M.S. THESIS

Hierarchical Power Management Framework on Manycore Systems Using OS Migration Techniques

매니코어 시스템 상에서 OS Migration 기술을 활용한 계층적 전력 관리 구조

FEBRUARY 2015

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

Chanseok Kang

M.S. THESIS

Hierarchical Power Management Framework on Manycore Systems Using OS Migration Techniques

매니코어 시스템 상에서 OS Migration 기술을 활용한 계층적 전력 관리 구조

FEBRUARY 2015

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

Chanseok Kang

Hierarchical Power Management Framework on Manycore Systems Using OS Migration Techniques

매니코어 시스템 상에서 OS Migration 기술을 활용한 계층적 전력 관리 구조

지도교수 Bernhard Egger

이 논문을 공학석사학위논문으로 제출함

2014 년 10 월

서울대학교 대학원

전기 컴퓨터 공학부

강 찬 석

강 찬 석의 석사학위논문을 인준함

2014 년 12 월

위 원 장	유 석 인	(인)
부위원장	Bernhard Egger	(인)
위 원	Robert Ian McKay	(인)

Abstract

Hierarchical Power Management Framework on Manycore Systems Using OS Migration Techniques

Chanseok Kang
Department of Electrical Engineering and Computer Science
Collage of Engineering
The Graduate School
Seoul National University

Recently, power management schemes are required in manycore systems for gratifying the chip-level power and thermal constraints. Most research focuses on optimizing power consumption for parallel applications running on all cores. In this thesis, we tried to investigate the possibilities of feasible power management of a manycore systems running several independent operating systems (OS). Exploiting hardware support such as memory address indirection and global shared memory, a zero-copy OS migration technique is implemented on the Linux platform with minimal overhead. In the context of dynamic voltage and frequency scaling for many-core chips where due to the cost and few practical reasons, the voltage and frequency can only be controlled for a group of cores physically grouped into a voltage and/or frequency domain. In this case, Zero-copy OS migration can be employed to group OSes with similar performance characteristics into one voltage domain which then allows DVFS algorithm to better match the voltage and frequency settings to the characteristics of that

domain. From hierarchical power management framework implemented on the Intel Single-chip Cloud Computer (SCC) running up to 40 independent Linux instances, We show that our approach, on average, saves the energy consumption by 30% over existing DVFS policies for a wide range of load patterns with minimal performance degradation. In the conclusion, it can improve the performance per watt ratio by 27% in most of benchmark situations.

Keywords: Many-core Architecture, OS Migration, Power Management, SCC
Student Number: 2013-20738

Contents

List of Figures

List of Tables

Chapter 1

Introduction

According to the improvement in transistor integration on chip during the last decade, the recent trend of CPU architecture has changed from single-core to multi-core. But it led to power consumption and thermal constraints becoming one of the primary design consideration. Originally, Moore‘s law [1], coupled with Dennard scaling [2], tends to increase the performance of a core by increasing the number of transistors on the chip. But in fact, the power wall effect causes more switching activities on it as we increase the number of transistors, and core itself requires more power density [3]. As a result, the higher power consumption leads to an increase not only energy costs but also chip temperature, and it affects chip reliability and lifetime.

For managing power consumption, most processors provide hardware support for the dynamic voltage and frequency scaling (DVFS). Examples are Intel SpeedStep [4] and AMD PowerNow [5]. The operating system (OS) periodically monitors the workload of each core, and changes the voltage and frequency to improve the efficiency of power management. Viewed from an abstract design level, the dynamic power consumption of a semiconductor device is proportional to the frequency of the processor cores and the voltage supplied to the cores.

It means that when the user wants high performance, DVFS policies manages to increase the core frequency. In the other case, by lowering the voltage and frequency, it can reduce the dynamic power consumption.

But the drawback of DVFS is that it can be reduced both frequency and voltage changes incur some overhead. In some situations, it may degrade the performance of task execution. For this reason, algorithms for maintaining energy efficiency are required. Albers [6] reviewed some DVFS algorithms to minimize energy consumption and maintain the task performance. Actually, task is aware of time variable. To minimize performance degradation which may result in missed deadlines, the core itself should notice the execution time of task. In fact, it is an online problem, so core does not know about the next workload. More briefly, when the core operates in idle mode, core does not have any time information about the task. This increases the probability of deadline misses, and causes the performance degradation. Machine Learning techniques are proposed to solve these kind of problems [7, 8, 9].

In views of chip multiprocessors (CMP), [10] shows the compared result between the per-core DVFS and chip-wide DVFS. And some researches mentioned that domain-specific power management can achieve good performance [11, 12]. Cores are physically allocated in voltage/frequency domains. All cores located in a specific domain, have the same power properties. So it can reduce the hardware overhead, and eventually increase the performance.

Prior research on power management for CMPs has mainly focused on optimizing power consumption/performance under a certain limit such as a power budget [13]. To deal with the constraints imposed by voltage and/or frequency domains, different heuristics to compute appropriate settings for each voltage and frequency domain without moving threads around have been proposed [14]. Due to the cost for managing voltage/frequency through hardware, this kind of approach get effects of power management. But the user cannot obtain the full potential of DVFS through this approach in some cases. For example, when

core executing different processes required different voltage/frequency setting are co-located in the same domain, the individual cores may have either been observed a power-overconsumption or performance degradation. To overcome this problem, cores with similar performance requirements should be merged in to one domain. Some research has dealt with this approach in views of threads [15, 16]. Implemented in the context of one operating system with shared memory, moving a task to a different core is a scheduling decision and does not involve copying of a task's memory.

For power management on manycore systems, scalability is one of the most important considerations. Usually, when the number of cores are increased, the complexity of handling features (DVFS, task scheduling, data analysis) also increases linearly. Especially in a centralized framework where one dedicated core manages the entire process, the computational complexity leads to a lack of scalability for manycore systems. To overcome this problem, agent-based approach has been proposed [17].

This paper proposes a new zero-copy OS live migration method for independent OSes running on CMPs exploiting hardware features such as global shared memory, address translation and domain-specific power management. In the past, OS Migration was an expensive operation due to the high cost of copying the volatile state of the OS, i.e., contents of the memory. In order to provide separable address spaces to OSes running in parallel, CMPs often provides an additional level of indirection in the memory address translation from core-physical to system memory addresses. This feature enables us to implement zero-copy migration of completely independent OSes by modifying the physical-to-system address translation tables. Contrary to the previous migration techniques, it has same effect of physical memory copying with low overhead. Currently, no commercial CMP provides the means to access the register file of a halted core. To overcome this limitation, an interrupt handler processes migration requests by copying and restoring the internal volatile state of a core

to a designated space in the global shared memory. We have implemented the zero-copy migration techniques in the Linux operating system running on the Intel Single-chip Cloud Computer (SCC) [18], a research prototype CMP comprising 48 IA-32 Pentium III processors. With hierarchical power management policy, it measured the Energy-Delay Product [19], and compared with state-of-the-art techniques [14]. As a result, the proposed approach gets significantly better figures by 27% over wide range of synthetic benchmarks.

1.1 Outline

The rest of this thesis is organized as follows: Chapter 2 gives the information about the architecture of the Intel Single-chip Cloud Computer(SCC) and necessary background. Chapter 3 describes the implementation of the proposed zero-copy OS migration and power management framework in detail. Chapter 4 presents the experimental setup and results. Related work is discussed in Chapter 5; and Chapter 6 concludes the thesis.

Chapter 2

Intel Single-chip Cloud Computer

The proposed technique can be implemented on any DVFS-capable CMP architecture providing global shared memory or private memory with core-to-system address translation. In this section, it provide an overview of the many-core architecture used in our experiments and discuss some of the relevant capabilities in more detail.

2.1 Architecture Overview

The Single-chip Cloud Computer (SCC) [18] is a 48-core prototype many-core architecture created by Intel Labs. It consists of 48 independent cores interconnected by a routed Network-on-Chip (NoC). The cores are Intel P54C Pentium®cores with L1 caches (16KB) and L2 caches (256KB). And it supports for managing the on-chip scratchpad memory termed message passing buffer (MPB), which is typically used to implement message passing. No cache coherence is provided for the core-local L1 and L2 caches. Two neighbor cores are bundled in frequency domain so-called `tile`; the 24 tiles are organized on a 6 by 4 grid. Four memory controllers in the four corners of the chip provide

access to up to 32GBs of memory. A system FPGA provides the interface between the CMP and the management console PC (MCPC). Figure 2.1 shows the SCC block diagram.

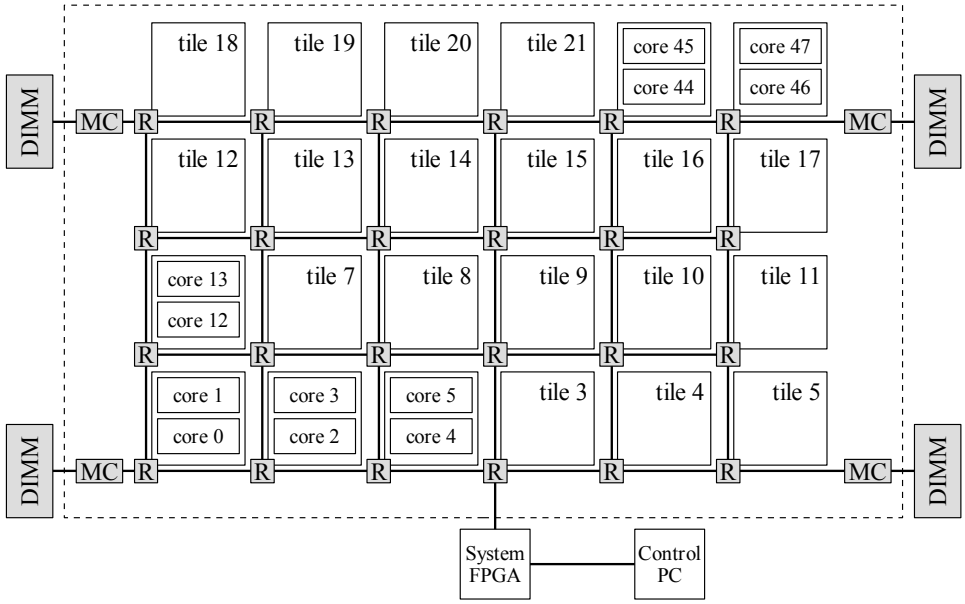


Figure 2.1: Intel Single-chip Cloud Computer(SCC) block diagram

2.2 Dynamic Voltage/Frequency Scaling

The SCC provides voltage and frequency control over the cores and the NoC. The frequency can be controlled per-tile. It means that the neighbor cores located on the same tile always run at the same frequency and constitute a clock domain. The voltage can be regulated for a group of four tiles. For example, a voltage domain comprises a total of eight cores. Figure 2.2 illustrates the clock and voltage domains on the SCC. The figure does not show the voltage domain 2 and 6, because they are the same and represents the entire mesh and system interface.

Voltage and frequency are controlled and queried through registers. Each tile has specific register to write/read the tile‘s frequency. Voltage control is

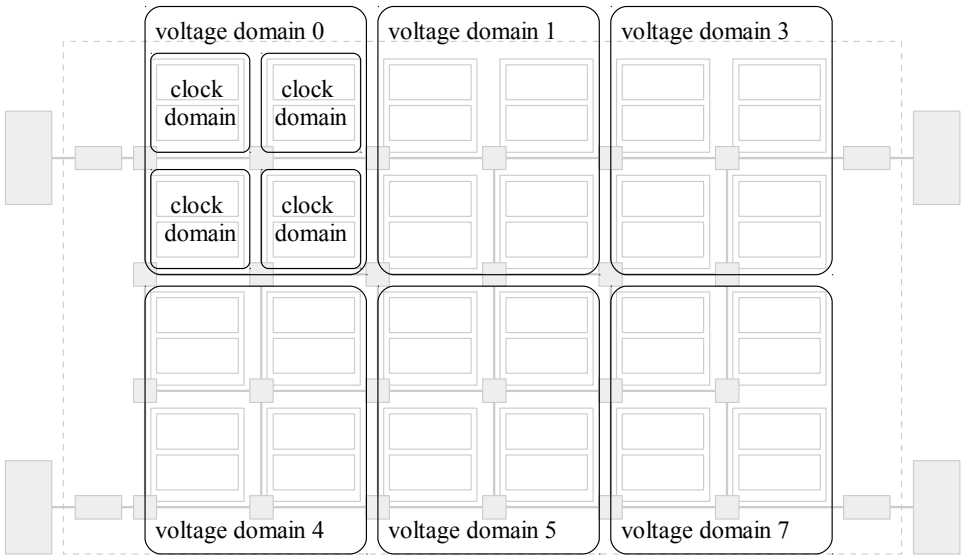


Figure 2.2: Voltage and clock domains on the Intel SCC

also controlled through the register interface. Frequency changes happen almost immediately (10ms) , however, measurements on SCC revealed that voltage changes may take up to 100ms to complete (this value is obtained by comparing the difference in progress of two cores, one control core running on a unchanged voltage domain, and one operating on the domain whose voltage is changed). In addition, voltage change requests can only affect in single domain; requests for several domains must be serialized.

In specification document, SCC supports seven different supply voltage level, but only three level are of practical interest: 1.1V to run at a frequency of 800MHz, 0.8V to run at a frequency of 533MHz, and 0.7V for frequencies between 400MHz and 100MHz. But when it is measured the actual voltage in practice, the required voltage for running core in 400MHz is over 0.7V. So it decides to increase the reference voltage for 533MHz and remaining frequencies by 0.1V. The frequency is set by writing a divisor between 2 and 16 for the 1.6GHz(1600MHz) clock resulting in core frequencies from 800MHz to 100MHz.

Table 2.1 lists the available voltage and frequency settings for the cores. The notation of designator is selected from the first digits of Frequency. We'll use this notation to explain the migration policy in Section 3.2.6.

Voltage [V]	**Frequency [MHz]**	**Designator**
1.1	800	8
0.9	533	5
0.8	400	4
0.7	320	3
	200	2
	100	1

Table 2.1: Voltage and frequency settings on the SCC

2.3 Power Measurement

The SCC provides a number of heat sensors by each core plus voltage and ampere meters on-board for measuring supply power on SCC board. The total power consumed by the SCC is obtained by multiplying the (almost constant) supply voltage with the supply current for the entire board. The power consumption of individual voltage domain cannot be computed because only the per-domain supply voltage is available but not the current consumed by the domain. So this thesis always report the total power consumption of entire board in our experiments in Chapter 4, not the individual core.

The sensors and meters can be read via telnet protocol from the system FPGA in management console or by directly querying the system FPGA from a core in the SCC. We chose the former approach because it can obtain the power consumption without core overhead.

2.4 Memory Addressing

Each core provides the standard virtual-physical memory translation; all addresses leaving the core are 32 bit physical addresses. 32 bit addresses are not wide enough to express the entire 32-GB address range; to allow access to a total of 4 GB of memory located somewhere in the SCC‘s 32GB address space, an additional address translation takes place.

The address translation from core (physical) to system address is provided by a core-local lookup table (LUT). Each LUT has 256 entries and is indexed by the top eight bits of the 32 bit core address. Without going into much detail, a LUT entry contains an 8 bit destination ID *destID* designating one of the four memory controllers (MC), and 10 address bits that are pre-pended to the remaining 24 bits of the core address to form a 34 bit address. One LUT entry maps 16 MB of memory. Together with the memory controller designation, this translation allows to access the entire 32 GB memory space of the SCC. Figure 2.3 illustrates the core-to-system address translation.

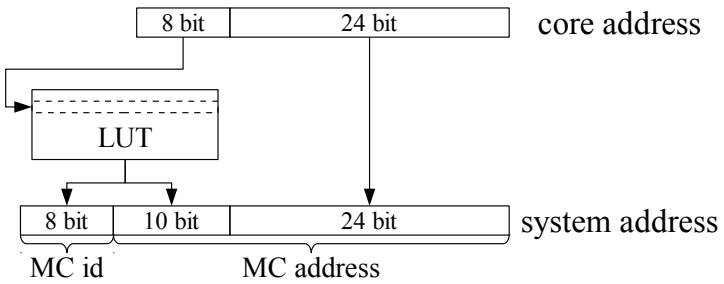


Figure 2.3: Core-to-system address translation on the SCC

Chapter 3

Implementation

In the previous sections, it have motivated the need to cluster workloads with similar performance requirements in a voltage and/or frequency domain to achieve optimal results when applying DVFS. A workload in this context can mean anything from a thread of a parallel application to an entire operating system running exclusively on a core. Clustering threads or processes in the presence of an operating system with shared memory amounts to re-scheduling them on a different core. For applications exhibiting periodic behavior and fork-join parallel programs special techniques allow accurate estimation of the expected performance requirements and thus more aggressive DVFS policies.

This sections describes the technical issues of OS migration; the migration policy is discussed in the section on power management.

3.1 Zero-copy OS Migration

Moving an operating system from one physical core to another can be implemented with or without cooperation of the migrated OS. In a co-operative setting, the OS can enter a safe state in which it is moved to the newly assigned

core and then resumed. The OS itself takes care of changed memory mappings and the like. To keep the modifications to the OS to a minimum and potentially support a wide range of OSes, it exploit features of CMPs to implement uncooperative zero-copy OS migration.

For independent workloads running on different physical cores the main caveat is how to deal with the volatile state, i.e., the assigned memory of the workload and values currently held in registers in the CPU core. If the CMP implements a global shared address space, the assigned memory does not have to be moved physically; the same physical addresses are still valid on the new core. Since such designs cannot provide total isolation of independently running workloads, CMPs often implement an additional step in the memory translation process from physical to system addresses. The physical-to-system address translation operates in almost the same way virtual-to-physical translation works: instead of per-process page tables, the CMP provides per-core translation tables indexed by the higher part of the core address . We exploit this additional translation step to realize zero-copy migration for independently running OSes.

Without cooperation from the OS, the volatile state of the OS also includes the data values kept in the registers of the CPU core. Similar to preemptive task switch, these register values need to be saved on the source core and restored on the destination core. If the CMP provides the means to read and write the register file of physical cores, OS migration can be implemented without any cooperation from the migrated OS. To this day, however, no many-core chip aware of provides such a feature. As a consequence, a minimal amount of cooperation from the migrated OS is necessary.

3.1.1 Migration Steps

In the proposed implementation, zero-copy OS migration is orchestrated by a migration manager. The migration manager initially signals the OS running

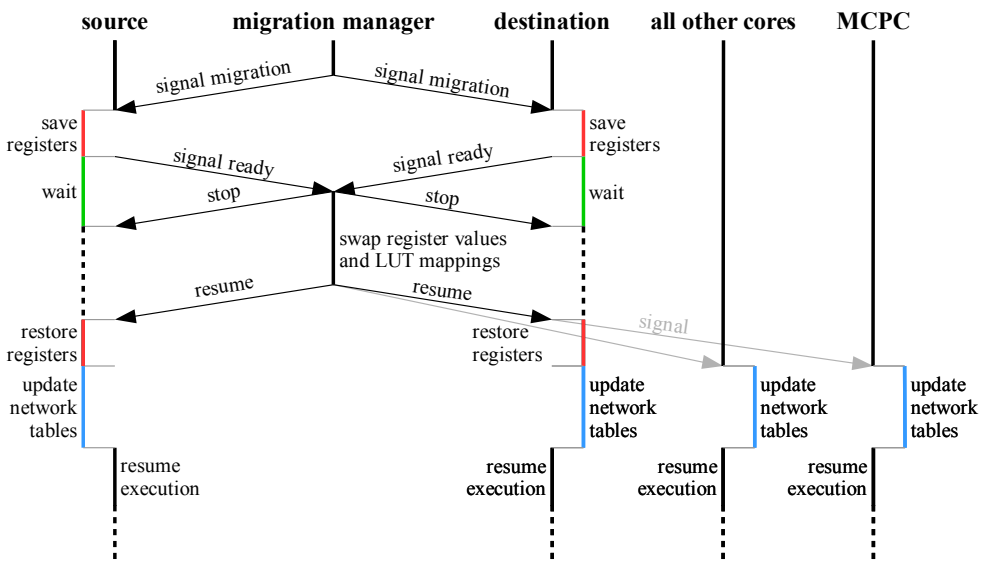


Figure 3.1: OS Migration Steps

on the source core that it is about to be migrated. The OS then saves the values of CPU-internal registers to a designated memory area and enters a special state during which the actual migration takes place. On the destination core, the system is brought into an identical migration state as the source core. As soon as both cores signal completion readiness, the core-to-system memory mappings are swapped. The migration manager then signals the completion of the migration to the destination core which then restores the register values and resumes execution. Figure 3.1 illustrates the migration steps.

Since it is required a minimal amount of cooperation from all involved cores, it assumes that the bare OS runs on all (including unused) cores. The clock of unused cores can be gated to avoid wasting power. The migration signal is transmitted in form of an interrupt to the affected cores. The migration interrupt is handled by our custom interrupt handler which saves the necessary registers into a per-core designated memory area. After all registers have been saved, the affected cores signal completion to the migration manager and completely flush

their caches. The migration manager then stops all affected cores by gating their clock, and swaps the cores' register values and the memory mappings. Next, the migration manager signals completion of the migration by resuming the clock on the migrated cores. The cores proceed by restoring the (new) register values from memory, exit the interrupt handler and resume operation. In addition, all cores, including the MCPC need to update internal network routing tables to reflect the new locations of the cores.

3.1.2 Migrating Volatile State

The migrated cores save/restore register values to a designated memory area in a custom interrupt handler. In principle, exactly the same registers need to be saved/restored as when performing a process context switch in a preemptive multitasking system. After saving the registers, the migrated cores flush all caches and enter a busy loop. It is impossible to flush a core's cache externally; as a consequence the code of the busy loop will reside in a migrated core's instruction cache. In addition, it is impossible to set the program counter immediately after resuming the clock since we do not know what instruction of the busy loop the core was executing when the clock was gated. However, it can ignore this technical difficulty by assuring that the busy loop, including the code to save and restore the registers, is located at identical virtual addresses on all migrated cores. Since we currently only use a modified version of the sccLinux OS, this condition is always met. If several different OSes are involved in OS migration, it may be necessary to turn off virtual-to-physical memory translation temporarily and only re-enable it once the new page table base register has been set.

The memory of the affected cores is migrated by swapping the corresponding entries in the LUT tables. All 256 entries are swapped.

3.1.3 Networking

Apart from the migration interrupt handler, no modifications have been made to the sccLinux kernel. As a consequence, the cores keep their networking configuration, including IP addresses across migrations.

On the SCC, there exist two separate networks: one network for on-chip networking, and a subnet for communication with the MCPC. Data packets sent on-chip from one core to another are first stored in a buffer (in a special on-chip SRAM buffer called *message-passing buffer* (MPB)) on the sender side. The sender then signals the receiver with an interrupt, and the receiver fetches the message data directly from the sender's buffer. For migrated cores the location of their MPB remains unchanged; storing/retrieving network packets from the buffer is thus unaffected by migration. However, the target core of a network interrupt is identified by its physical core ID which corresponds to the x/y-coordinates of the core on the grid. In the original sccLinux the interrupt target ID is computed from the core ID. In order to support migration, a table containing the IP-to-coreID mappings is added and kept up-to-date by each core. After each migration, the migration manager thus notifies *all* cores about the changes necessary to the IP-to-coreID mapping table.

A very similar data structure is maintained by the MCPC to route data packets from external sources to each of the individual cores. The migration manager notifies the MCPC through the system interface about migrations taking place such that the MCPC can keep an up-to-date list of IP-to-coreIDs.

These two simple modifications are enough to keep networking, including open connections, alive across migrations. No other devices exist on the SCC; input/output, including access to permanent storage, are routed through the network.

3.2 Hierarchical Power Management

Power management for many-core CMPs is often organized in a hierarchical manner in order to remain scalable even for a large number of cores. This section describes our implementation.

3.2.1 Organization

The structure of the hierarchical power manager reflects the structure of the SCC with its different voltage and frequency domains. At the lowest level in the hierarchy is a single core because individual cores exhibit different performance values and can be clock-gated individually. The next level is the tile which comprises two cores and represents a clock domain. Decisions about which clock frequency to run at are made at this level. One level up is the voltage domain. A clock domain consists of four tiles and represents the unit where voltage changes can be initiated. The highest level models the entire chip.

For simplicity of the implementation, currently all three levels except the core-level are grouped together and executed on one single core in the SCC. This is feasible for the 24 clock domains, the 6 voltage domains, and the global manger on top; for CMP with more cores the voltage and/or clock domain managers will need to be distributed as well.

3.2.2 Local Performance Monitoring and Prediction

On each active core, a local agent monitors the current performance of the core. Depending on the load factor, it requests a higher, the same, or a lower frequency from the next-higher level in the hierarchy. In its current implementation, the local agent only considers the CPU load. We expect better results with more sophisticated implementations such as using the core's performance monitoring unit to measure the number of ALU and memory instructions. In this thesis, we just considered the CPU workload as performance measure cases.

The local agent uses the core‘s performance monitoring unit (PMU) to gether statistics about the number of executed instructions. During the regular intervals, the local agent collects the instruction counts, and predicts the load of a core based on weighted average of collected data. When the core is not fully utilized, the optimal frequency can be analytically computed. In the case of CPU-bound test case, for example, if the core operates in 800MHz with 50% utilization, we expect that it also works optimally in 400MHz with 100% utilization. Frequency values we use in this system are discrete, the computed frequency is thus always rounded up to the next higher available frequency.

However, if the core is fully optimized, it is not clear by how much the next frequency is appropriated. In the previous example, if the core works in 100% utilization at 400MHz, we are not sure that the optimal frequency is either 533MHz or 800MHz. So we have experimented with three policies: `STEP, 2-UP, HALF-UP`. The first two policies increase the operating frequency by one and two steps in that case. And `HALF-UP` policy computes the next frequency by adding half the difference of the current frequency to the maximal frequency (800MHz). Experiments have shown that in our framework the performance and power consumption are almost indifferent in regard to the three policies.

Performance is measured periodically; experiments have shown that values between 0.5 and one second are short enough to quickly react to changing performance requirements, but long enough to avoid too much noise in the signal.

3.2.3 Domain Managers

Each domain, clock, voltage, and global, maintains its own domain manager. Each level only communicates directly with the level above or below, i.e., the clock domain manager interacts with the voltage domain manager, the voltage domain manager interacts downstream with the clock domain, and upstream with the global domain manager. The functionality of the different domain

managers is elaborated in more detail in the following expression.

- **Clock Domain Manager:** For each clock domain, its manager computes and sets the appropriate frequency. The frequency of a clock domain is constrained by the current voltage level of the corresponding voltage domain and computed based on the performance counters reported by the local agents and the currently active DVFS policy. Each clock domain manager maintains sorted lists of the current and requested frequencies for all of its cores. The clock domain managers communicate with their voltage domain manager by periodically sending the list of requested frequencies. The voltage domain manager signals changes in the voltage level.

- **Voltage Domain Manager:** The voltage domain manager computes and sets the operating voltage of a voltage domain. Due to the nature of DVFS, voltage changes must happen in close collaboration with frequency changes: before lowering the voltage, all frequencies must be lowered to a values supported by the lower voltage. Similarly, for higher voltages, the voltage must be increased before the frequencies can be raised as well. Each voltage domain manager maintains sorted lists of the current and requested voltages per clock domain. The voltage domain managers communicate with the global manager by periodically sending the list of requested frequencies and voltages upstream.

- **Global Domain Manager:** The global manager gathers the sorted voltage/frequency requests from the domain managers and determines which cores to migrated where based on the current policy. After the migration has completed, the global domain managers informs the voltage domain managers of the migration such that the voltage may be changed immediately. This is not absolutely necessary since the information will eventually be sent from the local agents to the voltage domain managers, however,

giving the voltage domain managers a chance to immediately react to migration leads to slightly better results.

3.2.4 Power Management Policies

We employ DVFS and OS migration to achieve a better *performance per watt* ratio. Other goals such as, for example, avoiding local hot spots are also possible but outside the scope of this thesis.

3.2.5 DVFS Policies

We implement the same DVFS policies as a state-of-the-art hierarchical power manager for CMPs [14]. This power manager has been implemented for the Intel SCC chip and thus provides a good reference point. The policies proposed in [14] and duplicated here are:

- `Allhigh`: this policy runs all cores within a voltage domain at the highest requested frequency.
- `TileIndiv` grants the requested frequency to each clock domain and sets the voltage accordingly. Within each clock domain, the higher of the requested frequencies is chosen.

For all policies, the voltages of the voltage domains are computed such that the all clock frequencies of the associated clock domains can be satisfied, i.e., $v_{VD} = \max_i v(f_{CDi})$. $v(f)$ for a given frequency f is a simple table lookup (Table 2.1).

3.2.6 OS Migration Policy

Without OS migration, OSes are pinned to their cores. For voltage domains containing very busy cores and mostly idle cores any voltage setting thus has its shortcomings: if the voltage is too low, the performance of busy cores is

severely affected. 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.

Thanks to OS migration, it is possible to migrate cores with similar performance requirements onto the same voltage domain. Since all cores in the voltage domain require similar performance the requested frequency range is narrower which in turn allows for a more optimal voltage setting.

In other words, the goal of OS migration is to group workloads with similar performance requirements in the same voltage and/or frequency domain such that the DVFS policy can set a voltage/frequency closer at the optimal value for a given voltage/frequency domain.

A naïve algorithm is to sort the OSes 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 one time quantum, this algorithm fails to consider the overhead of OS migration. The actual live migration of a OS is very quick ($\leq 3ms$), each time a OS is migrated it will experience a lot of cold misses in the local instruction and data caches which will lead to both a performance reduction as well as increased memory traffic. The migration algorithm must thus also consider the current positions of the OSes and minimize the number of migrations.

Algorithm 1 shows the pseudo-code of the migration process worked in global domain manager. For achieving the advantage of hierarchical power management, the decision part is separated with two parts: voltage decision and frequency decision. We implemented the simple migration policy based on economic principle, named `Buyer-Seller` algorithm. The goal of this policy is to minimize the number of migrations with consideration of voltage and frequency domain at the same time. Mentioned in Section 2, two cores consists in one frequency domain. So it is impossible to set the optimal frequency for each core. To minimize the performance degradation or over power consumption, similar

Algorithm 1 BuyerSellerMigration

```
1: procedure MIGRATE
2:     while target > 1 do
3:         for each voltage domain do
4:             buyer ← getBuyerDomain()
5:             while hasTileBuyer(buyer) do
6:                 if seller ← getTileSellerDomain() then
7:                     break
8:                 end if
9:                 target ←doTileMigration(buyer, seller)
10:            end while
11:            while hasCoreBuyer(buyer) do
12:                if seller ← getCoreSellerDomain(buyer) then
13:                    break
14:                end if
15:                target ←doCoreMigration(buyer, seller)
16:            end while
17:        end for
18:    end while
19: end procedure
```

workloads should operates in same domain. In this algorithm, it tries to migrate the core/domain unit as a view of `Buyer` or `Seller`. The concept of `Buyer` is that specific domain wants to bring some cores from another domain, and vice versa.

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		~~$\{5,4\}, \{5,4\}, \{3,3\}, \{4,4\}$~~
5	(5)	$\{3,1\}, \{5,3\}, \{2,2\}, (5)$
7	(4)	$\{1,1\}, \{4,5\}, \{1,1\}, (4)$

Table 3.1: Example of `Keep`/`Sell` lists for the configuration.

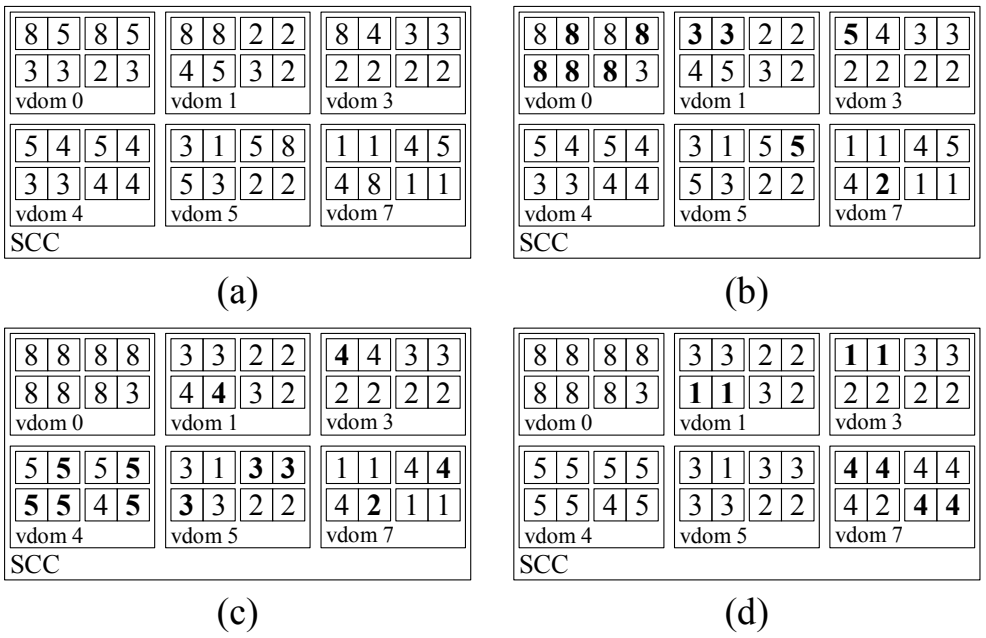


Figure 3.2: Example of Buyer-seller algorithm

Figure 3.2 (a) displays the estimated frequencies for each core before the buyer-seller algorithm starts. Figures 3.2 (b)-(d) show the layout after each

repetition for $v_i = 8, 5$ and 4, respectively; (d) represents the final configuration. In this figure, bold values represent tiles/cores migrated in that iteration.

This algorithm runs once for each voltage level except the lowest, starting at the highest voltage. In each iteration, domains report the information about which tiles or cores they would like to buy and which to sell. In the iteration for voltage v_i, a voltage domain adds to its `Sell` list (a) all tiles which request a frequency that can be run at a lower voltage than v_i, i.e., $v_{req}(tile) \leq v_i$, and (b) all single cores that requires a voltage smaller than v_i but are located on a tile where the other core requests a higher or equal voltage than v_i, $v_{req}(core) \leq v_i$. On the `Keep` list tiles and cores that request voltage v_i, that is $v_{req}(tile/core) = v_i$, are included. Table 3.1 shows the `Keep` and `Sell` lists for the configuration shown in Figure 3.2 (a). Note that in the `Keep` list not the actual single core itself is listed but its sibling from the same tile. Consider `vdom0`: the two tiles at the top are expected to require the frequencies 8 and 5 (800 and 533MHz, respectively, see Table 2.1, described in Section 2). The `Keep` list of `vdom0` then contains two single core entries (5), (5), expressing that it contains two tiles with one core each running at the target frequency that it wants to keep. It would like to buy two single cores running at v_i 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.

After initializing the `Keep/Sell` lists, the buyer-seller algorithm runs. The algorithm repeatedly selects two tiles or single cores to swap based on the information in the `Keep/Sell` lists until no further chances occur. In each repetition, it first selects the voltage domain that offers the fewest tiles for sale and as its counterpart the domain that tries to keep the fewest tiles. In the first round

for $v_i = 1.1V$ (i.e. frequency 8) the domains `vdom0` and `vdom1` are chosen (Table 3.1). `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 operated in v_i. 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_i 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 is thus selected and exchanged with tile $\{8, 8\}$ from `vdom1`. This operation also updates the domains' `Keep/Sell` lists. The 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 tiles 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 voltage v_i and the entry in the `Keep` list represents the frequency of the sibling on the same tile. Again, this process is repeated until no further cores can be exchanged. Tiles on the `Keep` list can be split up into two single cores if there are single cores to be sold but the `Keep` lists only contain tiles. 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.

As a result, 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.

3.2.7 Phase Ordering and Frequency Considerations

In order to achieve maximum power saving, migration should occur before applying DVFS. 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 migration process. Voltage changes are quite expensive operation because the clock of all affected cores(i.e., whole cores in one domain) is stopped during the rather long voltage adjustment. Frequency changes, on the other hand, are almost instantaneous and can thus be performed often. On the SCC specifically, it have 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. Nevertheless, for our server/desktop benchmark scenarios with rather slow changes in the CPU load, migration and voltage changes can be performed at every step. Chapter 4 discusses the benchmarks and results in more detail.

Chapter 4

Experimentation and Evaluation

4.1 Experimental Setup

All experiments are executed on an Intel Single-chip Cloud Computer. The cores all run independent instance of sccLinux executing a variety of workloads reflecting some corner cases and desktop usage patterns. The baseline is obtained by running the benchmark at the SCC‘s default setting with all cores running at 533MHz and no DVFS or OS migration taking place before experimentation.

The proposed technique is compared with a state-of-the-art domain-specific power management techniques implemented on the Intel SCC[14]. Unlike the work in [14], phase-detector based on message passing buffer is not implemented on the proposed framework since it is aimed at independently running OSes on a CMP. We compare each of their DVFS policies without OS migration, `Allhigh`, and `TileIndiv`, against the same policy with zero-copy OS migration.

For comparing the performance among previous policies, this thesis choose the Energy-Delay Product(EDP)[19] as a measurement factor. To calculate this, total energy consumption and the total progress of each core are required.

Especially on power consumption, this data is stored in system registers, so it can obtained from reading these registers via telnet protocol.

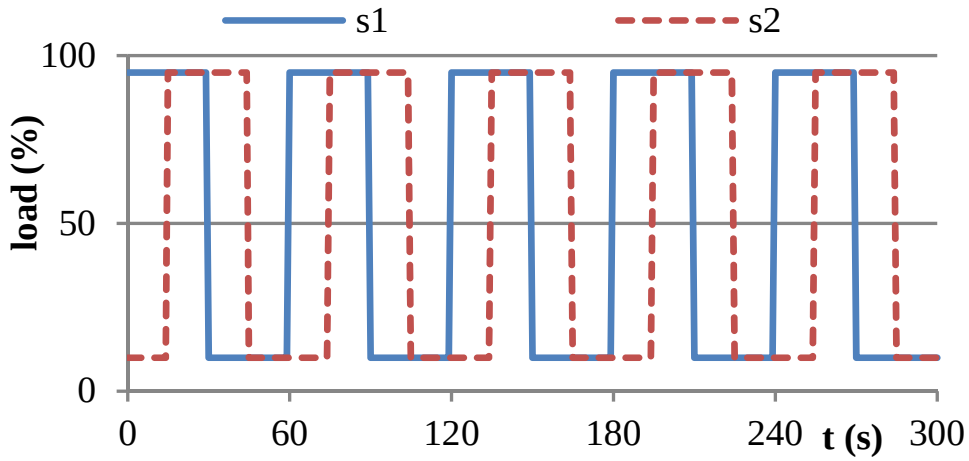
The workload scenarios comprise a number of synthetic and real-world workloads mimicked after desktop usage pattens[20].

4.2 Results

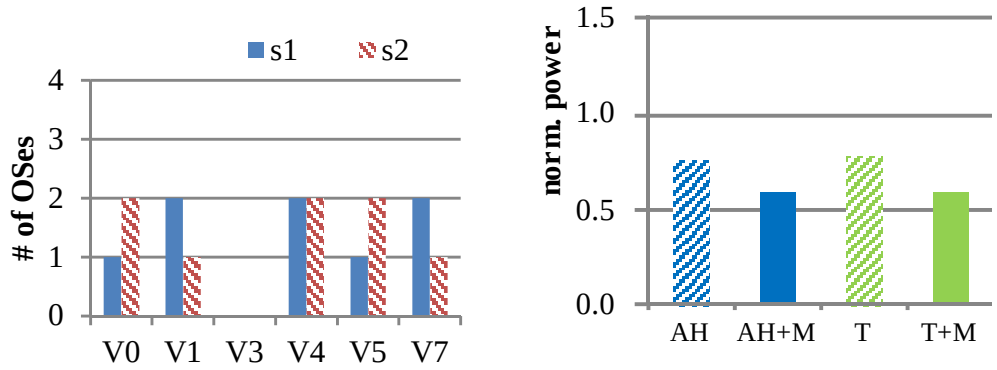
The setup and the results of the first benchmark scenario are shown in Figure 4.1 and Figure 4.2. The load pattern shown in Figure 4.1(a) consists of two simple periodic synthetic workloads that alternate between 10% and 90% load. The second workload `s2` is slightly time-shifted compared to `s1`. The initial OS distribution onto the different voltage domains of the SCC is shown in the left chart of Figure 4.1 (b). Each domain initially contains three or four OSes of both load patterns.

The results of running this first benchmark scenario are shown in Figure 4.1 (b) - (c). Figures 4.1 (b) (right-hand chart) and (c) show the normalized power consumption, the normalized performance, and the normalized performance per watt, respectively, for the `Allhigh` and the `TileIndiv` policy, denoted `AH` and `T`, without and with (appended `+M` postfix) OS migration.

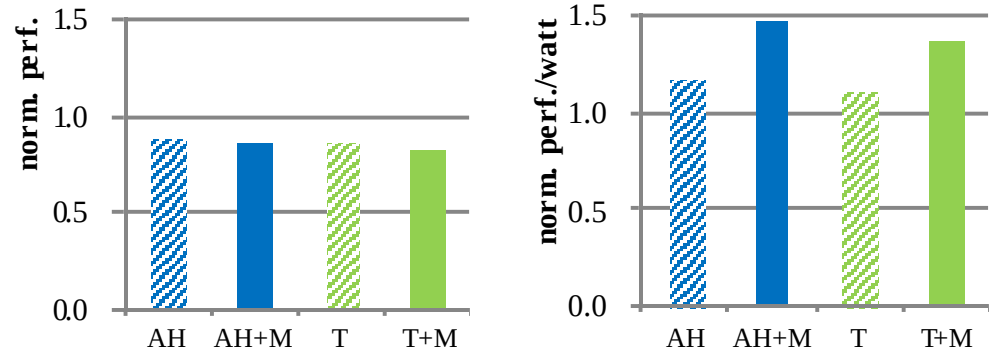
We observe that both DVFS only and DVFS with migration suffer from a performance loss. In the case of DVFS only, there are two reasons for this loss: first, a voltage change operation of an island stops execution on all cores, adjusts the voltage, and then resumes the core clock. This process is implemented in hardware and takes about 10ms per voltage change. The second reason for reduced performance is observed when the workload prediction model fails to foresee a sudden raise in the load and selects a too low operating frequency for a workload. OS migration also requires stopping and resuming all involved cores during migration (3ms). Additionally, the changes in the routing tables are propagated to all active OSes through external interrupt; processing these interrupts on the individual cores also causes a minimal performance overhead.



(a) Load pattern

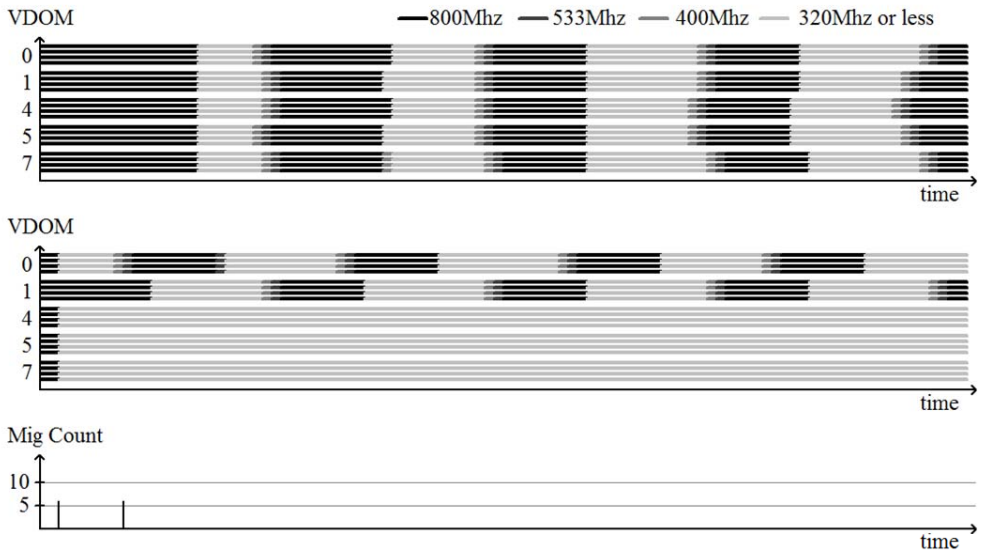


(b) Load distribution & power consumption

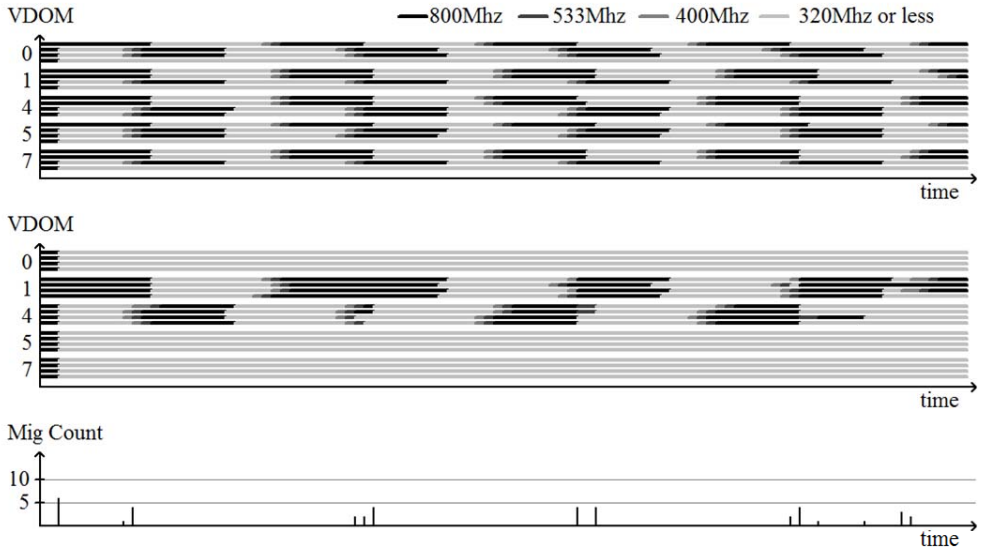


(c) performance & performance per watt

Figure 4.1: Simple alternating synthetic load



(a) Requested Frequency map - `Allhigh`



(b) Requested Frequency map - `TileIndiv`

Figure 4.2: Requested Frequency map dependent on management policies

Nonetheless, as can be seen from the normalized power consumption in Figure 4.1 (b) on the right-hand side, the reduction in power by far outweighs the loss in performance. In terms of performance per watt (right-hand of Figure 4.1 (c)), both DVFS only and DVFS with migration show better results. In particular, the proposed method of combining DVFS with OS migration achieves about a 30% improved performance per watt compared to DVFS-only policies.

Figures 4.2 (a) and (b), finally, visualize the effect of DVFS only and DVFS with migration on the individual voltage domains' frequency settings for the two DVFS policies `Allhigh` and `TileIndiv`. The frequency over time is shown for each voltage domain for DVFS only (upper part) and DVFS with 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.

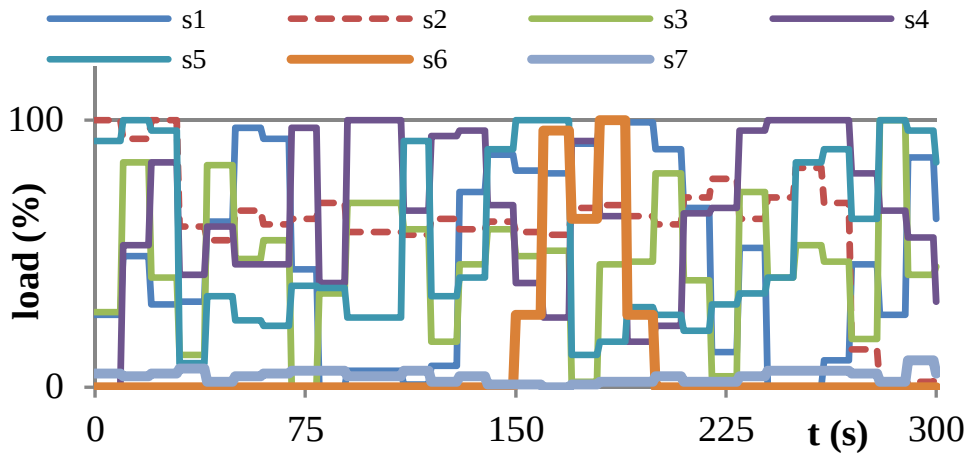
Comparing DVFS and DVFS+migration with the `Allhigh` DVFS policy in Figure 4.2 (a) clearly shows how migration is able to group OSes with similar performance characteristics together and thus select voltages that are closer to the optimal value. In the `Allhigh` policy in particular, if only one core in a particular voltage domain requests a frequency of 800MHz and thus the highest voltage setting, the entire domain will run at $1.1V$. This can be clearly seen by the dark parts in the figure. Since the OSes are evenly spread over all voltage domains, with DVFS only all domains run at maximum voltage as long as only of the load patterns is at 90%. In comparison with DVFS+migration, we clearly observe that the migration policy first gathers all OSes into the first two domains, `vdom0` and `vdom1`, also visibly by the first spike in the number of migrations. About 30 seconds into the benchmark, the OSes running load pattern `s1` drop to 10% load which causes another batch of migrations and results in grouping the OSes running the same load pattern together. After this, the OSes running in the same voltage domain observe similar load patterns and no more migrations are necessary. The DVFS policy can select the appropriate

voltage and frequency for the first two domains.

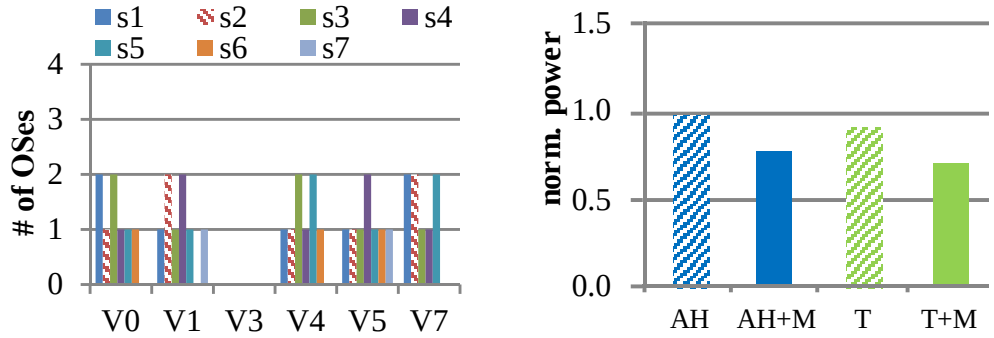
For the `Tile` DVFS policy we observe a similar pattern (Figure 4.2 (b)). Here, the frequency can be set on a tile-basis which is clearly visible by the different frequency settings within the different voltage domains. Again, DVFS only cannot consolidate OSes with similar load patterns into few voltage domains, resulting in voltage settings that are too high for most cores in a domain. DVFS with migration, on the other hand, groups all OSes into `vdom1` and `vdom4`. We see that migration fails to group the OSes running identical load patterns into distinct domains at first which causes some migration activity after about one third of the benchmark's runtime. From then on, the two load patterns are nicely separated. Even though the load patterns are perfectly synchronized at the beginning of the run, the overhead of DVFS and migration causes them to drift apart slowly which then again triggers migrations. For real-world benchmarks without perfectly recurring load patterns this is not an issue.

Figure 4.3 and Figure 4.4 show the setup and the results of a real workload scenario. Seven different load patterns obtained from profiling data of graduate students' computers, `s1` to `s7`, are assigned to a total of 40 OSes and initially placed onto the different voltage domains (Figures 4.3 (a) and left-hand of (b)).

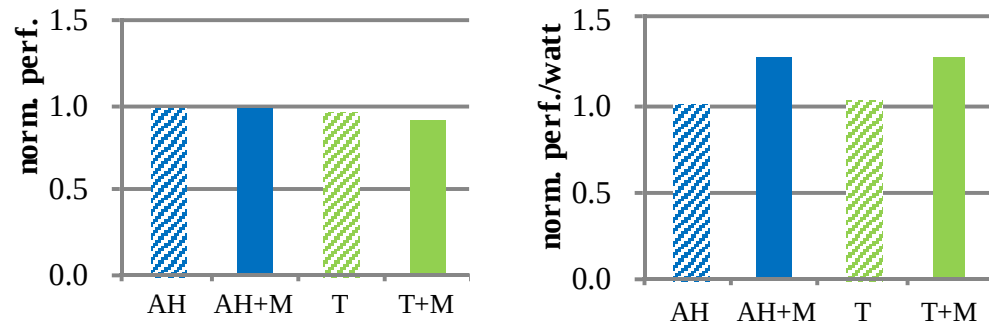
The results of this real-world benchmark scenario are shown in Figure 4.3 (b) - (c). Figures 4.3 (b) (right-hand chart) and (c) show the normalized power consumption, the normalized performance, and the normalized performance per watt, respectively, for the `Allhigh` and the `TileIndiv` policy, denoted `AH` and `T` without and with (appended `+M` postfix) OS migration. We observe that with real load patterns the performance loss is not as pronounced as with the synthetic load pattern. This is because real benchmarks have smoother transitions between busy and idle periods whereas the synthetic benchmark repeatedly jumps from 10 to 90% and back to 10%. The workload estimation, based on the weighted average of a sliding window over the observed load, predicts the required performance for the real load patterns much more accurately then in



(a) Load pattern

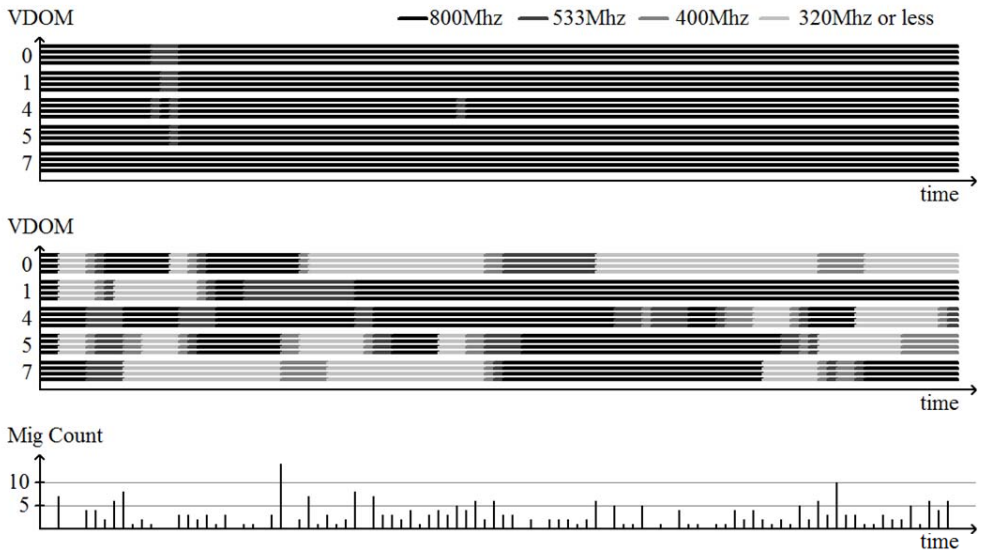


(b) Load distribution & power consumption

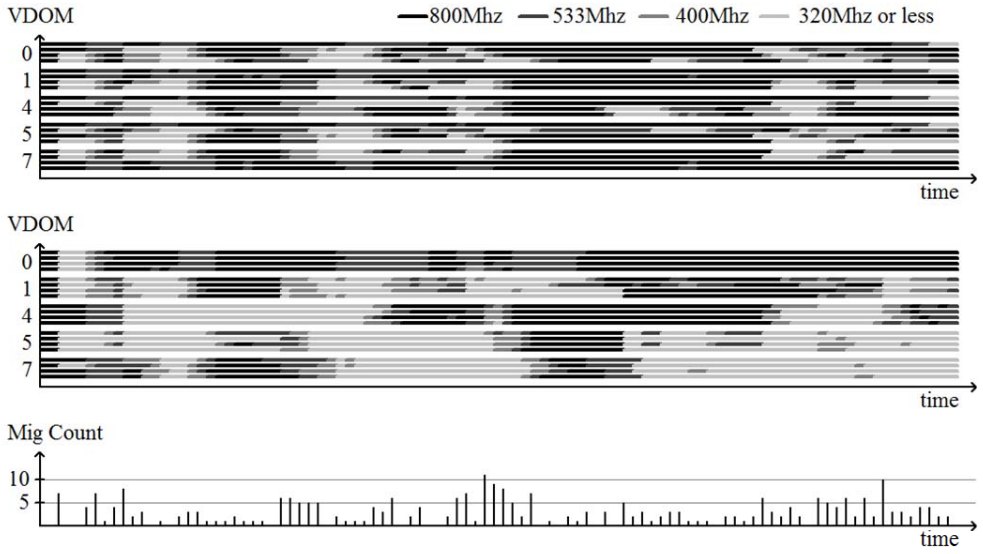


(c) performance & performance per watt

Figure 4.3: Results for a real-world load pattern.



(a) Requested Frequency map - `Allhigh`



(b) Requested Frequency map - `TileIndiv`

Figure 4.4: Requested Frequency map dependent on management policies

the case of synthetic loads. The power savings of DVFS with migration compared to DVFS only range from 25 to 30% which results in an increase of the performance per watt by about the same ratio.

The parts (a) and (b) of Figure 4.4 again visualize the operating frequencies of the different voltage domains. With the `Allhigh` DVFS policy without migration, all voltage islands run at the highest frequency for almost all the time. With OS migration, however, OSs exhibiting similar load patterns are grouped together and despite the `Allhigh` DVFS policy, most voltage domains are able to operate at considerably lower frequencies than with DVFS only. Due to the irregularities of the different load patterns the migration manager fails to consolidate only OSs exhibiting the same load into one voltage domain. This leads to a significantly increased number of migrations, however, since the migration overhead is very small the performance loss caused by the migration is of no consequence.

A similar picture is presented in Figure 4.4 (b) for the `TileIndiv` DVFS policy. Since the frequency is regulated individually for each frequency domain (each tile), the DVFS only policy is able to reduce the operating frequency for tiles requesting less than the maximum frequency. Note, however, that the voltage domains operate at maximum voltage for almost the entire duration of the benchmark because the required voltage is determined by the highest frequency in each domain. The results for the `TileIndiv` DVFS policy with migration display the grouping of similar workloads into voltage domains. Again, it is difficult to clearly identify and separate the different load patterns, nevertheless, OS migration is able to group similar loads together which allows the DVFS manager to select both lower voltages and frequencies.

Figure 4.5, finally, displays the normalized power, normalized performance, and normalized performance per watt over the baseline, respectively, for the `Allhigh` and the `TileIndiv` policy, denoted `AH` and `T`, without and with (appended `+M` postfix) OS migration for five different benchmark scenarios. The

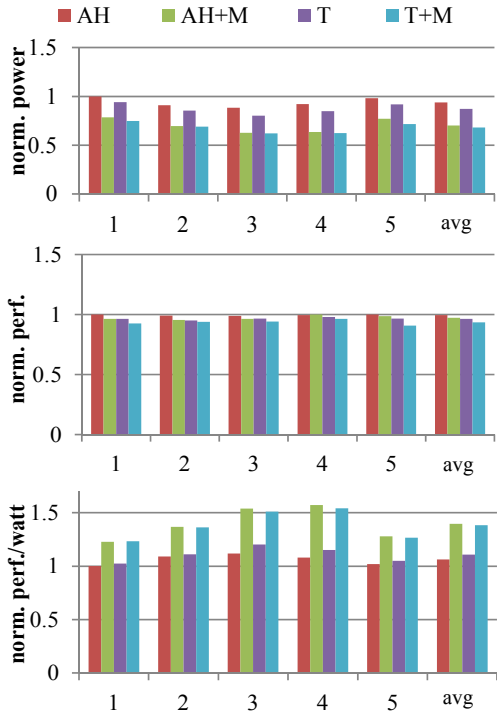


Figure 4.5: Results for different benchmark scenarios.

first two scenarios are constructed from synthetic benchmarks whereas the other three benchmark scenarios are based on real workload patterns. The last group of bars shows the arithmetic average over all benchmark scenarios.

Independent of the workload at hand, migration OSes before applying a DVFS policy results in a significantly reduced power consumption at the expense of a very moderate performance degradation. The increase in performance per watt ranges from 20 to 50% with an average increase of 32% for the `Allhigh` and 25% for the `TileIndiv` DVFS policy. The `TileIndiv` policy allows more fine grain control over the frequency, which is why the effect of migration is a bit less than in the case of `Allhigh`. Overall, the results show that combining OS migration with DVFS is feasible for CMPs and achieves significantly better results than a DVFS-only policy.

Chapter 5

Related Work

There is a significant amount of work focusing on the design and implementation of power management techniques for CMPs. This thesis mainly focuses on the independent workloads(i.e., OSes) executing in a space-shared manner on the CMP and on exploiting the hardware capabilities of existing and future many-core systems with regard to coarse-grained voltage and/or frequency domains.

Some research has focused on exploiting idle periods, Meisner et al propose PowerNap [21] and DreamWeaver [22]. Both techniques assume quick transition between on-state and off-state with hardware support; the letter work implemented wake-up events to increase the sleep periods. Our approaches is quite different with that kind of implementation. In some situations like frequently changing the state, this feature will leads to longer the execution time. In our approaches, we try to choose the optimal voltage/frequency while runtime.

A number of researchers have proposed heterogeneous power management techniques for CMPs [15, 14, 13, 23, 24]. Qiong et al [15] propose the Thread Shuffling, the method of combining DVFS features and thread migration in one operating system. Isci et al [13] apply different DVFS policies under a given

power budget and show that their best policy performs almost as good as an oracle policy having some limited knowledge of the future. Meng et al [23] propose an adaptive power saving strategy that adheres to a global chip-wide power budget through run-time adaptation of configurable processor. Rangan et al [24] propose ThreadMotion, a technique that moves threads around in order to improve power consumption. Their approach requires hardware support to move threads quickly from one core to another. Our approach is similar, but it can be implemented on an available CMP without additional hardware support.

For implementing the hierarchical power management, we get an idea of agent-based framework [17]. Ebi et al proposed agent-base approach for managing thermal issue. In every period, agent operated on each core can communicate adjacent cores. Unlike this, our approach is to transfer the data globally through global shared memory. So it can communicate not only neighbor cores but ones far away from origin.

The work most closely related to ours has been presented by Ioannou et al [14]. In their work, a hierarchical power manager for the Intel SCC is proposed. And they also described several domain-specific policy. A phase detector implemented on their framework detects application phases to exploit DVFS better. In contrast to their work, we add zero-copy OS live migration and can thus achieve significantly improved performance/watt ratio from grouping similar works in one domain.

Chapter 6

Conclusion

Current and future many-core chips are likely to support DVFS with voltage and frequency domains at a coarser level than individual cores. As a consequence, cores with similar performance requirements should be grouped in one domain to achieve optimal results when managing the system's power resources. Grouping similar workloads requires moving the workload from one core to another core (or set of cores). This is standard practice for multi-core OSes executing several applications; for independent OSes running independently on different cores the large overhead of copying the volatile state of the OS is often prohibitive.

In this work, it shows that features of many-core systems allow zero-copy OS migration and that combining zero-copy OS migration with DVFS achieves significantly improved performance/watt ratios on a real system. Our experiments reveal that zero-copy OS migration has a lower latency than voltage changes; therefore, it is, in fact, more beneficial to move OSes from one voltage domain to another voltage domain than changing the voltage of the domain.

We have implemented the zero-copy OS migration in Linux running on the Intel SCC. Experiments show that, compared to a state-of-the-art hierarchical

power management for many-core chips exploiting DVFS, the proposed method achieves, on average, around a 27% improved performance per watt ratio for a wide range of workloads.

Bibliography

[1] Gordon E Moore et al. Cramming more components onto integrated circuits, 1965.

[2] Robert H Dennard, Fritz H Gaensslen, V Leo Rideout, Ernest Bassous, and Andre R LeBlanc. Design of ion-implanted mosfet's with very small physical dimensions. *Solid-State Circuits, IEEE Journal of*, 9(5):256–268, 1974.

[3] Hadi Esmaeilzadeh, Emily Blem, Renee St Amant, Karthikeyan Sankaralingam, and Doug Burger. Dark silicon and the end of multicore scaling. In *Computer Architecture (ISCA), 2011 38th Annual International Symposium on*, pages 365–376. IEEE, 2011.

[4] Intel Corporation. Enhanced intel speedstep technology for the intel pentium m processor - white paper, March 2004.

[5] Advanced Micro Devices. Amd powernow! technology: Dynamically manages power and performance, 2000.

[6] Susanne Albers. Energy-efficient algorithms. *Communications of the ACM*, 53(5):86–96, 2010.

[7] Hwisung Jung and Massoud Pedram. Supervised learning based power management for multicore processors. *Computer-Aided Design of In-*

tegrated Circuits and Systems, IEEE Transactions on, 29(9):1395–1408, 2010.

[8] Shie Mannor, Branislav Kveton, Sajid Siddiqi, and Chih-Han Yu. Machine learning for adaptive power management. *Autonomic Computing*, 10(4):299–312, 2006.

[9] Matthew Curtis-Maury, Ankur Shah, Filip Blagojevic, Dimitrios S Nikolopoulos, Bronis R de Supinski, and Martin Schulz. Prediction models for multi-dimensional power-performance optimization on many cores. In *Proceedings of the 17th international conference on Parallel architectures and compilation techniques*, pages 250–259. ACM, 2008.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] Vaibhav Jain. *Fast Process Migration on Intel SCC*. PhD thesis, Arizona State University, 2013.

[17] Thomas Ebi, M Faruque, and Jörg Henkel. Tape: Thermal-aware agent-based power econom multi/many-core architectures. In *Computer-Aided Design-Digest of Technical Papers, 2009. ICCAD 2009. IEEE/ACM International Conference on*, pages 302–309. IEEE, 2009.

[18] 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.

[19] JHI Laros, Kevin Pedretti, Suzanne M Kelly, Wei Shu, Kurt Ferreira, JV Dyke, and Courtenay Vaughan. *Energy-efficient high performance computing*. Springer, 2013.

[20] Cao LeThanhMan and M. Kayashima. Desktop workload characteristics and their utility in optimizing virtual machine placement in cloud. In *Cloud*

Computing and Intelligent Systems (CCIS), 2012 IEEE 2nd International Conference on, volume 01, pages 333–337, Oct 2012.

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

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

[23] 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.

[24] 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.

요약

최근에 연구용으로 사용되는 매니코어 기반의 시스템에는 core 자체가 소모하는 전력과 chip에 가해지는 열에 대한 trade-off를 만족시키기 위한 전력 관리 체계가 필요하다. 학계에 발표된 대부분의 연구는 전체 core에서 돌아가는 병렬처리 프로그램에 초점을 맞춰서 전력을 효율적으로 관리하는데 초점을 맞추고 있다. 이 논문에서는 매니코어 시스템상에서 독립된 운영체제가 돌아갈 경우 효율적으로 전력을 관리할 수 있는 방법에 대해서 다뤘다. 이 논문에 다룬 Framework는 memory address indirection 이나 global shared memory와 같은 하드웨어적인 기능을 포함한 매니코어 시스템이라면 구현할 수 있는 형태로 되어 있다. 이 논문에서 주로 다룬 기술인 Zero-copy OS migration은 도메인 단위로 관리되는 시스템에서 같은 성능을 요구하는 Core들끼리 묶어서 관리하게 함으로써 기존에 언급되어 있던 도메인 단위에서의 전력 관리 정책과 함께 적용할 경우 조금 더 효율적인 성능을 얻을 수 있다. 실제 실험에서는 Intel에서 출시된 Single-chip Cloud Computer (SCC) 에서 40개의 리눅스 운영체제가 각 core에 돌아가고 있는 상태에서 계층적으로 전력 관리에 필요한 Data를 주고받을 수 있도록 구현되었으며, 대부분의 시나리오에서 평균적으로 27% 정도의 성능 향상을 얻을 수 있었다.

주요어: 매니코어 시스템, OS Migration, 전력관리, SCC
학번: 2013-20738