant energy savings through space sharing manner.

Without explicit hardware support, the application runtimes only executing on the individual cores cooperate with the global power management by saving and restoring the volatile state of the core on demand. Combined with dynamic monitoring of each core's performance metrics and, this technique allows the power manager to group cores with similar performance requirement so that traditional DVFS policies can apply DVF settings closer to the optimal value on the CMP. Furthermore, our technique considers the data location for memory-bound applications, performance penalty and migration benefit threshold to get higher performance per watt ratio with a minimum performance degradation.

The cooperative power manager has been implemented and evaluated on a real system, the Intel Single-chip Cloud Computer. Experiments show that, on average, the proposed technique outperforms existing DVFS policies by 27-32% at the expense of a small performance loss of 1-2%.

Appendix A

Profiled Workload Benchmark Scenarios

This appendix describes the details of the benchmarks evaluated in this work. Each benchmark scenario consists of two parts:

- Two or more workload patterns that describe how the workload changes.
- An initial assignment of the workloads to the 48 cores of the SCC.

Each workload pattern (WL), denoted S{1-10} in the tables below, lists the CPU workload as `C` and MEM workload as `M` for every epoch (10 or 15 seconds, depending on the benchmark) for the duration of one period (300 seconds). A workload never stops, it keeps repeating the workload pattern period after period. Only memory synthetic benchmark contains CPU and MEM workload together. Another workloads are pure CPU-based workloads.

The core assignment tables show what workload patterns are assigned to which cores when the experiment starts. In our setup, voltage domain 3 runs various logging and measurement services and is thus not available for benchmarks. The power measurements include the power consumed by `vdom3` because power is only reported for the entire chip and not for individual voltage domains.

A benchmark ends after the predefined number of seconds (in our example, after 300 seconds). The total progress of each workload is measured externally and thus includes all overheads caused by migration, voltage changes or slowdowns cause by too low frequency settings.

A.1 Synthetic Benchmark Scenario based on Periodic Workloads

A.1.1 Synthetic Benchmark Scenario 1

Workload patterns:

WL	Epoch (1 epoch = 15 sec)																				
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
S1	95	95	10	10	95	95	10	10	95	95	10	10	95	95	10	10	95	95	10	10	10
S2	10	95	95	10	10	95	95	10	10	95	95	10	10	95	95	10	10	95	95	10	10

Core assignment:

vdom0		vdom1		vdom3		vdom4		vdom5		vdom7	
-	-	-	-	n/a	n/a	-	-	-	-	-	-
S2	-	S2	-	n/a	n/a	S2	S2	S2	-	S2	-
-	-	-	-	n/a	n/a	-	-	-	-	-	
S1	S2	S1	S1	n/a	n/a	S1	S1	S1	S2	S1	S1

A.1.2 Synthetic Benchmark Scenario 2

Workload patterns:

WL	Epoch (1 epoch = 10 sec)																														
	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
S1	42	77	25	11	34	36	30	14	33	26	22	58	100	52	30	13	15	0	21	39	48	43	40	41	40	42	41	40	39	36	35
S2	45	15	6	27	25	9	64	55	27	28	18	51	46	100	56	20	25	25	12	0	0	0	0	0	0	0	0	0	0	0	0
S3	71	53	26	9	34	25	23	38	37	26	30	23	34	41	39	29	29	12	17	30	27	21	31	35	41	84	89	63	100	96	2
S4	11	22	20	10	27	12	45	100	22	9	4	14	9	43	19	6	17	18	14	21	5	5	5	6	25	16	7	0	0	0	0
S5	42	66	40	67	57	67	66	71	75	72	31	38	59	54	86	80	68	55	95	100	89	85	86	77	64	0	0	0	0	0	0

Core assignment:

vdom0		vdom1		vdom2		vdom4		vdom5		vdom7	
S5	-	-	-	n/a	n/a	S5	-	S5	-	S5	-
-	-	S5	-	n/a	n/a	S4	-	S4	-	S4	-
S2	S4	S2	S4	n/a	n/a	-	S3	S2	S3	S2	-
S1	S3	S1	S3	n/a	n/a	S1	S2	S1	-	S1	S3

A.2 Memory Synthetic Benchmark Scenario based on Periodic Workloads

A.2.1 Memory Synthetic Benchmark Scenario 1

Workload patterns:

WL		Epoch (1 epoch = 15 sec)																				
		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
S1	C	95	95	10	10	95	95	10	10	95	95	10	10	95	95	10	10	95	95	10	10	10
S2	C	10	95	95	10	10	95	95	10	10	95	95	10	10	95	95	10	10	95	95	10	10
S3	M	100	100	5	5	100	100	5	5	100	100	5	5	100	100	5	5	100	100	5	5	5
S4	M	5	100	100	5	5	100	100	5	5	100	100	5	5	100	100	5	5	100	100	5	5

Core assignment:

vdom0		vdom1		vdom3		vdom4		vdom5		vdom7	
-	S4	-	-	n/a	n/a	S4	S3	-	S4	S3	-
S2	S3	S2	S3	n/a	n/a	S2	S2	S2	S3	S2	-
S4	S4	-	S3	n/a	n/a	S3	S4	S4	-	S4	S3
S1	S2	S1	S1	n/a	n/a	S1	S1	S1	S2	S1	S1

A.2.2 Memory Synthetic Benchmark Scenario 2

Workload patterns:

WL		Epoch (1 epoch = 10 sec)																													
		00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
S1	c	42	77	25	11	34	36	30	14	33	26	22	58	100	52	30	13	15	0	21	39	48	43	40	41	40	42	41	40	39	36	35
S2	c	45	15	6	27	25	9	64	55	27	28	18	51	46	100	56	20	25	25	12	0	0	0	0	0	0	0	0	0	0	0	0
S3	c	71	53	26	9	34	25	23	38	37	26	30	23	34	41	39	29	29	12	17	30	27	21	31	35	41	84	89	63	100	96	2
S4	c	11	22	20	10	27	12	45	100	22	9	4	14	9	43	19	6	17	18	14	21	5	5	5	6	25	16	7	0	0	0	0
S5	c	42	66	40	67	57	67	66	71	75	72	31	38	59	54	86	80	68	55	95	100	89	85	86	77	64	0	0	0	0	0	0
S6	m	42	77	25	11	34	36	30	14	33	26	22	58	100	52	30	13	15	0	21	39	48	43	40	41	40	42	41	40	39	36	35
S7	m	45	15	6	27	25	9	64	55	27	28	18	51	46	100	56	20	25	25	12	0	0	0	0	0	0	0	0	0	0	0	0
S8	m	71	53	26	9	34	25	23	38	37	26	30	23	34	41	39	29	29	12	17	30	27	21	31	35	41	84	89	63	100	96	2
S9	m	11	22	20	10	27	12	45	100	22	9	4	14	9	43	19	6	17	18	14	21	5	5	5	6	25	16	7	0	0	0	0
S10	m	42	66	40	67	57	67	66	71	75	72	31	38	59	54	86	80	68	55	95	100	89	85	86	77	64	0	0	0	0	0	0

Core assignment:

vdom0		vdom1		vdom2		vdom4		vdom5		vdom7	
S10	-	-	-	n/a	n/a	S5	-	S10	-	S5	-
-	-	S5	-	n/a	n/a	S9	-	S4	-	S9	-
S2	S4	S7	S9	n/a	n/a	-	S3	S2	S8	S7	-
S6	S8	S1	S3	n/a	n/a	S1	S7	S6	-	S1	S3

A.3 Benchmark Scenario based on Profiled Workloads

A.3.1 Profiled Benchmark Scenario 1

Workload patterns:

WL	Epoch (1 epoch = 10 sec)																														
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
S1	27	49	31	32	62	77	80	44	0	6	1	1	8	73	87	81	80	91	100	99	89	67	13	52	0	0	10	46	27	86	63
S2	69	57	68	60	55	66	61	63	69	58	56	57	63	59	62	58	57	67	68	64	61	71	78	63	71	82	69	14	0	2	4
S3	28	84	41	12	83	48	55	0	35	69	42	59	17	46	59	49	51	2	46	47	80	40	4	73	41	53	47	18	100	42	45
S4	27	49	31	32	62	77	80	44	0	6	1	1	8	73	87	81	80	91	100	99	89	67	13	52	0	0	10	80	66	56	32
S5	71	53	26	9	34	25	23	38	37	26	96	92	34	41	89	100	100	12	17	30	27	21	31	35	41	84	89	63	100	96	84
S6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	27	96	63	100	27	0	0	0	0	0	0	0	0	0	0	0
S7	5	4	5	7	2	4	5	6	6	4	100	6	2	4	1	1	0	1	2	2	4	2	2	4	6	6	6	5	2	10	5

Core assignment:

vdom0		vdom1		vdom3		vdom4		vdom5		vdom7	
S4	S6	S4	S4	n/a	n/a	S5	S5	S5	S6	S5	S5
S3	S3	S3	S7	n/a	n/a	S3	S3	S4	S4	S2	S2
S2	S5	S2	S2	n/a	n/a	S2	S6	S2	S7	S3	S4
S1	S1	S1	S5	n/a	n/a	S1	S4	S1	S3	S1	S1

A.3.2 Profiled Benchmark Scenario 2

Workload patterns:

WL	Epoch (1 epoch = 10 sec)																														
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
S1	27	49	31	32	62	77	80	44	0	6	1	1	8	73	87	81	80	91	100	99	89	67	13	52	0	0	10	46	27	86	63
S2	82	39	55	42	96	42	100	33	53	20	20	10	11	14	13	11	13	13	1	5	1	0	23	45	61	42	83	83	20	15	3
S3	8	20	21	30	80	100	24	50	36	54	83	92	91	73	27	1	0	1	1	1	1	0	1	1	10	1	21	17	33	5	7
S4	27	49	31	32	62	77	80	44	0	6	1	1	8	73	87	81	80	91	100	99	89	67	13	52	0	0	10	10	15	30	27
S5	27	49	31	32	62	77	80	44	0	6	1	1	8	73	87	81	80	91	100	99	89	67	2	2	1	0	5	3	7	13	0
S6	53	21	52	48	33	92	89	100	39	38	29	41	48	4	64	45	36	31	42	41	42	35	15	80	93	62	10	23	48	32	0

Core assignment:

vdom0		vdom1		vdom2		vdom4		vdom5		vdom7	
S6	-	S6	-	n/a	n/a	S6	-	S6	-	S6	-
S3	S4	S3	S4	n/a	n/a	S3	S4	S3	S4	S3	S4
-	S5	-	S5	n/a	n/a	-	S5	-	S5	-	S5
S1	S2	S1	S2	n/a	n/a	S1	S2	S1	S2	S1	S2

A.3.3 Profiled Benchmark Scenario 3

Workload patterns:

WL	Epoch (1 epoch = 10 sec)																														
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
S1	27	49	31	32	62	77	80	44	0	6	1	1	8	73	87	81	80	91	100	99	89	67	13	52	0	0	10	46	27	86	63
S2	82	39	55	42	96	42	100	33	53	20	20	10	11	14	13	11	13	13	1	5	1	0	23	45	61	42	83	83	20	15	3
S3	28	84	41	12	83	48	55	0	35	69	42	59	17	46	59	49	51	2	46	47	80	40	4	73	41	53	47	18	100	42	45
S4	27	49	31	32	62	77	80	44	0	6	1	1	8	73	87	81	80	91	100	99	89	67	13	52	0	0	10	10	15	30	27
S5	71	53	26	9	34	25	23	38	37	26	96	92	34	41	89	100	100	12	17	30	27	21	31	35	41	84	89	63	100	96	84
S6	53	21	52	48	33	92	89	100	39	38	29	41	48	4	64	45	36	31	42	41	42	35	15	80	93	62	10	23	48	32	0

Core assignment:

vdom0		vdom1		vdom3		vdom4		vdom5		vdom7	
-	-	-	-	n/a	n/a	-	-	-	-	-	-
S5	S6	S5	S6	n/a	n/a	S3	S6	S4	S5	S4	S5
-	-	-	-	n/a	n/a	-	-	-	-	-	-
S1	S4	S1	S2	n/a	n/a	S1	S2	S2	S3	S1	S3

Bibliography

[1] Luiz André Barroso et al. Piranha: A scalable architecture based on single-chip multiprocessing. In *Proceedings of the 27th Annual International Symposium on Computer Architecture*, ISCA '00, pages 282–293, New York, NY, USA, 2000. ACM.

[2] Alejandro Duran and Michael Klemm. The intel many integrated core architecture. In *High Performance Computing and Simulation (HPCS), 2012 International Conference on*, pages 365–366, July 2012.

[3] NVIDIA. GeForce GTX TITAN X. `http://www.geforce.com/hardware/desktop-gpus/geforce-gtx-titan-x`. Online, accessed June 2015.

[4] EZchip. TILE-MX Multicore Processor. `http://www.tilera.com/products/?ezchip=585&spage=686`. Online, accessed June 2015.

[5] Saurabh Dighe et al. Within-die variation-aware dynamic-voltage-frequency-scaling with optimal core allocation and thread hopping for the 80-core teraflops processor. *Solid-State Circuits, IEEE Journal of*, 46(1):184–193, Jan 2011.

[6] Wonyoung Kim, Meeta Sharma Gupta, Gu-Yeon Wei, and David Brooks. System level analysis of fast, per-core dvfs using on-chip switching regulators. In *High Performance Computer Architecture, 2008. HPCA 2008. IEEE 14th International Symposium on*, pages 123–134. IEEE, 2008.

[7] Sebastian Herbert and Diana Marculescu. Analysis of dynamic voltage/frequency scaling in chip-multiprocessors. In *Low Power Electronics and Design (ISLPED), 2007 ACM/IEEE International Symposium on*, pages 38–43. IEEE, 2007.

[8] Efraim Rotem, Avi Mendelson, Ran Ginosar, and Uri Weiser. Multiple clock and voltage domains for chip multi processors. In *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, pages 459–468. ACM, 2009.

[9] Stephen Soltesz, Herbert Pötzl, Marc E. Fiuczynski, Andy Bavier, and Larry Peterson. Container-based operating system virtualization: A scalable, high-performance alternative to hypervisors. In *Proceedings of the 2Nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007*, EuroSys '07, pages 275–287, New York, NY, USA, 2007. ACM.

[10] Silas Boyd-Wickizer et al. Corey: An operating system for many cores. In *Proceedings of the 8th USENIX Conference on Operating Systems Design and Implementation*, OSDI'08, pages 43–57, Berkeley, CA, USA, 2008. USENIX Association.

[11] Andrew Baumann, Paul Barham, Pierre-Evariste Dagand, Tim Harris, Rebecca Isaacs, Simon Peter, Timothy Roscoe, Adrian Schüpbach, and Akhilesh Singhania. The multikernel: A new os architecture for scalable multicore systems. In *Proceedings of the ACM SIGOPS 22Nd Symposium on Operating Systems Principles*, SOSP '09, pages 29–44, New York, NY, USA, 2009. ACM.

[12] Rose Liu, Kevin Klues, Sarah Bird, Steven Hofmeyr, Krste Asanović, and John Kubiatowicz. Tessellation: Space-time partitioning in a manycore client os. In *Proceedings of the First USENIX Conference on Hot Topics*

in Parallelism, HotPar'09, pages 10–10, Berkeley, CA, USA, 2009. USENIX Association.

[13] David Wentzlaff and Anant Agarwal. Factored operating systems (fos): The case for a scalable operating system for multicores. *SIGOPS Oper. Syst. Rev.*, 43(2):76–85, April 2009.

[14] Jason Howard, Saurabh Dighe, Yatin Hoskote, Sriram Vangal, David Finan, Gregory Ruhl, David Jenkins, Howard Wilson, Nitin Borkar, Gerhard Schrom, et al. A 48-core ia-32 message-passing processor with dvfs in 45nm cmos. In *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2010 IEEE International*, pages 108–109. IEEE, 2010.

[15] Chanseok Kang. Hierachical power management framework on manycore system using os migration techniques. Master's thesis, Seoul National University, 2015.

[16] Rakesh Kumar, Dean M. Tullsen, Parthasarathy Ranganathan, Norman P. Jouppi, and Keith I. Farkas. Single-isa heterogeneous multi-core architectures for multithreaded workload performance. In *Proceedings of the 31st Annual International Symposium on Computer Architecture*, ISCA '04, pages 64–, Washington, DC, USA, 2004. IEEE Computer Society.

[17] Soraya Ghiasi. *Aide De Camp: Asymmetric Multi-core Design for Dynamic Thermal Management*. PhD thesis, Boulder, CO, USA, 2004. AAI3136618.

[18] David Meisner, Brian T Gold, and Thomas F Wenisch. Powernap: eliminating server idle power. *ACM SIGARCH Computer Architecture News*, 37(1):205–216, 2009.

[19] David Meisner and Thomas F Wenisch. Dreamweaver: architectural support for deep sleep. *ACM SIGPLAN Notices*, 47(4):313–324, 2012.

[20] Nikolas Ioannou, Michael Kauschke, Matthias Gries, and Marcelo Cintra. Phase-based application-driven hierarchical power management on the single-chip cloud computer. In *Parallel Architectures and Compilation Techniques (PACT), 2011 International Conference on*, pages 131–142. IEEE, 2011.

[21] Cai Qiong, José González, Grigorios Magklis, Pedro Chaparro, and Antonio González. Thread shuffling: Combining dvfs and thread migration to reduce energy consumptions for multi-core systems. In *Low Power Electronics and Design (ISLPED) 2011 International Symposium on*, pages 379–384. IEEE, 2011.

[22] Canturk Isci, Alper Buyuktosunoglu, Chen-Yong Cher, Pradip Bose, and Margaret Martonosi. An analysis of efficient multi-core global power management policies: Maximizing performance for a given power budget. In *Proceedings of the 39th annual IEEE/ACM international symposium on microarchitecture*, pages 347–358. IEEE Computer Society, 2006.

[23] Kai Ma, Xue Li, Ming Chen, and Xiaorui Wang. Scalable power control for many-core architectures running multi-threaded applications. In *Proceedings of the 38th Annual International Symposium on Computer Architecture*, ISCA '11, pages 449–460, New York, NY, USA, 2011. ACM.

[24] Ke Meng, Russ Joseph, Robert P Dick, and Li Shang. Multi-optimization power management for chip multiprocessors. In *Proceedings of the 17th international conference on Parallel architectures and compilation techniques*, pages 177–186. ACM, 2008.

[25] Krishna K Rangan, Gu-Yeon Wei, and David Brooks. Thread motion: fine-grained power management for multi-core systems. In *ACM SIGARCH Computer Architecture News*, volume 37, pages 302–313. ACM, 2009.

[26] András Vajda. Multi-core and many-core processor architectures. In *Programming Many-Core Chips*, pages 9–43. Springer, 2011.

[27] Tilera. TILE-Gx Instruction Set Architecture. `http://www.tilera.com/scm/docs/UG401-ISA.pdf`. Online, accessed June 2015.

[28] Tilera. Multicore Development Environment Optimization Guide. `http://www.tilera.com/scm/docs/UG105-Optimization-Guide.pdf`. Online, accessed June 2015.

[29] Tilera Corporation. UG130. Tile processor architecture overview for the tile-gx series, May 2012.

[30] Martin Arlitt and Tai Jin. A workload characterization study of the 1998 world cup web site. *Network, IEEE*, 14(3):30–37, May 2000.

[31] FIFA. 1998 World Cup Web Site Access Logs. `http://ita.ee.lbl.gov/html/contrib/WorldCup.html`. Online, accessed June 2015.

요약

최근 Cloud Computing 서비스를 제공하는 데이터센터 등에서는 Many-core chip이 기존 Multi-core를 대체하여 사용되고 있으며 Operating System도 Many-core 시스템을 사용할 수 있게 Space-sharing 방식으로 설계가 변경되고 있다. 이러한 추세속에서 기존의 전통적인 DVFS 방식을 이용해서는 Many-core 환경에서 효율적인 전력 사용이 어렵기 때문에 추가적인 전력 관리 방법과 Many-core의 특성을 고려한 Core 재배치 기술이 필요하다.

Space-shared OS는 Core와 물리적인 메모리의 구성에 대한 자원 관리를 하는데, 최근의 Chip multiprocessor (CMP) 들은 각각의 Core에서 독립적으로 DVFS를 동작하도록 하지 않고 몇개의 Core들을 그룹화하여 Voltage 또는 Frequency를 함께 변경할 수 있도록 지원하고 있으며 메모리 또한 Coarse-grained 방식으로 독립된 파티션으로 할당 할 수 있게 관리된다. 본 연구는 이러한 CMP의 특성을 고려하여 Core 재배치와 DVFS 기술을 이용한 계층적 전력 관리 시스템을 연구하는데 목표가 있다. 특히 Core 재배치 기술은 Core의 위치에 따른 Data 성능도 함께 고려하고 있다. 이에 추가로 DVFS 성능 손실을 고려한 에너지 효율성 상승과 Core 재배치시 발생할 수 있는 효과를 미리 계산하여 최소한의 성능저하로 더 좋은 에너지 효율성을 얻을 수 있도록 연구를 진행하였다. 또한 실제 구현 및 실험은 Intel에서 출시한 Single-chip Cloud Computer (SCC)에서 진행하였으며 시나리오별로 1-2%의 성능 손실로 Performance per watt ratio가 27-32% 향상되었다. 또한 Migration 효과와 Data 지역성 등을 고려하지 않았던 기존 연구보다 성능이 5-11% 좋아졌다.

주요어: 매니코어 아키텍쳐, 스케쥴링, 동적 전압 및 주파수 변경, 에너지 효율
학번: 2013-23128