M.S. THESIS

NUMA-aware Hierarchical Power Management for Chip Multiprocessors

NUMA 구조를 인지한 칩 멀티프로세서를 위한 계층적 전력 관리

August 2017

DEPARTMENT OF COMPUTER SCIENCE ENGINEERING
COLLEGE OF ENGINEERING
SEOUL NATIONAL UNIVERSITY

Changmin Ahn

M.S. THESIS

NUMA-aware Hierarchical Power Management for Chip Multiprocessors

NUMA 구조를 인지한 칩 멀티프로세서를 위한 계층적 전력 관리

August 2017

DEPARTMENT OF COMPUTER SCIENCE ENGINEERING
COLLEGE OF ENGINEERING
SEOUL NATIONAL UNIVERSITY

Changmin Ahn

NUMA-aware Hierarchical Power Management for Chip Multiprocessors

NUMA 구조를 인지한 칩 멀티프로세서를 위한 계층적 전력 관리

지도교수 Bernhard Egger

이 논문을 공학석사 학위논문으로 제출함

2017 년 04 월

서울대학교 대학원

컴퓨터 공학부

안 창 민

안 창 민의 공학석사 학위논문을 인준함

2017 년 06 월

위 원 장	Srinivasa Rao Satti	(인)
부위원장	Bernhard Egger	(인)
위 원	허 충 길	(인)

Abstract

Traditional approaches for cache-coherent shared-memory architectures running symmetric multiprocessing (SMP) operating systems are not adequate for future many-core chips where power management presents one of the most important challenges. In this thesis, we present a hierarchical power management framework for many-core systems. The framework does not require coherent shared memory and supports multiple-voltage/multiple-frequency (MVMF) architectures where several cores share the same voltage/frequency. We propose a hierarchical NUMA-aware power management technique that combines dynamic voltage and frequency scaling (DVFS) with workload migration. A greedy algorithm considers the conflicing goals of grouping workloads with similar utilization patterns in voltage domains and placing workloads as close as possible to their data. We implement the proposed scheme in software and evaluated it on existing hardware, a non-cache-coherent 48-core CMP. Compared to state-of-the-art power management techniques using DVFS-only and DVFS with NUMA-unaware migration, we achieve on average, a relative performance-per-watt improvement of 30 and 5 percent, respectively, for a wide range of datacenter workloads at no significant performance degradation.

Keywords: Many-core Architecture, NUMA, Scheduling, DVFS, Energy Efficiency
Student Number: 2015-22902

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The past decade has brought a shift from high-performance single- or dual-core processors to chip multiprocessors (CMPs) integrating from a few tens up to a thousand cores into one processor die [1,2,8,9,30,31]. Chip-level power and thermal constraints are now one of the primary design constraints and performance limiters [2]. Higher power consumption not only leads to increased energy cost but also causes higher die temperatures that adversely affect chip reliability and lifetime [7]. Even in commodity processors, such as Intel processors based on the Haswell, micro architecture power constraints result in reduced per-core performance when multiple cores are active [16].

To reduce overall chip energy consumption, modern processors provide hardware support to dynamically lower the operating voltage and clock frequency of clocked resources through dynamic voltage and frequency scaling (DVFS). Depending on the utilization of processor cores, for example, core voltages and frequencies are adjusted in order to minimize power consumption while, at the same time, meeting performance requirements [5]. On CMPs, the required logic for individually controlling the voltage for each core is becoming too costly [22]; instead, cores are physically clustered

into voltage and frequency domains leading to so-called *multiple-voltage/multiple-frequency* (MVMF) designs where all cores within a domain run at the same voltage or frequency [7, 14, 15, 34].

Managing power on CMPs has recently received considerable attention [6, 10, 11, 13, 18, 25, 28, 32, 33, 35]. Existing research on power management for CMPs foremost focuses on minimizing power consumption or optimizing performance for a given power budget [10, 18, 25, 28, 33]. Solutions for MVMF architectures combine DVFS with thread migration [6, 17, 19, 21, 33], because co-locating threads with similar performance requirements into the same domain allows for better tailored DVFS settings for that domain [17].

The integration of more and more cores into CMPs poses several other architectural challenges. First, to cope with the increased bandwidth requirements the cores of a CMP are typically connected to several memory controllers by a network-on-chip (NoC). Depending on the location of the core and the accessed memory controller large differences in memory access latency are observed, resulting in a non-uniform memory access (NUMA) architecture on a single chip. The second challenge for CMPs is that maintaining a coherent global view of shared memory in the presence of local caches is becoming difficult. While today's commercial CMPs typically still maintain cache coherency to support existing operating systems and parallel runtimes, the trend goes towards partial or no coherence [2, 15, 31].

In this thesis, we propose a hierarchical power management technique for MVMF CMPs that considers the (not necessarily cache-coherent) NUMA memory architecture. Existing techniques fall short for a variety of reasons. Many works assume per-core DVFS control which limits their applicability to MVMF designs. Researches employing thread migration assume symmetric multiprocessing with one centralized kernel and cannot easily cope with non-coherent memory architectures. Lastly, to the best of our knowledge, no work considers the NUMA properties of CMPs resulting in core

mappings that are not optimal with respect to the locality of the data accessed by individual threads.

The presented power management technique can be applied to monolithic kernels running on a cache-coherent SMP processor as well as non-coherent memory architectures running a distributed micro kernel. The hierarchical design naturally maps to the architecture with per-core utilization monitors, individual frequency controllers for the frequency domains, voltage controllers for the voltage domains, and a central migration controller. The solution is entirely implemented in software and does not require special hardware support. The migration controller computes and orchestrates migration of workloads based on a cost-benefit model. The individual frequency and voltage controllers regulate the frequency/voltage for the controlled domain. A working implementation is provided for the Intel Single-Chip Cloud Computer (SCC) [15] and evaluated with real-world workloads. All experiments and measurements are performed on the architecture itself and thus include overhead incurred by DVFS transitions, cold cache misses, workload migration, and the overhead caused by the different controllers. We compare the proposed technique to a DVFS-only approach [17] and a method with DVFS and migration [21]. On average, we achieve a 54, 33, and 5% higher performance-per-watt ratio over standard Linux, DVFS-only and DVFS with migration at no performance degradation.

We show that, even with complete separation and isolation of processes, it is possible to change the allocation of the physical cores to the applications with almost zero overhead on CMPs and very little support from the application-specific runtime environment. This technique allows us to group, on a global level, cores that exhibit similar load patterns onto voltage and frequency domains before applying DVFS.

In previous work [21], we proposed the design and implementation of a hierarchical power management technique using workload migration for multi-voltage/multi-frequency CMPs. The logical abstraction mimics the physical layout of the CMP (core,

frequency domain, voltage domain, and chip). In summary, the contributions of this work are as follows:

- we analyze the interplay of DVFS with workload migration and propose a data-locality-aware migration heuristic.
- we describe and evaluate a proof-of-concept software implementation for the Intel SCC [15] architecture running up to 40 different real-world workloads. All measurements are performed on a real system.

The remainder of this thesis is organized as follows: Chapter 2 discusses the problem formulation and related work. Chapter 3 describes the hierarchical power management framework, and Chapter 4 discusses the DVFS and migration policies in detail. Chapter 5 describes the implementation for the Intel SCC. The experimental setup and the results are presented in Chapters 6 and 7, respectively. In Chapter 8, we talk about the limitations and expected improvement of this work. Finally, Chapter 9 concludes this thesis.

Chapter 2

Motivation and Related Work

2.1 Characteristics of Chip Multiprocessors

Technology scaling, thermal limitations and the insight that doubling the logic in a processor core only delivers about 40% more performance, known as Pollack's Rule, have led to the introduction of chip multiprocessors with tens or hundreds of cores on one processor die [3, 4]. Architectural characteristics of today's and future many-core CMPs impose new restrictions on the design and implementation of operating systems in particular with respect to workload scheduling and power management.

The cores of a CMP are typically organized in a 2-D array. The Kilocore processor, for example, arranges its 1000 cores on a 32x32 grid [2]. A network-on-chip (NoC) interconnects the cores of a CMP and is used both for inter-core communication and accesses to memory and external devices such as network or storage controllers. The flow of data packets through the NoC is controlled by routers; this routing comes with a small delay. As a consequence, the distance between the source and the destination (number of hops in the NoC) has a significant effect on the access latency of indi-

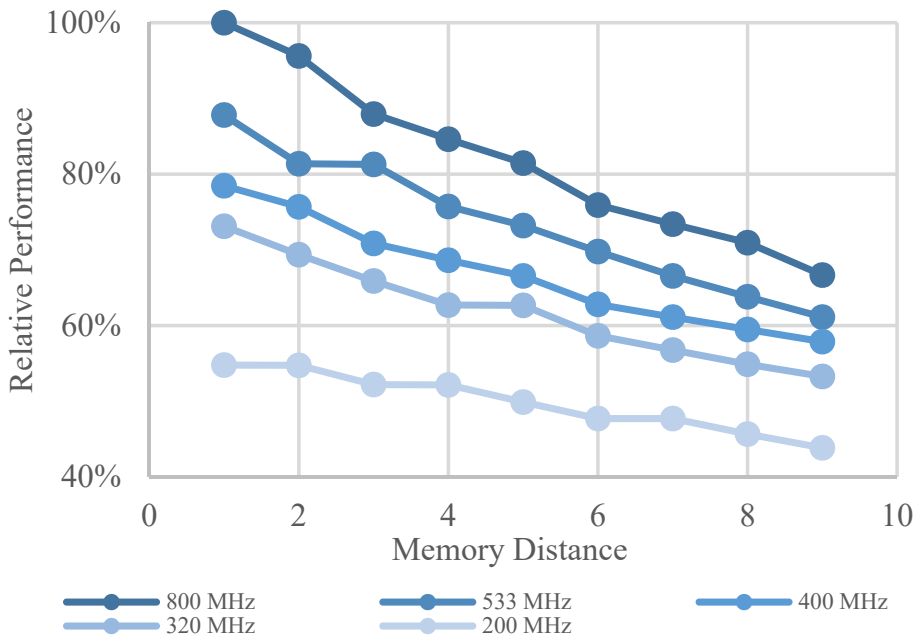
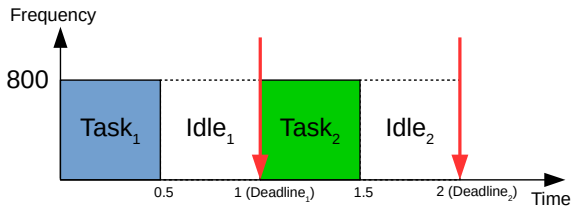


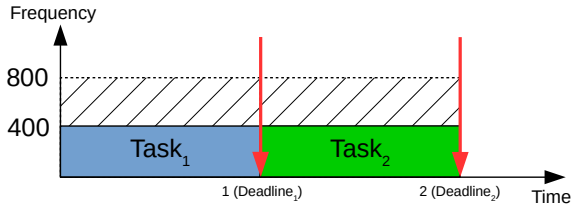
Figure 2.1: Normalized memory bandwidth in dependence on the frequency and the distance from the memory controller.

vidual cores to memory, resulting in a NUMA architecture on a single die. Figure 2.1 shows the relative memory bandwidth of a 48-core CMP, the Intel Single-Chip Cloud Computer (SCC) [15], in dependence the number of hops between the core and the memory controller where the accessed data is. Each criteria represents each frequency level which SCC supports. Y-axis is relative memory performance to the best performance which a core can achieve when it has the highest frequency and the closest location.

The memory organization of CMPs combines several memory controllers to access off-chip data with on-chip local memory in the form of scratchpad memories or local or shared caches. As an example, the Intel SCC processor has four memory controllers and integrates both caches and user-managed scratchpad memory on the die [15]. It is foreseeable that maintaining cache coherency at the hardware level will become



(a) Run tasks as fast as possible.



(b) Run tasks as slow as possible.

Figure 2.2: Potential of DVFS.

extremely challenging with hundreds or thousands of local caches on one die, and most research prototypes integrating many cores do not provide a coherent cached view of global memory anymore [2, 15, 31].

2.2 Dynamic Voltage and Frequency Scaling

$$P_{chip} = P_{dyn} + P_{SC} + P_{leak} \tag{2.1}$$

The power consumption of a chip includes dynamic power consumption, short-circuit power consumption, and power loss of transistor leakage currents (Equation 2.1). In these factors, short-circuit power consumption and transistor leakage are steady. So, we can only save dynamic power consumption, $P_{dynamic}$, calculated by equation from [29]:

$$P_{dynamic} = ACV^2 f \tag{2.2}$$

where A is gate activity factor, C is capacitance, V is voltage, and f is frequency.

From Equation 2.2, because A and C are constants, we save power by lower voltage and frequency. This technique which changes voltage and frequency of a chip in runtime is dynamic voltage and frequency scaling, which called DVFS. But, when we use DVFS, a chip has a necessary condition when supports a frequency. For each frequency level, a chip has to support some minimum voltage matching with the frequency level.

The potential of DVFS is represented in the Figure 2.2. These two graphs are represent running time and frequency when running same tasks within deadlines at different frequencies. Then, with Equation 2.2 and voltage and frequency values from real system, 1.1 V to run frequency 800 MHz and 0.8 V for 400 MHz, we can calculate the energy for Figure (a) and Figure (b). For the Figure (a) which it runs tasks without DVFS, we can get $(1.1\,\mathrm{V})^2 * 800\,\mathrm{MHz} * 0.5\,\mathrm{sec} * 2 = 48.400\,\mathrm{V^2MHz\,sec}$, even if we assume 0 power consumption at idle stage, which is not true. For the Figure (b) which it runs tasks with DVFS, we can get $(0.8\,\mathrm{V})^2 * 400\,\mathrm{MHz} * 1\,\mathrm{sec} * 2 = 25.600\,\mathrm{V^2MHz\,sec}$, which save almost half of power when we run the tasks without DVFS.

2.3 Power Management on CMPs

In the past, DVFS has proved to be an effective technique to limit power dissipation, and an enormous body of research exists on that topic. On CMPs, equipping every single core with a voltage regulator is becoming too costly [22]; multiple-voltage/multiple-frequency (MVMF) designs are being proposed. In a MVMF design, cores in the same voltage or frequency domain share the same voltage or frequency, respectively. Since the range of valid frequencies depends on the supply voltage, one voltage domain typically contains one or more frequency domains (Figure 2.3).

The additional constraints imposed by the CMP architectures and in particular their

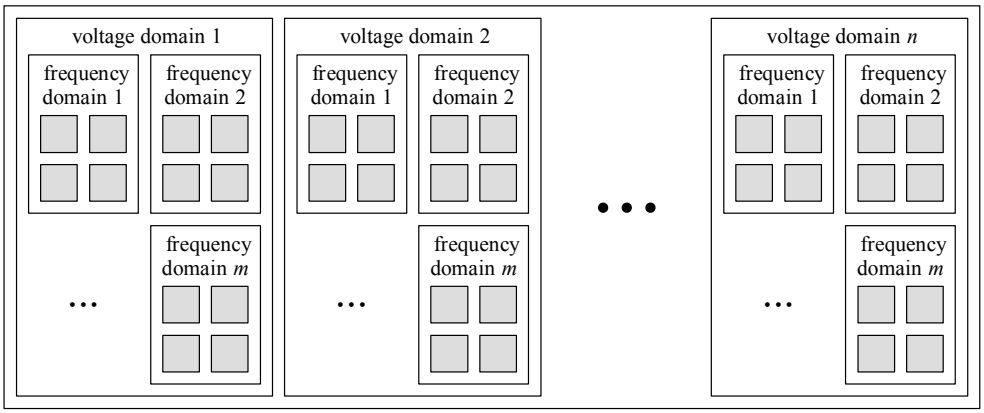


Figure 2.3: Frequency and voltage domains in many-core CMPs.

power management capabilities require new approaches power management. Existing designs that assume per-core DVFS control, cache-coherent global shared memory, and uniform access latency to the memory are not able to cope with today's characteristics of CMPs. What is required is a power management approach that considers all aspects of modern CMPs. In addition, power management has to go hand-in-hand with workload scheduling. From a power management perspective only, cores with similar performance requirements need to be grouped together in voltage and frequency domains in order to achieve optimal power savings. On the other hand, the NUMA characteristics of the chip require the scheduler to place workloads as close as possible to the accessed memory controllers. In addition, the scheduler and the power manager may need to adhere to user-defined performance goals such as improving performance, maintaining Quality of Service (QoS), dissipate heat evenly, or minimize power for a given budget. In this paper, we describe our solution of a cooperative and hierarchical power management technique that balances the conflicting goals of scheduling and power management in order to achieve a better energy efficiency at no or minimal performance degradation.

2.4 Related Work

There is a significant amount of work focusing on the design and implementation of power management techniques for CMPs. One line of related work considers heterogeneous CMP designs with the goal of minimizing power consumption with no or minimal performance loss. Kumar *et al.* [23] propose heterogeneous CMPs composed of cores with an identical ISA but different power characteristics. Ghiasi [12] proposes CMPs with cores executing at different frequencies. Both works show that such systems offer improved power consumption and thermal management. Our work differs in that our approach modifies the voltage/frequency of cores dynamically, without being bound to certain hardware heterogeneity.

Another line of research has focused on exploiting idle periods. Meisner *et al.* propose PowerNap [26] and DreamWeaver [27]. Both assume hardware support for quick transitions between on- and off-states; the latter work batches wake-up events to increase the sleep periods. Our work is orthogonal to such approaches.

A number of researchers have proposed heterogeneous power management techniques for CMPs [6, 17, 18, 20, 24, 25, 28, 33].

Li *et al.* [24] provide an analytical model and experiments to show to what extent parallel applications can be parallelized given a power-budget. Isci *et al.* [18] apply different DVFS policies under a given power budget and show that their best policy performs almost as good as an oracle policy having limited knowledge of the future. Meng *et al.* [28] propose an adaptive power saving strategy that adheres to a global chip-wide power budget through run-time adaptation of configurable processor cores. They integrate multiple power optimization techniques (in this case DVFS and cache resizing) into single power management unit. To optimize performance of a CMP under given power budget by using power optimization techniques, they use a greedy search algorithm to select a technique. They introduce models to predict performance

of a core after apply power optimization techniques. Especially, for cache resizing, they achieve reasonable accuracy. But unlike us, they target per-core DVFS supported system which doesn't need thread migration.

Rangan *et al.* [33] propose ThreadMotion, a technique that moves threads around in order to improve power consumption. In a multi-core system which has homogeneous cores with heterogeneous power setting, they use thread migration to exploit fine variation in program behavior which DVFS cannot exploit because DVFS is to slow. They use a coarse-grained prediction-driven approach and a last-level cache miss driven approach to trigger thread migration. This technique requires hardware support to quickly move threads from one core to another. Our approach is similar but can be implemented on available CMPs without extra hardware support.

Cai *et al.* [6] propose Thread Shuffling, a technique that migrates hardware contexts around to exploit non-critical threads; non-critical threads can then be executed at reduced speed. They identify critical threads by using meeting point thread characterization. They assume a core has multiple hardware context and per-core DVFS which corresponds with per-tile DVFS. This work can be applied when a single parallel application is running on the system. In this thesis, we focus on independent OSes as opposed to threads within a parallel application.

Ma *et al.* [25] propose a scalable solution aiming at a mixed group of single-threaded and multi-threaded applications. They introduce hierarchical management to reduce the complexity of scheduling. Their framework periodically groups cores which running same applications. It partitions chip power budget between groups according to power efficiency. Then, it partitions quota again among cores in a group by analyzing thread criticality. They implemented and tested the power controller not only on a simulator but also on a real system. Unlike our approach with is best-effort, they aim at minimal performance reduction while maintaining a global power budget in per-core DVFS systems.

Jha *et al.* [20] propose a hierarchical power management on systems which have per-tile DVFS and shared a last-level cache. They classify threads with DVFS sensitivity and cache behavior. They migrate threads based on its' class. They call this DVFS and cache-aware thread migration (DCTM). This work aims getting best performance under power budget. But our focus is reducing power consumption under satisfying performance target. And our system doesn't have shared caches.

A previous work of our lab is a hierarchical power manager for the Intel SCC [17, 21]. While Ioannou *et al.* [17] apply DVFS to a static workload assignment, Kang *et al.* [21] demonstrate that adding workload migration can yield a significant improvement in the performance/watt ratio. The proposed buyer-seller algorithm used for workload migration, however, fails to consider data locality and thus results in suboptimal core assignments. In this thesis, we follow the overall system architecture of the previous two works but present a new greedy workload placement algorithm that balances the conflicting goals of the scheduler and power management for MVMF CMPs.

Chapter 3

Cooperative Power Management

The proposed cooperative hierarchical power manager combines workload migration with DVFS to achieve optimal power efficiency. We target both CMPs with and without cache-coherent global shared memory. We employ a distributed OS design with small individual kernels running on each core.

3.1 Cooperative Workload Migration

Workloads with similar performance characteristics need to be grouped in frequency / voltage domains to allow for optimal DVFS and, as a consequence, improved power efficiency. As an example, consider two frequency domains with two cores each. In both domains, one core is running at 100% CPU load, the other one is only 10% loaded. To maintain throughput, both domains need to run at the highest frequency f_{max} in order to provide the computing power required by the busy core. Both lightly loaded cores will also run at f_{max} even though theoretically $\frac{1}{10} f_{max}$ would suffice. Workload migration allows us to group the heavily loaded cores into one and the lightly loaded

cores into the other domain. One domain can then run at f_{max}, the other one at $\frac{1}{10} f_{max}$ without sacrificing performance. Taking data locality into consideration complicates the situation. In the above example, one of the busy cores needs to be moved into the domain of the other busy core. This relocation may change the distance of the core from the memory controller holding its data and reduce the throughput on the newly assigned core.

With a distributed OS comprising microkernels that individually schedule the tasks assigned to them, task migration is more difficult to achieve than in global shared memory systems with a monolithic kernel. Since each kernel has its own network ID on the NoC, moving a task from one core to another would disconnect established communication channels. If properly orchestrated, however, the architecture of CMPs allows for dynamic workload re-allocation without side-effects. The idea is to migrate the entire microkernel from one core to another with its entire workload. Since we assume (non cache-coherent) global shared memory, migration of a microkernel only requires moving the volatile state of a core, i.e., its processor state, from one core to another. A greedy algorithm to deal with these potentially conflicting goals of core placement is presented in Chapter 4. Implementation details about microkernel migration on existing hardware are discussed in Chapter 5.

3.2 Hierarchical Organization

The logical structure of the hierarchical power manager reflects the structure of the CMP with separate frequency and voltage domains. 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 with n number of *frequency controllers*. 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.

3.3 Domain Controllers

Each domain, from core to frequency, voltage, and the global chip level, operates 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 sections and a possible implementation is discussed in Chapter 5.3.

3.3.1 Core Controller

The task of the *core controllers* is to monitor and predict the performance of the workload on the associated core. Each microkernel runs a core controller daemon monitoring the performance (load or instructions per clock (IPC)) and the number of memory accesses by periodically querying the performance monitoring unit (PMU). The core controllers also predict the required computational performance based on extrapolated measured data. At regular intervals, the core controllers communicate with their frequency controllers To report the required operating frequency and memory-boundness. The core controllers run on every kernel.

3.3.2 Frequency Controller

For each frequency domain, the *frequency controller* gathers data about the frequencies and workloads from core controllers within its domain, 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 Chapter 4.1).

3.3.3 Voltage Controller

The *voltage controllers* gather data from their frequency controllers and forward it to the chip controller. In addition, the voltage controllers also compute and set the operating voltage of their domains. Note that voltage changes must happen in close collaboration with the frequency controllers because the maximal operating frequency has a linear relationship to the supply voltage. This is particularly important if the voltage of a domain is to be lowered. In that case, the frequency controllers must first reduce the frequency to a value below or equal to the maximal operating frequency of the new supply voltage before the voltage change can occur.

3.3.4 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 levels. The chip controller migrates the microkernels before signaling the voltage controllers to initiate DVFS adjustments.

3.3.5 Location of the Controllers

In a pure software implementation, one kernel per frequency and voltage domain needs to run the respective controller. Similarly, the chip controller also runs on one of the cores. Since we migrate entire kernels, it is impossible to designate the kernel for each of the controllers offline. Instead, we run an instance of each controller in *every* kernel. The frequency, domain, and chip controller in a kernel are activated and deactivated

depending on the physical location on the CMP. In other words, the functionality is pinned to the physical core and not the kernel. For example, if the frequency controller for frequency domain 1 is pinned to core 0, the kernel that is currently running on core 0 will activate its frequency controller. Such a scheme has the additional benefit that no discovery service is needed to find the controllers.

Chapter 4

DVFS and Workload Migration Policies

In this thesis, we focus on optimizing the *performance per watt* ratio of the overall chip. Other policies, such as, for example, even heat dissipation or adhering to a given power budget, can also be implemented within the framework of the presented collaborative hierarchical framework and are part of future work.

The power management policy is implemented in the global domain manager. The migration and DVFS algorithms are invoked at regular intervals by the scheduler. The DVFS and migration policy, though the former depends on the latter, are completely separated to be able to combine different migration and DVFS policies freely.

4.1 DVFS Policies

We implement two DVFS policies employed in the hierarchical power manager for CMPs proposed by Ioannou *et al.* [17] and employed by Kang *et al.* [21]. Both works have been implemented and evaluated on the same hardware and provide a good reference point.

- *Allhigh*: this DVFS policy runs all cores within a *voltage domain* at the highest frequency requested by the subordinate frequency domains. The supply voltage is set to the lowest voltage that supports the requested frequency.

- *Tile*: grants the requested frequency to each *frequency domain* and set the voltage accordingly. In [17] this policy is denoted *Simple*, we follow Kang's nomenclature here.

In both policies, the supply voltage of the domain is set to the lowest voltage that supports the highest frequency of any of the subordinate frequency domains.

We have not implemented the *Alllow* and *Allmean* policies since they sacrifice too much performance in return for power savings.

4.2 Phase Ordering and Frequency Considerations

In order to achieve maximum power savings, migration should occur before applying DVFS. The frequency of migration, and voltage/frequency changes is determined by the cost of the individual operations. The time required for migration is largely unaffected by the number of kernels that are migrated because the involved kernels can migrate in parallel. Kernels involved in a migration flush their caches and are briefly stopped, while the other kernels continue to run. Voltage changes incur a not insignificant overhead because all cores in the domain are stopped during the rather long voltage adjustment. Frequency changes, on the other hand, are almost instantaneous and can be performed often. On our specific target architecture, the Intel SCC, we have measured the following latencies: $\leq 3ms$ for migration, $\leq 10ms$ for voltage changes, and a few thousand cycles for frequency changes. We perform workload migration and DVFS at a 3 second interval, because of high latency for voltage changes. Besides, the SCC only supports one voltage change at a time; i.e., different domains cannot change the voltage in parallel. Nevertheless, in our experiment, workload migration and voltage

changes can be performed at every step. Chapter 7 discusses the benchmarks and results in more detail.

4.3 Migration of Workloads

As outlined in Chapter 3.1 workloads which have similar load pattern need to be grouped in order to achieve good power savings. A naïve algorithm would be to sort the workloads by their performance requirements and then assign them into the voltage and frequency domains. While the resulting migration of workload to domains is optimal for CPU-bound applications, the algorithm fails to consider the overhead of kernel migration. The migration of a kernel itself is very quick (measurements on a real system yield an overhead of $\leq 3ms$). However, each time a kernel is migrated to a different core, the workload running on the kernel will experience cold misses in the local caches that in turn lead to a loss in performance as well as increased memory traffic. To minimize this overhead, the number of migrated kernels should be kept as low as possible.

Due to the NUMA nature of CMPs, kernel migration can have a significant effect on the access latency and bandwidth of memory accesses. Since the data of a migrated workload is not moved, migrating a memory-intensive workload executing on a core close to the memory controller to a core far away from the memory controller can cause a significant drop in memory bandwidth and access latency (Figure 2.1).

4.4 Scheduling Workload Migration

We have two conflicting goals. One of the goals is optimizing power. Another is optimizing memory performance. To solve this problem, we develop a greedy algorithm to schedule workload migration which makes CMP optimize power consumption and memory distance with given performance requirement. When we make every voltage

domain always use the minimum voltage which can support maximum frequency request of workloads in it, the main concepts of our algorithm are as follows:

- for each target frequency level f_{target}, collect T, which are workloads with f_{target}, into minimum number of voltage domains.
- migrate each workload in T, in descending order of memory load, to a core which has minimum distance to memory controller.

In this algorithm, for each frequency level, we first generate voltage domain combination, *Comb*, which has every possible combination of voltage domains to place T. For every voltage domain set, *set*, in *Comb*, we assign each T, in descending of memory intensity, on a core which has minimum distance to memory in *set*.

To evaluate expected energy consumption of each *set*, we introduce a model to evaluate the power state of a chip. We discuss details about this model later in Chapter 4.5. Base on this evaluation model, we select the best voltage domain *set* among *Comb*, and repeat for next target frequency which one step lower than f_{target} until the minimum frequency. After schedule migration, we perform workload migration only if the chip's power status over a threshold rate.

4.4.1 Schedule migration

In this step, we get workload migration schedule for each f_{target} in descending order, as represented line $3-7$ in Algorithm 1. If the schedule is better than the previous result, we save it and give the workload mapping to next f_{target} level (line 4 in Algorithm 1). Then, we use the evaluation model. If and only if the final migration result's power status is over the threshold, we perform workload migration (line $7-8$ in Algorithm 1).

Algorithm 1 Decide Migration

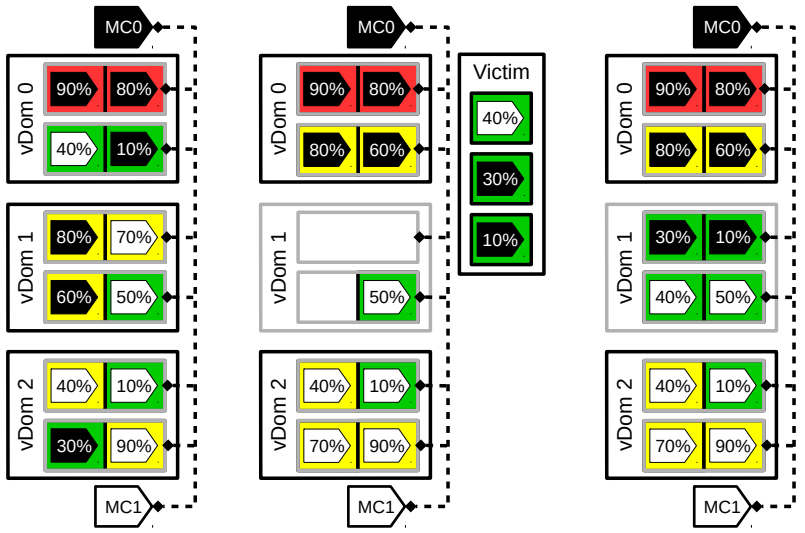
EvalMigBenefit(*migMap*): returns a power rate between original state of chip and *migMap*

```
1:  function DecideMigration(migThreshold)
2:      migMap ← Current core mapping
3:      for each f ∈ Freq_Range do                          ▷ Decending order
4:          tmpMap ← LevelMig(migMap, f)
5:          if EvalMigBenefit(tmpMap) is better than before then
6:              migMap ← tmpMap
7:          end if
8:      end for
9:      if maxMigBenefit > migThreshold then
10:         return migMap
11:     end if
12:     return NULL
13: end function
```

4.4.2 Level migration

For given f_{target}, in this step, we try to minimize the number of voltage domains which have a core that has requested f_{target}. In Algorithm 2, this algorithm collect T (line 3 in Algorithm 2), and calculate how many voltage domains we need to allocate all T. To calculate this, we divide the voltage domains into two groups. When V_l represent a voltage which is minimum to support frequency level l, one group consist of voltage domains which have $V > V_{target}$. We call this group as $vDom_{used}$. Another is a group of voltage domains which have $V \leq V_{target}$, which called $vDom_{left}$.

In the context of minimizing N, the number of voltage domains which have V_{target}, we can achieve this by placing T in $vDom_{used}$. Because this migration won't increase N. From $vDom_{used}$, we collect candidate cores which have a workload with $f \leq f_{target}$, which called $Cand_{in\_used}$. Then, we can calculate N, the minimum number of voltage domains without the number of $vDom_{used}$ by dividing the number of cores in a voltage



(a) Inital status (b) After assign T (c) After assign *victim*

Figure 4.1: Workload migration steps with f_{mid} example.

domain with the ceiling function. This calculation is given as follows

$$N = \left\lceil \frac{|T| - |Cand_{in\_used}|}{|CoresInVoltDom|} \right\rceil \tag{4.1}$$

After get the number of voltage domain we need, we can make combination with N number of voltage domains from $vDom_{left}$ (line 4 in Algorithm 2). Then, we can make complete *set* which T going to be by adding $vDom_{used}$ to each combination (line 6 in Algorithm 2).

For example, let's assume we have target frequency f_{mid} with core mapping like Figure 4.1 (a). In the figure colors red, yellow, and green boxes represent high, middle, and low workloads in a core, and the pentagons represent memory controller which the workload uses with color and amount of memory load with the number in it. We have voltage domain groups, $vDom_{used} = \{vDom0\}$ and $vDom_{left} = \{vDom1, vDom2\}$.

Algorithm 2 Level Migration

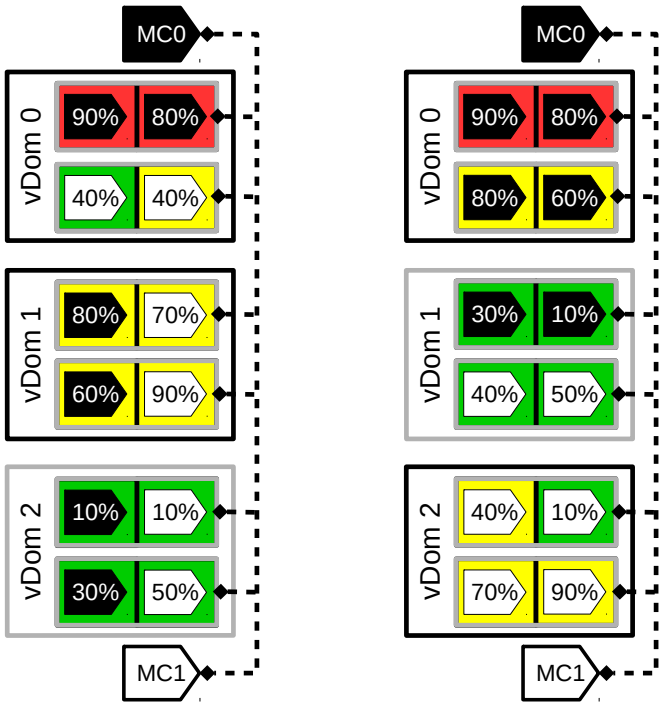
```
1:  function LevelMig(migMap, f_target)
2:      migResult ← migMap
3:      T ← GetWorkloadList(f_target)
4:      Comb ← MakeVDomComb(f_target, migMap)
5:      for each set ∈ Comb do
6:          set ← set + vDom_used
7:          tmpMap ← PlaceTarget(T, set, f_target)
8:          migMapList.add(tmpMap)
9:      end for
10:     migResult ← BestPowerState(migMapList)
11:     return migResult
12: end function
```

Table 4.1: Result of migration example

	{vDom0, vDom1}	{vDom0, vDom2}
# of migration	4	6
sum of weighted memory distance	19.8	11

Also, the number of T and $Cand_{in\_used}$ are 5 and 2. Then $N = \lceil \frac{5-2}{4} \rceil = 1$ and we can make *set* by selecting 1 voltage domain from $vDom_{left}$ and add $vDom_{used}$. The result is $Comb = \{\{vDom0, vDom1\}, \{vDom0, vDom2\}\}$.

Then, we assign T in a for each *set* in *Comb*, evaluate the assignments, and return the best migration map from the migration mapping list (line $9-11$ in Algorithm 2). The results look like Figure 4.2. We can calculate the number of migrations and sum of weighted memory distance like in Table 4.1 for each Figure (a) and Figure (b). How to place workloads will be discussed in Chapter 4.4.3. After getting the results, we evaluate each migration schedules' power status by using the evaluation model and choose the best in the *Comb*. Then, we calculate migration for next frequency level.



(a) Result for a set $\{vDom0, vDom1\}$. (b) Result for a set $\{vDom0, vDom2\}$.

Figure 4.2: Workload migration steps with f_{mid} example result.

4.4.3 Assign target

In this step, we show how assign T to *Cand* for each *set* from *Comb*. First, we collect *Cand* in a *set*(line $3-5$ in Algorithm 3). But, at this time, the *Cand* are cores which have a workload $< f_{target}$. Because, we will make workloads which are initially placed in *set* keep position (line $9-10$ in Algorithm 3). Then, we allocate for each *workload* in T, in descending order of memory intensity, on the core which has the shortest distance to the workload's memory controller in *Cand* (line 8 in Algorithm 3).

After placing *workload*, there will be workloads which are kicked out from the own core. We shall call these workloads as *victim*. Moreover, cores which T have been

Algorithm 3 Place Target Workload

cand: Candidates of migration destination

```
1:  function PlaceTarget(T, vDomSet, f_target)
2:      SortByMemoryLoad(T)
3:      for each workloadw ∈ vDomSet do
4:          if w.requestFreq < f_target then
5:              cand.add(workload)
6:          end if
7:      end for
8:      for each workloadw ∈ T do
9:          if vDomSet.contain(w.id) then
10:             continue                                  ▷ Keep position
11:         end if
12:         dest ← GetMinDistCore(cand, w.memCntrl)
13:         victim.add(dest)
14:         empty.add(w.id)
15:         migMap.update(w, dest)
16:     end for
17:     migMap ← PlaceVictim(victim, empty, migMap)
18:     return migMap
19: end function
```

initially placed will be empty. We call the empty cores as *empty*. The list of victims, *victim*, and empty cores, *empty*, is updated at line 9 and 10 at Algorithm 3. And update *workload*'s placement in migration map (line 11 in Algorithm 3).

For example, we have $Comb$=$\{\{vDom0, vDom1\}, \{vDom0, vDom2\}\}$ in Figure 4.1 (a) with target frequency f_{mid}. For a *set* $\{vDom0, vDom2\}$, the *Cand* are four cores which have f_{high} and T which will move are three cores in $vDom_1$. The workload which uses $MC0$ and the other workload which uses $MC1$ are assigned to two cores which on the bottom of $vDom_0$ and a workload on the bottom left of $vDom_2$ which places close to each memory controller. The result will be like Figure 4.1 (b). There are three empty cores in $vDom_1$ with three victim cores on the right-hand side of the figure. To handle *empty* and *victim*, we call *PlaceVictim* (line 12 in Algorithm 3).

4.4.4 Assign victim

Algorithm 4 Place Victim OS

victim: A victim OS is a OS originally placed at a target OS's detination.

```
1: function PlaceVictim(victim, empty, migMap)
2:     SortByMemoryLoad(victim)                          ▷ Decending order
3:     for os ∈ vimtim do
4:         memCntrl ← os.memCntrl
5:         dest ← GetMinDistCore(empty, memCntrl)
6:         migMap.update(os, dest)
7:     end for
8:     return migMap
9: end function
```

Because *victim* lost its core, we should allocate *victim* to *empty*. This step is same as assigning T with *victim* and *empty* correspond to T and *Cand* (see Algorithm 4). Also, because *victim* and *empty* don't have original place and allocated *workload*, it does not generate any *victim* or *empty*. The result of example case in Figure 4.1 is Figure 4.1 (c) because $MC0$ and $MC1$ are closer from the and the bottom at $vDom_1$.

4.5 Workload Migration Evaluation Model

The energy for the next time quantum t of the status quo is computed as

$$E_{status\_quo} = P_{status\_quo} \cdot t \tag{4.2}$$

where P_{status_quo} can be obtained from the on-chip sensors or, in the absence of such, from Equation 2.2. Constants are obtained offline for each frequency. The expected

energy consumption if the migration is performed is given as follows

$$E_{migrated} = P_{migrated} \cdot (t + O_{migration} + O_{memory}) \tag{4.3}$$

$$O_{migration} = t_{migration} + t_{cache\_fill}(f_{target}) \tag{4.4}$$

$$O_{memory} = t \cdot \frac{throughput_{status\_quo}}{throughput_{migrated}} \tag{4.5}$$

where $P_{migrated}$ is computed based on offline power consumption data for each frequency level. 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 throughput at each frequency and core location is profiled once offline; the actually 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

$$E_{status\_quo} > E_{migrated} \cdot (1 + \Delta m) \tag{4.6}$$

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

Chapter 5

Implementation

This chapter describes the implementation of the proposed cooperative hierarchical power management on a concrete hardware platform, the Intel Single-Chip Cloud Computer [15]. We first provide a short overview of the SCC platform and its capabilities and then describe the implementation in detail. The implemented application-specific runtime is a modified version of the sccLinux provided by Intel.

5.1 The Intel Single-chip Cloud Computer

The Intel SCC is a concept vehicle created by Intel Labs as a platform for many-core research. It consists of 48 independent cores interconnected by a routed network-on-chip (NoC). The cores are Intel P54C Pentium® cores with bigger L1 caches (16KB) and additional support for managing the on-chip scratchpad memory, the so-called *message passing buffer* (MPB). The Intel SCC provides no cache coherence for the core-local L1 and L2 caches. Always two cores are grouped together to form a *tile*; the 24 tiles are organized on a 6 by 4 grid. Four memory controllers in the four corners of

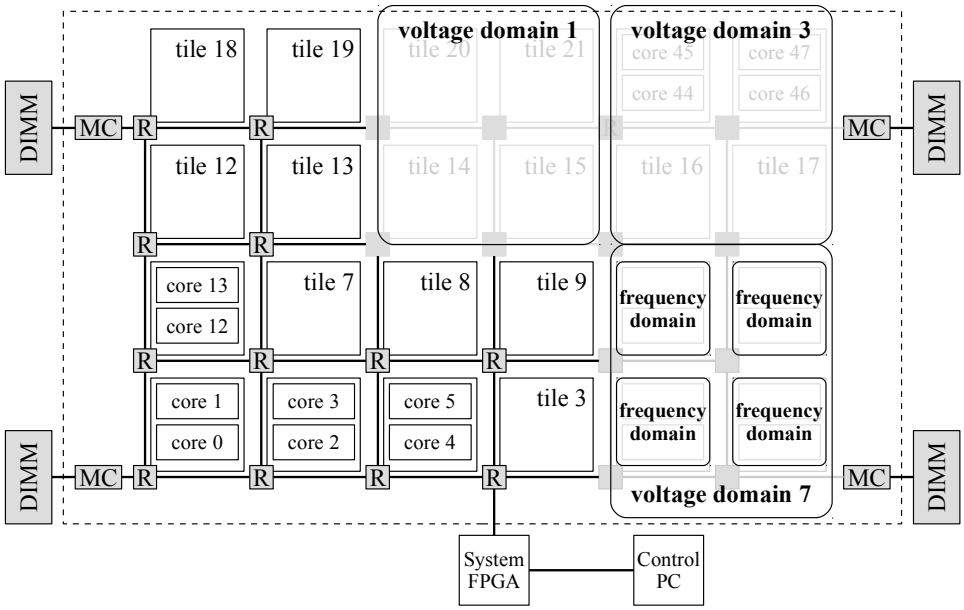


Figure 5.1: Intel SCC block diagram.

the chip provide access to up to 64 GB of memory. An FPGA provides the interface between the CMP and the management PC (MCPC). Figure 5.1 shows a block diagram of the SCC.

Memory Addressing. To support addressing up to 64GB of memory with 32-bit cores, the SCC implements the second level of indirection in the virtual-to-physical address translation. On the core, virtual-to-physical translation is performed as usual. The core-level physical addresses are then translated once again into system-level addresses through core-local lookup tables (LUT). With 64-bit cores, the LUT translation process will not be necessary anymore.

DVFS Capabilities. The SCC allows control over voltage and frequency for cores and the NoC. The frequency can be controlled *per tile*, that is, the two cores located at the same time always run at the same frequency and constitute a frequency domain (FD). The voltage can be regulated for a group of four tiles, i.e., a voltage domain

(VD) comprises a total of eight cores. The right upper hand of Figure 5.1 illustrates frequency and voltage domains on the SCC. In total, there are six voltage domains comprising four frequency domains à two cores each. The SCC supports seven different supply voltage levels. 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 between 320 and 100MHz.

Power Measurement. 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 Chapter 7.

5.2 Implementing Workload Migration

The Intel SCC provides no means to read/write core-local registers from outside a core. A minimal level of cooperation is thus required by the application-specific runtime. Here, we first describe the logical steps necessary to re-assign a core to a new application container and then discuss concrete implementation details.

5.2.1 Migration Steps

Figure 5.2 illustrates the necessary steps to carry out a migration plan computed by the chip controller (Chapter 4.3). The migration manager first signals all kernels that are about to be migrated through an interrupt. Upon receipt of a migration interrupt, the OSes first save their complete volatile state of the core to a designated area in shared memory and set a flag to indicate completion of saving the state. They then flush the TLB and the caches, and then enter a busy loop, waiting for a flag set by the global mi-

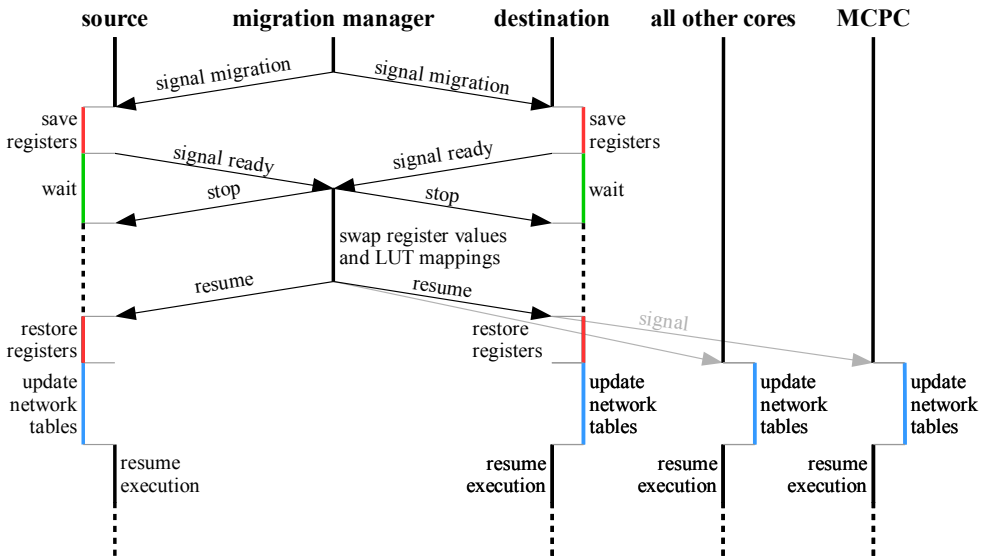


Figure 5.2: Workload migration.

gration manager to continue. Upon continuation, the volatile states are restored from the designated area, and the kernels return from the interrupt and continue execution. The migration manager waits for all kernels to save their volatile state and enter the busy loop. Before setting the completion flag, thereby allowing the cores to continue, the manager exchanges the contents of the saved volatile states of the migrated kernels with that of each target core. This means, in effect, that an entire kernel can be migrated from one core to another by copying a few hundred bytes of volatile state. This process is not much different from task switching with the difference that kernels are not scheduled in or out but rather swapped. To maintain consistent networking state, all cores, including the MCPC need to update internal network routing tables to reflect the new locations of the cores (Chapter 5.2.2).

5.2.2 Networking

The SCC provides two separate networks: one network for on-chip networking, and a subnet for communication with the MCPC. 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 transparent migrations, we have added a table holding the current IP-to-coreID mappings in each kernel. After each migration, the migration manager notifies *all* cores about the changes to the IP-to-coreID mapping tables. The same method is used on the MCPC. These simple modifications are enough to keep networking, including open connections, alive across migrations. DMA is not supported, and no other devices exist on the SCC; input/output, including access to permanent storage, are routed through the network.

5.3 Domain Controller Implementation

The domain controllers (core, frequency, voltage, and chip) are implemented in C and are present on each kernel. As outlined in Chapter 3.3.5, the physical core ID determines which controllers are (de-)activated in a kernel. The reason is workload migration. We implement workload migration as a OS migration that makes OSes float around in frequency and voltage domains. If we assign a domain controller to a certain OS, the OS might control a domain even if it is not in the domain and every domain controllers can not distinguish where the information are come from. After migration and before returning from the migration interrupt, kernels check if the core they are running on requires activation/deactivation of one of the four controllers. Core controllers are active on every kernel. The 24 frequency controllers are activated on the cores with an odd core ID. The six voltage controllers run on the lower-left core of each domain (core IDs 0, 4, 8, 24, 28, and 32). The chip controller runs on core 30.

Chapter 6

Experimental Setup

6.1 Hardware

All experiments were conducted on the Intel SCC [15]. The chip controller and other services such as monitoring logging, run on dedicated cores in voltage domain 1. The microkernels run a modified version of sccLinux that supports kernel migration and dynamic IP-to-coreID mappings. We chose this separation in order to separate the power consumption of the core OS from the application containers, voltage domain 1 does not participate in workload migration. However, the SCC only allows measuring the *total chip power*; the power consumption of the OS services are therefore also included in all results. Power consumption is computed using the on-chip voltage and ampere meters. The meters are queried 10,000 times per second. Power is computed by multiplying the measured chip supply voltage by the current. This includes the power consumed by all 48 cores and the NoC. In particular, since the power manager is implemented entirely in software and runs on the cores of the SCC, the power measurements include all the overhead caused by the propose power manager.

Table 6.1: Datacenter scenarios: distinct workloads patterns

Scenario	G1	G2-G5	G6	G7-G11
# patterns	4	7	10	40

Table 6.2: Average CPU and memory load

Average	CPU load	Mem load
G1	34	14
G2	47	14
G3	40	13
G4	37	14
G5	39	16
G6	39	17
G7	42	17
G8	37	16
G9	41	17
G10	39	14
G11	39	18

6.2 Benchmark Scenarios

A benchmark scenario is defined by a mapping of a number of workload patterns to a number of cores. Depending on the scenario, we map from 2-40 different patterns onto 8-40 cores. Cores with no assigned workload only run the modified sccLinux kernel and domain managers depending on the core location (Section 5.3).

In this evaluation, we focus on workload scenarios occurring on the servers of a datacenter. We have created three synthetic benchmark scenarios composed of synthetic workloads in order to explore the best and worst cases and show the effect of NUMA awareness for the proposed technique. The workload patterns of the datacenter scenarios are based on the Google cluster data [36]. For the average CPU usage and memory intensity, we used the information of mean CPU usage rate and the memory accesses per instruction (MAI) from the data set. We add up the number of individual processes running on the same physical machine to obtain a real-world workload of a

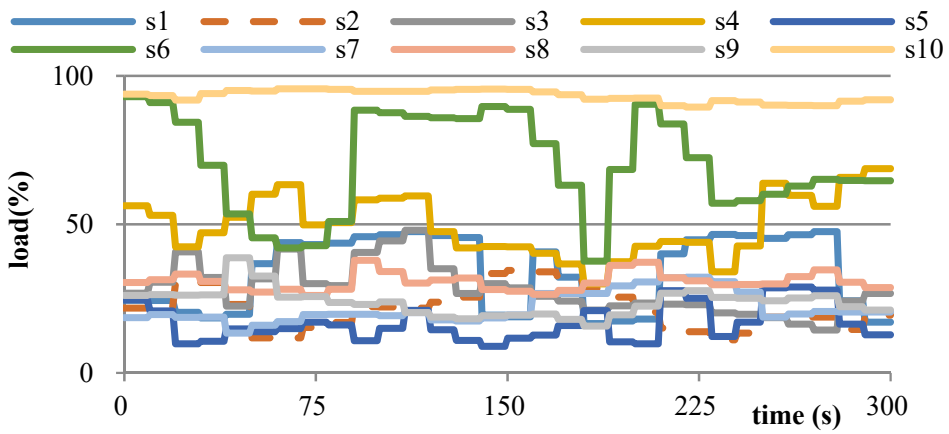


Figure 6.1: G6 workload patterns.

server over time. We scaled profiled time from average 5000 seconds to 300 seconds. We convert the numbers to a sequence of average utilization rates per 10 seconds. Then we assign the sequence as a workload pattern of a machine per a core. Each scenario has multiple workload patterns with different numbers. By this setup, we simulate not only multi-threaded applications but also multi-program environment. We have generated a total of 11 scenarios based on the Google cluster data, Table 6.1 lists the number of distinct patterns per scenario. The distinct patterns are assigned to a varying number of cores; details are given in Chapter 7. Figure 6.1 shows the 10 distinct patterns of the G6 scenario as an example. For simplicity we only display the CPU load; the memory load shows similar patterns. The average CPU and memory load of benchmark scenarios are in Table 6.2. Also, we have tested with a different number of containers which have workloads to show how it is hard to save power consumption with DVFS without workload migration according to the number of workloads increased.

6.3 Comparison of Results

The baseline of the experiments is result which is obtained by running the benchmark scenario on the SCC at full speed (800MHz) with no power management enabled. Unlike the work in [17] we do not use a phase-detector based on message passing since we are aiming at independent workloads running on a CMP. The workload of a kernel is estimated based on a weighted average of the past 10 measurements.

To show impact of workload migration and NUMA-awareness, we compare the presented NUMA-aware power management technique with the DVFS-only approach of Ioannou *et al.* [17] and the DVFS+migration technique with its locality-unaware buyer-seller algorithm described by Kang *et al.* [21]. The hierarchical framework and the DVFS policies for all three methods are identical. We evaluated the different core migration algorithms using the DVFS policies `Allhigh` and `Tile` (Section 4.1).

For all methods and benchmarks scenarios, the migration benefit threshold Δm is set to 10%. Because we want to keep overhead less than 1% of an epoch, with latencies $\leq 3ms$, $\leq 10ms$, and a few thousand cycles for migration, voltage change, and frequency changes, migrations are evaluated and performed once every 3 seconds. All benchmark scenarios are executed for 300 seconds. The reported results are the average of at least 3 runs executed at similar thermal conditions. Also, to reduce the effect of temperature, we use results which run in similar temperature for each scenario.

Chapter 7

Results

We have conducted a wide range of experiments to evaluate our proposed power management technique which is NUMA-aware and hierarchical. To show the potential of our technique, we compare it with state-of-the-art methods which are using DVFS-only [17] and NUMA-unaware workload migration [21] on synthetic benchmark scenarios. The real-world server workloads obtained from Google cluster data [36] are then used to compare the three techniques in more realistic workload scenarios. For these workloads, we conducted experiments with different number of workload patterns and different number of workloads to show the effects of the differences. At last of this chapter, we conclude this section with the overall results overall benchmark scenarios.

7.1 Synthetic Scenarios

We first compare the *DVFS only* method [17] with a data-locality-unaware *Buyer-Seller* migration algorithm [21] and our NUMA-aware *Greedy* migration technique in terms of the performance per watt (PPW) at equal turnaround time. The goal is to

Table 7.1: Normalized PPW for synthetic scenarios

BM	DVFS only		Buyer-Seller		Greedy	
	AH	T	AH	T	AH	T
SynMem	1.12	1.23	1.51	1.52	1.67	1.68
SynCpu	1.30	1.33	1.59	1.58	1.61	1.60
SynRandom	1.00	1.01	1.00	1.00	1.02	1.04

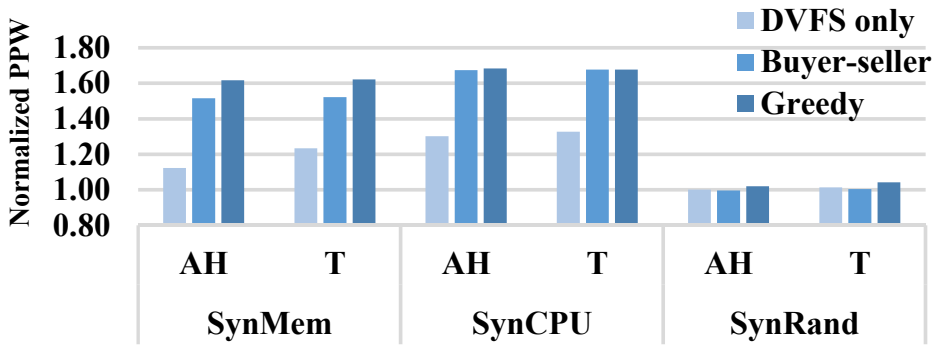
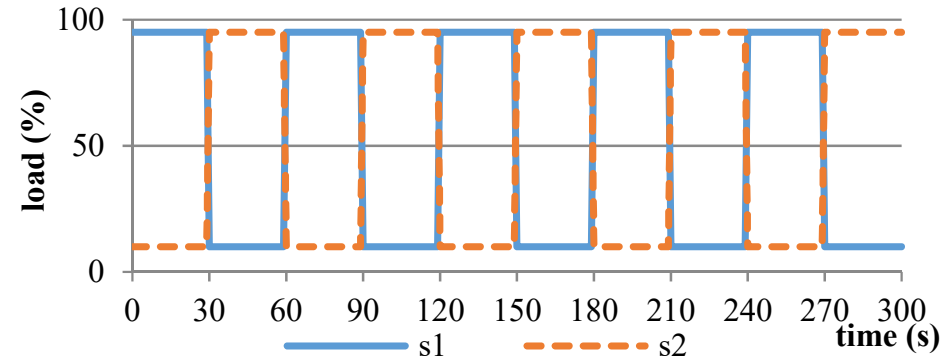
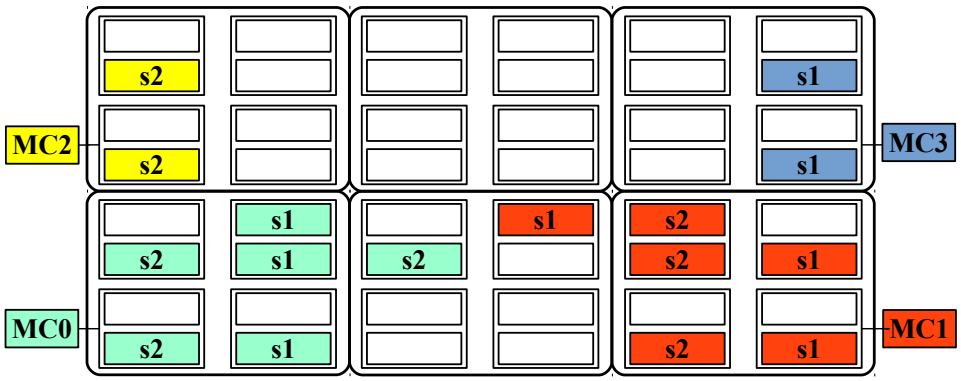


Figure 7.1: Synthetic scenario PPW.

show the necessity and the potential of NUMA-aware migration. For this, we have crafted three synthetic scenarios running synthetic workload patterns. Figure 7.2 (a) shows the workload patterns of *SynMem* and *SynCPU*. Two patterns denoted *s1* and *s2* show alternatively high and low utilizations. The patterns are crafted such that when *s1* shows a high utilization *s2* has a low load and vice-versa. The initial distribution of the workload patterns to the target architecture is shown in Figure 7.2 (b). The label refers to the workload pattern, and the coloring shows the affinity of the different workloads to the respective memory controllers. For example, *s2* running on core 0 in the left-bottom corner has its data located in memory controller *MC0*. This workload patterns and distribution make DVFS cannot lower voltage because *s1* and *s2* alternatively need high frequency. *SynCPU* and *SynMem* differ in that the former is CPU-bound (i.e., has very little memory accesses) whereas the latter is memory-bound. We expect that



(a) Load pattern



(b) Load Distribution

Figure 7.2: Synthetic scenario workload patterns.

migration outperforms DVFS on both scenarios and that especially for *SynMem* the proposed NUMA-aware algorithm achieves a better PPW. *SynRandom* is the worst-case scenario. It comprises forty distinct random workload patterns. With completely random utilizations and full occupation, migration is not expected to perform much better than DVFS only.

Table 7.1 shows the results of the synthetic scenarios for each of the three algorithms, we report the normalized PPW with respect to the baseline (no DVFS) for the *AH (Allhigh)* and *T(ile)* DVFS policy. We observe the expected behavior: for *Syn-*

CPU the migration-based algorithms outperform *DVFS only* by around 35%, and there is no significant difference between *Buyer-Seller* and *Greedy*. This result confirms the importance of workload migration on MVMF CMPs.

For *SynMem*, data locality comes into play. Even if both migration policies are effective, *Buyer-Seller* outperforms *DVFS only* by a similar margin as *SysCpu*. *Buyer-Seller* congregates high workloads to same voltage at a time, makes rest voltage domains lower power. In this case, however, the NUMA-aware *Greedy* algorithm is able to improve the PPW by 16% over *Buyer-Seller*, emphasizing the need to consider data-locality to achieve maximal power savings. This is because *Buyer-seller* algorithm does not consider memory locality.

Especially in case of *SynMem*, the result of *Buyer-seller*'s workload migration collect workload *s1* to left bottom voltage domain and *s2* to right top domain. This workload mapping makes containers apart from memory controllers, which lead to high memory access delay. On the contrary, our greedy algorithm assigns *s1* to right bottom and *s2* to middle bottom voltage domain which make lower memory access delay than the *Buyer-seller* case.

We can find a clue in the result of *SynCpu* which has same workload patterns and initial distribution with *SynMem* except it has CPU workload. *SynMem* shows the Greedy algorithm has an advantage over the *Buyer-Seller* algorithm, *SynCpu* shows similar results in both workload migration policies. In this context, we can infer the importance of memory locality.

For *SynRandom*, there is not much room for improvement for any algorithm. *DVFS only* and *Greedy* fail to improve the PPW compared to no power management, while the proposed *Greedy* algorithm improves the PPW by a few percent only. With 40 random workloads and no migration and the constraint of equal turnaround time, *DVFS only* is unable to apply DVFS.

The DVFS policies evaluated in this paper do not trade performance for power. As

a result there is no noticeable slowdown for any of the three algorithms, and the numbers have been omitted for brevity. Table 7.2 show the performance loss in numbers for the 11 datacenter scenarios.

7.2 Datacenter Scenarios

7.2.1 Varying Number of Workloads

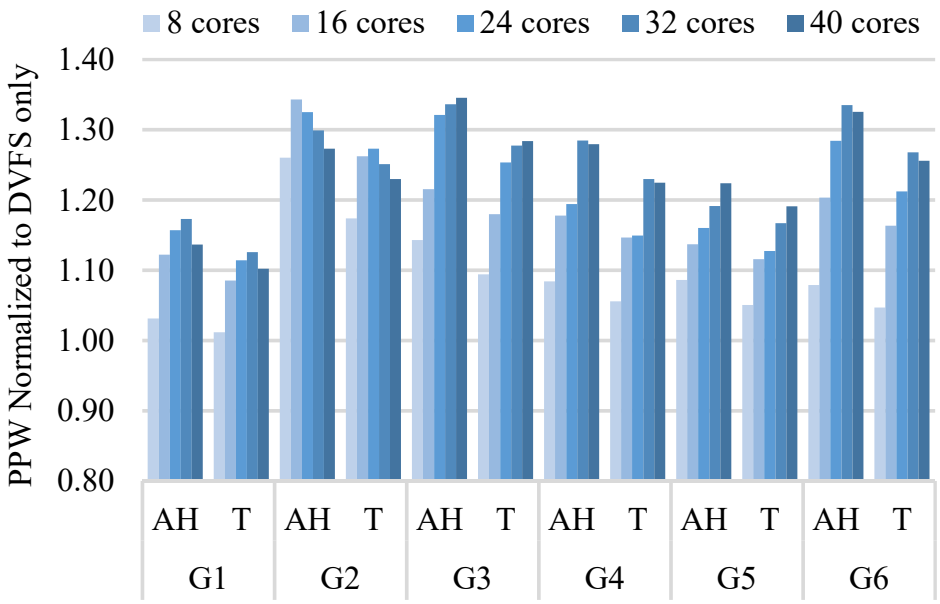


Figure 7.3: PPW for a varying number of workloads.

We first evaluate the real-world datacenter scenarios with respect to a varying number of assigned workloads from 8 to 40 in increments of 8. The number of distinct workload patterns for each scenario is given in Table 6.1; patterns are randomly assigned to the number of workloads (i.e., for the 8-workload case and *G1* we make 8 random selections from the pool containing the four workload patterns). The initial location of the workloads on the chip can affect the result; we create three different

random assignments and report the average of running each of the tree assignments three times. In other words, each individual result represents the average of nine runs.

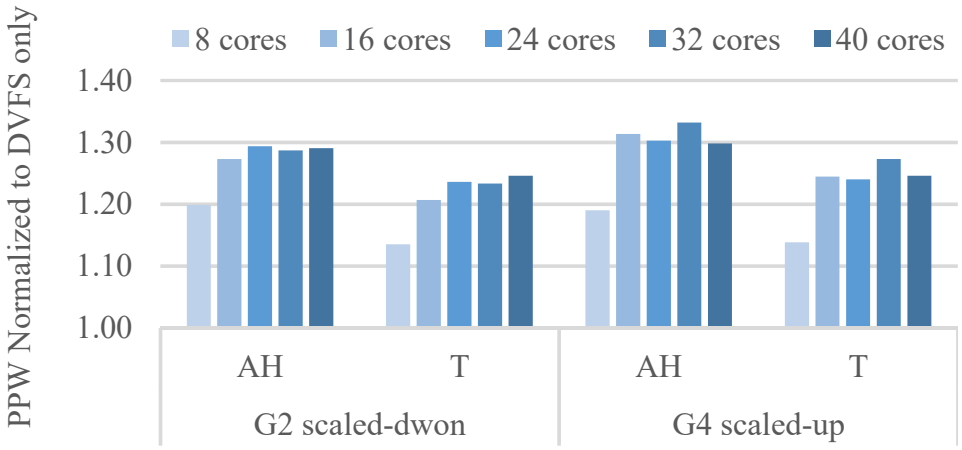


Figure 7.4: PPW for scaled scenarios with a varying number of workloads.

Figure 7.3 displays the results for the datacenter scenarios *G1* to *G6* with 8, 16, 24, 32, and 40 workloads running simultaneously. The Y-axis shows the PPW of the proposed greedy algorithm relative to *DVFS only*. We observe that *Greedy* shows better relative improvements if the number of active workloads (i.e., active cores) is 32 cores. In the case of 8 workloads, *DVFS only* manages to do quite a good job (50% over the baseline) despite its inability to migrate workloads because the low occupation still provides sufficient opportunities to apply DVFS. The reason is the number of workloads removes conflicting frequency needs. On the other end of the spectrum with 40 cores there are less opportunities for power savings with or without migration. The best case are moderately loaded CMPs where the proposed greedy algorithm outperforms *DVFS only* by 25% on average.

For *G2*, it show different tendency with increasing number of workloads. The reason why it shows different result is average utilization. The other scenarios have average utilization 35%˜41%, but *G2* has 49% average utilization. High workload av-

erage reduces space which we can save power. To prove it, we have conducted experiments with *G2* which is 25% scaled down. Figure 7.4 shows the result of scaled-down G2 which similar with the other scenarios. To show the dependency between PPW and average workload, we have conducted scenario *G4* with scaled up version. The result is showed in Figure 7.4 which decreased gap between the varying number of workloads.

Also, the workload migration shows less effective in scenario *G1*. That's because DVFS only policies also save significant power. It based on the minimum utilization (in this scenario group), and the number of workload patterns (see Table 6.1). Less number of workload patterns make a voltage domain be more likely to have similar workload patterns in it, not only for *DVFS only* but also for both migration policies. It is also shown on Table 7.2, which is the highest PPW for every policies.

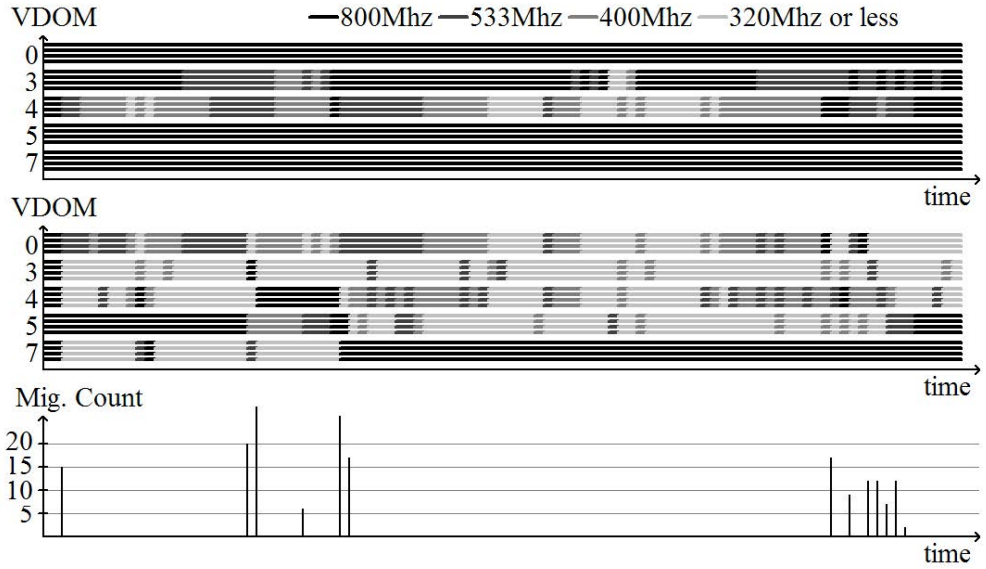


Figure 7.5: Frequency map example for *G6* and *Allhigh*.

The effect of workload migration is visualized in Figure 7.5. The topmost graph shows the frequency map of *DVFS only* with the *Allhigh* policy for the different voltage domains. Darker colors represent higher frequencies. The middle graph shows the fre-

quency map for the same workload with the proposed *Greedy* algorithm. While *DVFS only* is required to run most domains at a high frequency for most of the time, we observe that *Greedy* is able to group workloads with similar utilization into a few domains and apply aggressive DVFS on the lightly loaded domains. The bottom graph in Figure 7.5, finally, shows the number of workload migrations over time.

7.2.2 Independent Workloads

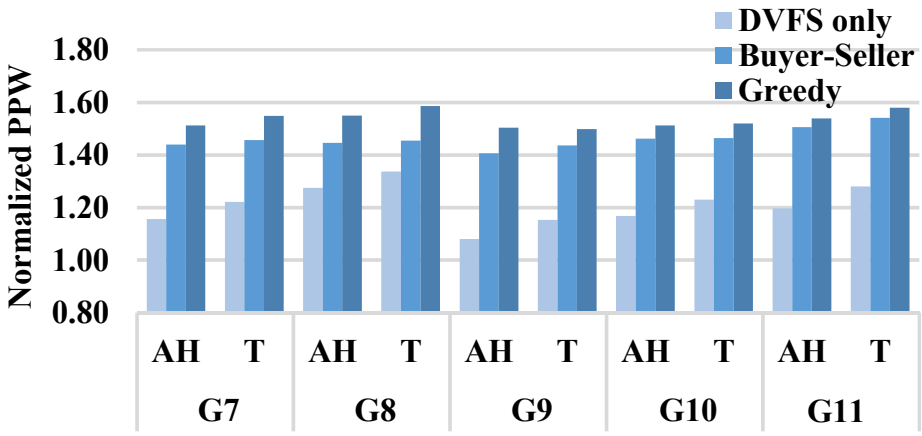


Figure 7.6: Experiment results for *G7* to *G11*.

Table 7.2: Normalized performance per watt for Google cluster data scenarios

	DVFS only				Buyer-Seller				Greedy			
	AH		T		AH		T		AH		T	
BM	PPW	Perf Loss	PPW	Perf Loss	PPW	Perf Loss	PPW	Perf Loss	PPW	Perf Loss	PPW	Perf Loss
G1	1.58	0.00%	1.61	0.00%	1.78	0.00%	1.76	0.00%	1.79	0.00%	1.77	0.01%
G2	1.00	0.00%	1.05	0.03%	1.26	0.02%	1.29	0.07%	1.28	0.02%	1.29	0.08%
G3	1.04	0.00%	1.10	0.00%	1.38	0.00%	1.39	0.04%	1.40	0.00%	1.41	0.06%
G4	1.23	0.00%	1.28	0.00%	1.55	0.00%	1.55	0.07%	1.57	0.01%	1.57	0.05%
G5	1.32	0.00%	1.37	0.00%	1.57	0.00%	1.59	0.30%	1.62	0.00%	1.64	0.04%
G6	1.16	0.00%	1.23	0.04%	1.51	0.51%	1.51	0.53%	1.54	0.20%	1.54	0.26%
G7	1.16	0.00%	1.22	0.00%	1.44	0.00%	1.46	0.00%	1.51	0.00%	1.55	0.00%
G8	1.27	0.00%	1.34	0.00%	1.45	0.00%	1.45	0.00%	1.55	0.00%	1.59	0.03%
G9	1.08	0.00%	1.15	0.00%	1.41	0.00%	1.44	0.01%	1.50	0.00%	1.50	0.04%
G10	1.17	0.00%	1.23	0.00%	1.46	0.00%	1.46	0.00%	1.51	0.00%	1.52	0.05%
G11	1.20	0.00%	1.28	0.00%	1.51	0.00%	1.54	0.00%	1.54	0.00%	1.58	0.05%
AVG	1.20	0.00%	1.26	0.01%	1.48	0.05%	1.49	0.09%	1.53	0.02%	1.54	0.06%

Figure 7.6 shows the normalized performance-per-watt over the baseline for datacenter scenarios *G7-G11*. Each scenario is composed of 40 independent server workloads as recorded in Google's datacenters which can be more realistic in real world. Overall, we observe that the proposed *Greedy* outperforms *DVFS only* by a large margin, once again emphasizing the importance of workload migration. Compared to *Buyer-Seller*, the NUMA-awareness of *Greedy* pays off in a 8% better energy efficiency. The reason for the larger gap is because as the number of workload patterns increase, the probability which a voltage domain get high frequency request increases. Also, more significant number of patterns needs more frequent workload migration. The *Buyer-Seller* algorithm which doesn't consider memory locality keeps cause high memory access delay in contrast with the greedy algorithm which make cores need higher frequency. As a result, in scenario *G7* to *G11*, performance per watt gap between the greedy and the Buyer-seller are increased as 7% for *AH* and 8% for *T* from 2% in case of *G1* to *G6*.

7.3 Overall Results Comparison

Table 7.1 and Table 7.2, finally, displays the performance per watt (PPW) and the performance loss over the baseline policy, respectively, for the *Allhigh* and the *Tile* policy for *DVFS only*, *Buyer-Seller*, and the proposed *Greedy* algorithm. Each scenario is run with the number of workloads listed in Table 6.1. Every result is the average of evaluated at least three times. For the real world workloads, we have also tested three random initial placements of workload patterns.

From synthetic workloads, Table 7.1, *SynMem* and *SynCpu* are periodic workloads scenarios. For *SynMem*, we use long period length and large memory workloads. It shows large gap not only between DVFS-only policies and DVFS with migration policies but also *Buyer-Seller* and *Greedy* algorithm which we introduce. For *SynCpu*,

we use long period length and large CPU workloads. It shows, also, the large gap between *DVFS-only* and *DVFS with workload migration* policies, but there is little gap between *Buyer-Seller* and *Greedy*. The *SynRandom* is a scenario consist of randomly generated workload patterns for each 40 application containers. The workload patterns change every 3 seconds. For this workload scenario, we get minimum improvement at performance per watt, 2%-4%.

In case of the real world workloads, overall, the NUMA-aware *Greedy* algorithm outperforms *DVFS only* by about 30% and *Buyer-Seller* by 5%. We observe that *Tile* outperforms *Allhigh* without migration whereas with migration they achieve similar performance. The reason is that OS migration can group OSes with similar performance requirements into voltage domains such that the superior *Tile* DVFS policy has less effect. The performance degradation is negligible for all three policies. The algorithms that support migration suffer from a slightly higher performance degradation (0.05% on average over no degradation with *DVFS only*), but the slowdown is insignificant compared to the 30% improved energy efficiency.

Chapter 8

Discussion

In this section, we discuss about limitations and extra hardware support to get more improvements of this work.

8.1 Limitations

The first topic is the limitations of this work. One of the limitations of this work is on an assumption. We assume each core has single workload on it which is not true on real system. It make scheduling workload migration more difficult. Because, if there is multiple processes which uses different memory controller with each other in a core. Then, workload migration scheduling would be more complicate.

Anther limitation of this work is scalability of algorithm. This algorithm runs at M number of voltage levels traverses combination of voltage domains which can be represented as

$$O(M * N^{min\{K,N-K\}}) \tag{8.1}$$

where N and K are the number of voltage domain which we can select and we need to

select. In Intel Single Chip Cloud Computer (SCC), our greedy algorithm take around 0.002 second, because, we have only four voltage levels and five voltage domains. But, if a system which have more fine granularity of voltage domain, the computation time would take longer.

The point we doesn't considered is temperature. In our framework, we collecting high workloads into same voltage domains. It could increase heat of the voltage domains which could make problems. This is a problem one of problems considered in future work.

8.2 Extra Hardware Support

In the system we use, Intel SCC, there is some hardware characteristics which make bottleneck. In this section, we discuss extra hardware support which can solve this problems and achieve more improvements from this work.

In the Intel SCC, dynamic voltage and frequency scaling (DVFS) too have to use more frequently ($\leq 10ms$). If we can have more light weight DVFS (less delay), we can response more quickly the cores' request changes.

Also, we implemented OS migration for workload migration. OS migration make more overhead than process migration because of swapping LUT and updating network tables. If we work with process migration, we can have less overhead on workload migration and better results.

Chapter 9

Conclusion

We have presented a NUMA-aware cooperative hierarchical power management technique for existing and future many-core systems. The technique employs workload migration. Without explicit hardware support, the microkernels running 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 this technique allows the power manager to group cores with similar performance requirement together so that traditional DVFS policies can apply DVF settings closer to the optimal value. In order to remain scalable, the power manager is implemented in a hierarchical fashion, logically re-creating the hierarchy imposed by the hardware through the different power management domains.

The cooperative power manager has been implemented and evaluated on a real system, the Intel Single-Chip Cloud Computer. Experiments with a wide range of real world workload benchmark scenarios show that, on average, the proposed technique outperforms existing DVFS policies by 30% and by 5% compared to a NUMA-unaware approach at the expense of little performance loss, less than 1%.

Appendix A

Benchmark Scenario Details

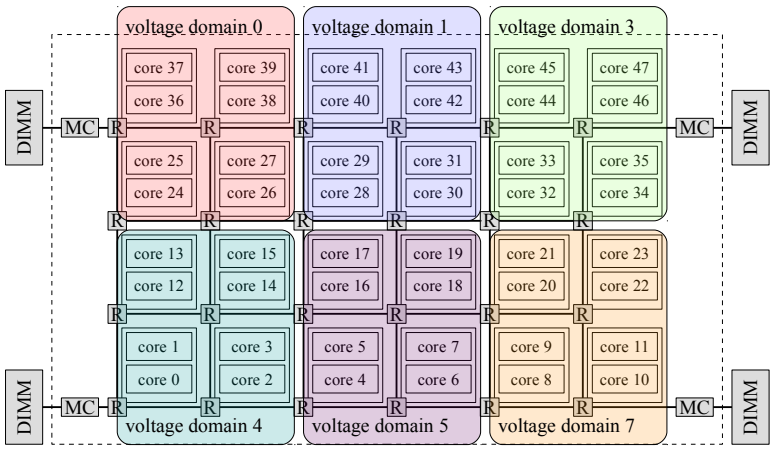


Figure A.1: SCC core map

In this appendix, we describe details of the benchmarks used for experiments. We have total 14 benchmark scenario, 3 for synthetic and 11 for real-world benchmarks. Each benchmark has workload patterns and distribution of workloads on cores. We

describe *synthetic benchmarks* and *real-world benchmarks* in Section A.1 and Section A.2.

Each workload pattern has CPU load and memory load. We described these in the tables below. The numbers represent ratio to maximum performance of a core at the highest frequency and the closest memory distance.

Benchmark has multiple workload patterns but the number of patterns is different. For benchmarks which have 40 workload patterns, each pattern initially placed at a core which has the same number which represented in Figure A.1 (a block diagram of Intel SCC). On the other hand, for benchmarks which have workload patterns less than 40, multiple cores can have same workload pattern. So, we describe initial workloads distribution in the tables followed by workload pattern table of each benchmark.

In our setup, voltage domain 1 is a control domain. So, it runs control processes and various measurement services. That's why cores in the domain don't get any workloads.

A.1 Synthetic Benchmark

Table A.1: *SynMem* benchmark scenario

(a) Workload pattern

WL		Epoch (1 epoch = 15 sec)																			
		00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19
S1	CPU	0	10	10	0	0	10	10	0	0	10	10	0	0	10	10	0	0	10	10	0
	Mem	95	0	0	95	95	0	0	95	95	0	0	95	95	0	0	95	95	0	0	95
S2	CPU	10	0	0	10	10	0	0	10	10	0	0	10	10	0	0	10	10	0	0	10
	Mem	0	95	95	0	0	95	95	0	0	95	95	0	0	95	95	0	0	95	95	0

(b) Workload distribution

vDom0		vDom1		vDom3		vDom4		vDom5		vDom7	
n/a	n/a	n/a	n/a	n/a	s1	n/a	s1	n/a	s1	s2	n/a
s2	n/a	n/a	n/a	n/a	n/a	s2	s1	s2	n/a	s2	s1
n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
s2	n/a	n/a	n/a	n/a	s1	s2	s1	n/a	n/a	s2	s1

Table A.2: *SynCPU* benchmark scenario

(a) Workload pattern

WL		Epoch (1 epoch = 15 sec)																			
		00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19
S1	CPU	95	95	10	10	95	95	10	10	95	95	10	10	95	95	10	10	95	95	10	10
	Mem	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
S2	CPU	10	95	95	10	10	95	95	10	10	95	95	10	10	95	95	10	10	95	95	10
	Mem	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

(b) Workload distribution

vDom0		vDom1		vDom3		vDom4		vDom5		vDom7	
n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
s2	n/a	n/a	n/a	s2	n/a	s2	s2	s2	n/a	s2	n/a
n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
s1	s2	n/a	n/a	s1	s1	s1	s1	s1	s2	s1	s1

Table A.3: *SynRand* benchmark scenario first half

Workload		Epoch (1 epoch = 3 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	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49
S00	CPU	10	55	81	26	93	50	77	20	20	29	80	29	85	62	10	91	1	24	58	6	83	78	37	32	60	22	76	65	35	19	97	16	22	11	40	38	33	2	60	31	97	33	61	29	22	41	2	96	78	33
	Mem	74	12	57	76	28	57	13	1	9	42	87	94	66	31	44	19	50	98	16	70	5	98	56	17	40	12	91	99	9	94	70	38	40	65	89	43	70	15	32	20	42	8	13	5	95	42	51	76	70	14
S01	CPU	26	40	11	10	82	79	41	26	90	48	99	67	59	8	92	30	91	94	16	29	92	75	1	4	20	0	40	65	39	94	66	13	58	76	17	11	28	25	85	21	64	53	2	1	49	67	57	49	44	13
	Mem	28	88	57	39	12	91	96	54	79	39	78	78	62	78	43	0	38	93	91	65	10	41	39	31	75	99	57	79	60	33	17	98	48	64	33	60	80	68	80	76	53	70	53	92	79	33	96	92	54	33
S02	CPU	77	54	1	63	63	19	23	71	35	43	12	3	63	17	64	47	22	27	40	22	12	75	72	57	86	20	80	53	75	58	94	45	91	71	18	38	99	87	35	75	99	21	27	60	90	71	32	69	5	93
	Mem	1	33	97	54	31	94	26	85	0	24	78	8	48	14	41	68	51	91	87	52	53	91	45	98	73	55	21	70	90	59	48	90	88	17	82	69	76	95	90	59	23	82	23	80	60	67	84	99	58	33
S03	CPU	95	82	60	43	40	92	29	86	77	17	80	17	28	59	70	77	46	17	94	94	18	52	75	90	54	95	63	41	75	45	59	65	16	24	46	27	32	32	29	4	69	78	24	43	49	61	96	15	1	38
	Mem	48	80	16	79	4	53	7	8	54	62	83	36	44	45	56	97	34	1	57	98	0	11	68	66	25	49	27	11	17	72	81	44	99	47	21	74	87	93	30	2	96	53	39	59	69	77	44	40	64	32
S04	CPU	0	80	74	53	32	21	76	77	54	77	31	41	26	22	57	58	64	90	18	68	35	37	75	41	0	39	32	93	86	87	29	15	99	79	99	67	10	28	53	13	10	93	93	70	89	63	16	38	15	50
	Mem	63	0	57	80	95	28	40	69	88	90	4	44	99	74	1	58	11	30	49	12	60	82	8	23	66	30	24	36	20	61	21	7	13	14	33	69	74	16	53	4	90	51	55	59	52	55	81	88	24	12
S05	CPU	85	92	80	38	9	53	21	72	64	45	16	20	10	30	70	9	75	50	27	77	59	41	84	41	99	29	84	16	50	74	71	54	4	62	45	68	71	8	54	47	68	31	61	1	67	61	69	90	43	89
	Mem	50	56	36	27	71	26	65	30	63	67	59	27	88	88	63	52	13	62	20	46	97	98	22	55	85	31	46	85	24	92	65	53	73	61	61	54	41	10	8	0	33	34	63	35	24	54	46	77	15	31
S06	CPU	36	96	39	89	58	98	46	35	49	41	52	16	71	67	28	37	10	19	70	50	13	43	70	36	56	23	99	66	16	80	36	42	35	61	11	38	56	26	31	72	45	4	96	44	21	15	13	80	8	98
	Mem	33	91	46	85	14	77	99	76	0	13	14	6	80	55	41	90	75	17	56	51	95	17	16	49	51	59	16	27	55	44	62	83	0	66	17	31	16	60	8	85	98	26	91	89	82	91	56	22	66	61
S07	CPU	23	29	82	17	43	18	12	87	14	31	44	19	99	4	53	51	93	33	97	12	6	12	95	94	48	49	28	12	99	47	44	27	98	92	60	0	86	47	80	86	48	26	82	88	13	2	11	92	67	88
	Mem	52	5	86	53	50	48	35	39	28	44	93	84	77	20	12	19	31	54	15	41	74	49	22	4	96	39	82	36	76	25	90	0	38	80	24	27	80	16	57	9	89	77	99	80	79	12	28	18	84	51
S08	CPU	77	43	13	15	29	98	98	49	82	34	85	66	22	81	5	25	45	70	70	92	15	78	47	40	30	63	49	31	63	51	14	38	76	28	24	95	15	44	15	12	52	9	88	74	72	31	16	33	12	80
	Mem	69	44	60	74	67	60	19	12	44	94	81	43	17	90	78	61	4	97	20	83	83	58	45	56	97	92	34	59	47	38	48	73	36	68	61	59	50	68	99	33	57	64	31	24	94	45	13	64	24	28
S09	CPU	85	34	13	9	82	29	31	6	61	67	17	6	99	49	4	19	54	42	30	78	8	96	46	10	94	26	73	69	49	81	4	85	5	38	39	5	34	63	30	33	60	14	67	6	20	51	81	13	24	48
	Mem	98	99	32	49	72	5	30	65	52	55	75	11	19	47	89	89	92	39	4	63	39	12	67	20	59	99	53	77	90	47	32	95	28	25	81	99	15	17	96	11	33	65	64	19	58	52	70	65	48	43
S10	CPU	64	52	21	99	0	49	48	20	16	78	96	94	17	24	40	37	6	55	89	62	13	91	5	47	24	22	66	56	24	22	0	51	52	3	96	78	19	94	26	6	49	46	66	83	90	24	95	91	62	68
	Mem	98	46	96	89	80	17	1	25	87	68	70	82	86	26	25	45	81	95	14	1	64	64	34	80	65	25	11	87	35	19	62	59	39	96	42	93	84	66	23	99	15	28	75	26	78	71	32	86	98	89
S11	CPU	79	21	30	53	85	99	81	72	61	89	15	76	54	73	50	24	49	95	64	69	79	96	36	85	53	19	4	16	33	13	90	7	64	21	8	37	0	96	52	31	8	54	29	36	56	39	41	62	96	65
	Mem	41	84	83	51	14	75	56	75	58	17	83	80	76	74	50	51	86	50	64	2	33	62	30	58	26	26	11	17	89	31	74	65	34	19	10	2	91	66	24	89	66	15	92	32	32	20	31	41	8	30
S12	CPU	80	90	3	31	6	36	23	73	89	42	73	99	6	9	38	53	47	72	92	24	98	10	89	83	29	33	88	22	61	25	42	34	28	73	20	58	47	64	72	84	45	68	82	54	91	9	59	28	75	79
	Mem	42	90	33	15	93	95	37	74	16	33	54	0	3	69	83	37	87	93	40	48	31	96	25	60	59	47	72	65	81	56	48	15	64	6	1	55	57	65	33	73	33	94	46	30	45	38	10	58	49	57
S13	CPU	97	18	40	69	0	11	40	74	85	88	83	93	5	94	95	85	83	7	80	69	91	0	86	69	10	57	81	76	23	2	93	38	27	28	14	65	87	12	38	83	77	66	90	75	34	99	77	96	79	25
	Mem	94	34	90	54	40	68	22	77	21	60	57	8	3	69	24	54	28	72	15	58	14	89	96	16	81	14	21	79	24	99	29	89	10	22	4	49	23	79	40	76	28	49	50	55	70	36	9	55	22	28
S14	CPU	3	24	34	48	74	29	13	74	79	85	14	8	31	54	17	33	1	88	7	0	39	43	42	7	20	90	27	64	69	20	19	87	20	58	41	80	98	67	57	88	85	48	26	41	54	1	43	99	16	31
	Mem	35	63	66	18	74	61	46	95	43	99	34	49	39	70	87	69	90	68	42	18	55	24	5	37	68	41	50	7	8	3	65	65	38	85	42	32	24	15	19	28	99	27	77	2	85	87	19	36	70	45
S15	CPU	99	73	96	67	58	1	15	6	7	8	67	66	88	41	42	35	2	73	15	82	10	76	47	64	86	81	65	55	58	17	4	25	72	25	10	63	68	6	21	43	54	67	7	89	48	90	26	19	75	12
	Mem	44	25	14	63	68	78	63	48	28	45	67	97	20	19	76	96	0	72	90	59	53	89	23	5	97	4	45	69	67	9	46	86	60	49	74	82	22	66	36	66	41	66	84	70	44	60	45	76	54	40
S16	CPU	7	21	17	90	42	86	43	54	48	73	89	23	36	31	56	56	21	53	66	9	31	54	41	59	39	64	17	57	80	28	32	55	85	49	48	44	43	61	33	27	48	5	17	0	85	72	5	77	91	66
	Mem	9	46	77	17	45	14	17	46	47	27	22	72	59	75	71	74	7	93	88	30	85	97	30	9	72	2	86	1	3	99	14	93	23	22	21	35	30	37	96	1	39	43	46	91	26	87	12	68	34	32
S17	CPU	5	56	1	14	92	36	92	40	47	95	28	95	91	1	1	11	95	62	97	50	22	97	78	77	99	95	26	30	60	51	45	63	32	45	72	4	18	86	74	5	68	73	23	47	12	51	27	73	6	53
	Mem	91	38	99	90	24	8	23	60	87	4	73	18	20	76	13	49	26	74	11	50	90	56	43	68	85	32	40	54	80	99	31	76	60	6	83	65	61	73	98	97	70	80	62	58	53	28	34	0	71	69
S18	CPU	52	8	46	46	47	67	99	31	83	2	20	65	19	90	74	19	66	1	11	73	45	70	38	48	26	58	75	50	66	44	80	20	96	77	52	16	21	85	34	42	17	69	25	69	64	78	2	90	99	9
	Mem	76	99	64	80	11	84	9	8	41	94	17	60	61	63	29	66	90	55	84	8	52	43	30	29	48	72	79	13	72	90	51	5	94	78	56	2	34	78	5	61	7	97	89	71	79	16	57	19	56	87
S19	CPU	20	7	77	16	36	26	59	10	41	75	99	38	26	49	11	14	62	11	85	24	57	40	39	40	75	65	39	9	99	19	4	2	11	72	50	51	99	26	38	33	40	30	61	93	26	81	52	18	26	27
	Mem	56	1	50	59	98	60	42	57	62	69	58	83	71	8	91	48	67	64	4	9	12	61	15	41	94	92	39	59	89	16	32	79	84	11	81	45	65	36	55	27	91	41	21	13	8	76	59	86	83	21
S20	CPU	62	74	3	81	78	76	79	94	21	50	95	12	43	24	87	26	64	19	79	27	65	72	23	93	78	95	26	77	35	70	39	33	72	41	14	52	36	48	96	42	91	0	55	65	25	20	64	15	63	88
	Mem	75	36	99	63	13	75	28	37	54	32	42	7	74	40	26	45	96	22	34	96	44	89	35	75	8	57	78	85	49	94	85	99	93	6	86	64	57	21	13	36	68	63	98	13	96	67	97	46	28	83
S21	CPU	37	47	59	6	46	58	48	84	28	99	3	34	31	57	26	40	26	25	95	95	92	76	55	99	38	84	66	10	33	24	62	45	62	0	22	37	19	88	67	81	60	18	43	59	29	98	99	15	8	72
	Mem	61	52	64	49	19	34	73	0	89	4	59	60	45	81	42	59	48	0	3	29	2	39	45	56	65	99	95	25	23	68	6	9	46	58	24	1	3	55	96	86	56	84	58	92	33	20	87	67	72	81
S22	CPU	0	71	87	67	13	34	15	73	20	59	76	34	70	61	19	14	79	18	33	89	50	88	90	91	3	97	53	62	38	50	50	0	24	74	38	47	71	35	85	27	91	90	0	53	72	69	71	3	77	42
	Mem	2	75	43	57	44	87	99	48	20	73	29	59	61	70	79	13	67	71	78	2	90	11	11	55	99	70	69	38	61	30	26	3	47	40	35	3	4	23	18	16	80	23	11	3	21	77	13	22	14	97
S23	CPU	93	9	29	99	17	35	49	91	33	5	69	88	9	29	61	11	94	15	30	11	57	96	90	61	22	69	26	57	20	32	28	53	40	60	20	20	21	39	45	55	32	53	50	25	49	21	83	16	58	38
	Mem	2	15	63	14	38	74	93	76	92	12	53	81	1	0	28	88	60	51	26	48	91	73	88	68	33	65	32	33	81	51	89	58	68	76	18	41	60	35	45	10	41	12	63	47	84	62	91	17	26	3
S24	CPU	50	71	89	83	64	26	92	3	3	0	84	22	70	66	21	13	20	5	78	98	23	50	16	29	68	72	85	95	40	18	26	92	17	21	30	9	35	68	10	56	66	80	38	44	55	63	75	72	77	81
	Mem	52	74	82	27	27	52	31	75	92	3	77	17	54	54	31	48	1	3	84	59	30	29	92	34	28	21	16	55	30	19	24	6	69	82	34	18	35	68	63	3	8	59	10	45	69	57	16	88	77	97
S25	CPU	82	77	14	68	65	93	86	52	43	19	67	66	2	18	68	20	13	79	76	1	55	48	99	31	20	54	3	79	27	90	38	39	57	39	96	85	19	55	3	33	98	10	3	80	45	66	87	58	89	92
	Mem	35	57	55	93	94	10	59	13	51	99	61	6	84	83	69	4	71	42	76	17	80	1	28	1	7	52	16	12	14	6	53	2	63	98	48	62	61	86	82	55	52	50	65	96	9	73	51	55	99	79
S26	CPU	3	58	52	31	36	47	16	90	86	41	94	55	11	6	59	36	94	12	62	51	33	43	57	86	95	86	17	90	54	31	75	79	66	62	38	6	4	75	13	11	48	31	40	75	28	3	22	92	91	54
	Mem	32	18	28	71	9	4	79	59	61	2	45	63	13	45	90	24	31	23	93	50	84	39	78	95	15	28	51	17	62	9	67	99	22	30	14	33	88	13	12	5	0	36	99	38	66	42	37	59	67	97
S27	CPU	26	47	52	89	82	64	1	94	58	77	68	23	13	22	81	14	90	25	45	69	64	30	14	63	89	49	87	73	66	29	13	0	60	70	64	51	19	74	87	68	55	13	28	43	36	27	5	80	60	15
	Mem	41	76	6	46	77	98	36	66	40	91	63	10	99	48	53	99	26	47	3	15	6	74	34	83	68	36	31	93	10	39	60	47	97	19	92	99	98	20	72	54	15	18	4	17	38	91	85	43	99	82
S32	CPU	45	33	90	50	63	2	31	21	8	63	98	80	84	91	5	33	73	63	94	1	5	64	22	21	50	7	2	58	62	68	94	95	54	37	15	3	71	33	58	99	51	15	18	14	28	52	75	95	54	44
	Mem	79	39	96	53	54	53	39	96	20	92	76	66	88	78	40	98	98	66	5	5	28	85	71	2	63	44	97	83	94	21	9	25	38	97	91	25	18	24	6	70	51	32	32	27	42	72	1	89	66	30
S33	CPU	68	24	45	68	71	40	25	75	11	62	84	11	73	62	36	88	2	27	44	81	69	69	49	27	24	57	60	72	39	42	78	3	89	9	73	90	19	96	45	19	64	98	41	42	10	13	87	18	46	31
	Mem	59	91	9	52	54	18	72	86	49	84	95	52	2	12	27	46	87	98	56	18	22	68	47	10	57	39	70	67	4	9	93	23	51	66	90	29	89	57	11	9	93	88	10	95	11	2	42	81	80	24
S34	CPU	27	25	20	37	99	23	92	83	31	99	60	41	30	97	80	49	62	49	1	60	20	34	40	87	59	9	28	88	90	31	28	89	58	99	22	40	43	21	62	61	29	93	79	3	31	64	51	85	97	83
	Mem	77	10	70	12	64	29	57	23	88	54	79	23	7	32	71	99	13	10	21	47	41	31	78	26	86	10	89	23	25	6	73	76	27	44	94	21	24	70	31	48	6	18	48	85	15	30	71	55	89	20
S35	CPU	94	93	13	10	7	13	71	22	24	99	74	16	33	1	3	67	78	4	52	99	33	81	21	79	84	95	62	88	55	3	86	42	52	52	6	11	51	91	35	56	89	61	34	71	38	24	97	56	93	97
	Mem	93	96	93	5	82	78	2	70	18	2	34	89	73	1	92	20	15	83	23	82	89	28	97	40	16	63	79	59	41	57	12	4	71	60	9	20	40	90	61	44	52	37	95	56	96	83	99	65	9	89
S36	CPU	34	79	88	53	2	57	90	21	68	32	0	90	74	2	65	36	29	48	41	46	39	62	8	99	49	23	88	29	33	31	13	70	15	17	60	17	84	7	34	81	90	68	87	44	80	26	15	31	63	33
	Mem	54	31	78	15	11	31	91	26	31	97	76	79	43	55	0	35	57	43	67	25	67	19	73	77	36	35	53	80	45	21	71	88	6	60	19	27	89	43	26	44	32	83	39	68	99	21	97	68	18	93
S37	CPU	83	68	84	57	3	30	65	37	17	67	41	76	53	6	48	85	97	51	10	71	63	73	1	78	10	35	28	7	88	31	68	65	10	8	60	78	20	20	78	41	86	99	26	68	62	22	64	23	53	52
	Mem	44	62	85	31	26	59	87	91	26	2	34	7	74	20	73	41	70	86	48	62	40	72	84	85	89	21	77	22	80	33	31	72	53	24	32	75	87	51	32	70	34	83	32	22	63	82	24	8	54	3
S38	CPU	40	48	79	82	62	42	13	99	14	90	91	38	20	97	74	58	66	41	42	76	44	30	79	65	92	83	40	62	31	58	48	68	43	37	55	66	58	56	4	6	28	20	69	0	98	14	37	89	4	37
	Mem	2	41	69	43	88	73	3	66	31	37	11	12	80	49	12	45	6	50	87	35	11	45	90	26	84	69	42	38	13	60	42	81	56	67	79	52	65	89	17	51	89	61	69	93	58	54	49	73	58	22
S39	CPU	7	62	79	53	98	81	23	44	92	42	17	75	21	25	26	47	96	26	3	55	91	77	34	69	85	60	49	60	84	24	99	19	79	48	34	23	13	91	96	56	87	54	90	80	47	67	54	37	86	84
	Mem	85	82	9	99	51	50	14	0	2	54	53	98	90	8	71	81	47	97	75	28	8	73	67	11	17	42	49	97	9	3	13	96	74	25	93	19	60	67	74	63	94	15	96	5	1	29	99	48	8	99
S44	CPU	12	47	70	41	74	98	47	96	96	71	19	75	92	99	0	51	43	50	14	77	58	85	66	80	1	57	75	43	75	96	80	11	44	82	94	12	52	89	68	13	21	43	14	70	3	97	79	97	98	58
	Mem	17	93	39	0	42	31	63	22	53	95	88	15	44	66	38	57	0	54	62	56	62	13	13	67	7	72	79	36	37	27	62	47	19	3	46	73	67	48	53	23	8	90	4	80	59	86	83	20	69	80
S45	CPU	57	73	4	99	42	6	59	1	91	11	84	94	13	64	82	85	0	54	21	62	89	84	26	8	11	8	60	72	39	32	0	68	79	87	71	1	1	35	13	49	89	87	7	3	78	17	3	71	93	9
	Mem	94	9	68	87	64	10	75	76	66	74	36	43	65	39	41	93	96	70	2	98	89	89	35	99	77	54	90	78	78	76	10	48	51	55	11	40	3	38	80	30	23	69	91	0	36	84	64	49	84	70
S46	CPU	13	84	7	7	13	24	37	22	11	75	38	4	77	59	83	11	39	30	89	26	62	48	74	38	50	20	44	6	18	46	33	92	84	30	91	76	60	66	80	87	20	95	86	79	35	95	16	28	59	53
	Mem	21	10	95	75	35	61	29	98	65	81	69	3	22	37	20	87	71	8	81	81	42	37	8	94	66	56	83	61	12	69	44	36	22	85	92	66	33	98	92	3	27	0	94	72	76	34	87	47	41	9
S47	CPU	70	19	6	23	67	56	55	83	8	70	79	71	13	86	28	27	65	99	6	99	55	69	26	47	92	68	35	53	30	38	11	42	73	6	43	1	6	25	46	2	54	55	99	46	79	41	74	77	36	6
	Mem	44	19	75	93	29	66	37	57	94	31	99	57	63	26	16	93	58	58	18	96	38	37	95	72	49	4	60	35	17	32	84	48	82	19	20	20	84	53	6	45	93	31	12	47	72	53	83	30	98	60

Table A.4: *SynRand* benchmark scenario second half

Workload		Epoch (1 epoch = 3 sec)																																																	
		50	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99
S00	CPU	46	44	92	5	83	55	16	39	44	3	41	81	77	33	48	99	22	75	12	62	88	30	72	88	32	87	62	13	53	66	96	61	89	65	46	57	76	17	24	49	34	98	19	8	20	72	70	48	15	43
	Mem	73	30	8	66	77	33	5	46	10	93	76	5	1	44	74	71	5	98	17	52	59	8	93	45	41	21	56	29	52	77	58	77	55	61	50	58	84	34	72	6	12	0	94	72	88	48	49	31	84	72
S01	CPU	7	45	57	64	67	11	61	92	26	86	67	7	53	53	64	77	84	61	32	62	44	8	55	82	10	62	36	89	55	0	2	94	75	11	53	42	71	75	14	4	24	6	27	98	22	37	39	8	14	59
	Mem	74	94	44	24	24	45	44	34	82	24	24	97	30	88	17	14	96	93	49	53	19	50	71	1	86	29	23	94	6	1	5	99	11	76	91	65	4	78	2	9	90	69	71	20	40	1	45	29	21	87
S02	CPU	10	63	99	20	51	88	22	59	64	49	32	57	5	95	93	42	26	65	14	44	39	88	26	26	2	30	26	85	40	99	68	98	82	89	50	63	17	51	66	11	13	19	49	87	43	78	50	74	16	56
	Mem	88	47	17	12	71	7	38	1	52	31	70	80	36	18	74	99	2	78	67	17	97	38	70	98	41	29	79	22	85	65	76	28	96	3	54	86	93	78	57	63	16	1	20	58	0	78	11	57	79	6
S03	CPU	28	94	54	47	19	79	72	63	18	38	18	2	20	87	38	0	97	22	68	68	62	54	86	72	48	91	31	7	84	67	21	97	5	56	38	99	29	1	83	91	52	36	35	32	32	62	97	7	6	51
	Mem	27	8	0	61	65	89	78	56	40	27	89	6	13	34	26	60	34	47	36	2	2	29	46	28	64	24	36	63	85	76	37	16	8	2	28	78	36	81	78	95	92	78	9	28	76	99	65	76	0	54
S04	CPU	82	16	20	46	54	24	33	98	63	56	96	39	31	47	78	68	31	1	21	90	39	87	95	0	24	83	88	42	87	97	45	89	74	72	78	27	36	17	45	85	6	68	99	27	78	63	91	80	14	59
	Mem	71	82	5	55	41	64	29	9	24	39	29	3	24	42	68	83	28	31	56	74	71	84	38	97	64	2	99	51	24	88	15	8	86	16	91	0	83	42	15	11	43	71	33	55	85	62	18	13	6	27
S05	CPU	92	87	66	8	89	76	0	21	77	42	99	41	14	62	19	7	29	8	87	27	88	27	11	0	20	95	85	13	33	34	76	10	90	40	78	83	43	57	56	83	52	68	21	17	56	87	46	47	99	69
	Mem	43	19	55	23	66	72	7	7	88	29	20	58	71	80	30	99	77	84	25	30	10	47	47	69	45	49	98	80	42	77	18	56	97	55	54	89	43	0	46	16	60	30	83	28	8	9	31	46	39	99
S06	CPU	68	22	25	56	32	28	5	72	80	55	7	83	77	82	37	60	20	47	52	55	40	49	93	15	80	6	9	20	3	17	90	44	21	75	51	80	31	24	10	55	69	95	49	7	96	49	28	18	27	29
	Mem	17	49	15	75	86	99	6	80	98	68	77	21	62	64	60	51	35	51	20	46	76	20	22	99	96	54	22	1	88	78	89	69	26	1	34	83	71	13	29	28	2	32	99	4	79	41	60	53	21	69
S07	CPU	87	99	22	83	71	41	98	23	69	2	2	23	20	37	26	59	44	89	31	52	51	10	93	14	6	65	10	89	86	40	70	95	40	16	20	35	6	25	56	52	76	98	6	59	73	0	96	1	72	54
	Mem	4	81	93	31	70	94	70	49	57	48	92	79	58	23	36	34	42	95	28	5	42	63	36	68	24	8	85	54	45	69	90	69	60	76	9	1	39	61	84	20	28	45	93	48	81	77	30	57	2	81
S08	CPU	46	46	0	32	55	16	33	93	80	73	83	99	86	69	39	51	14	82	79	80	20	52	7	73	77	35	49	10	51	84	50	85	78	74	91	56	15	2	70	55	57	20	3	76	75	87	87	42	23	54
	Mem	0	22	45	51	17	51	83	56	32	80	81	57	42	35	29	67	48	21	80	3	15	59	76	20	42	4	70	3	64	74	51	67	99	5	34	69	72	25	57	27	41	24	15	39	14	7	22	77	89	29
S09	CPU	72	18	9	94	2	55	56	77	81	59	80	9	27	41	66	37	98	43	62	57	27	57	53	56	14	30	21	18	3	19	59	12	18	60	38	96	49	38	42	61	99	31	86	55	17	99	66	26	78	62
	Mem	22	21	53	15	19	50	43	18	82	26	48	18	32	69	25	93	91	31	49	76	0	21	99	1	93	69	96	81	54	64	19	87	84	21	65	19	22	52	40	36	43	53	99	42	92	72	57	62	7	37
S10	CPU	4	62	29	10	51	77	43	35	99	44	57	60	41	35	88	78	63	72	28	45	1	30	84	54	66	69	5	60	51	78	39	31	95	81	86	13	83	17	74	86	41	97	81	11	83	70	90	76	72	51
	Mem	19	69	34	79	53	11	0	74	98	72	85	82	54	26	93	56	87	65	41	26	97	24	75	75	45	68	92	94	3	66	59	79	51	91	15	43	1	95	12	16	45	45	93	75	99	16	61	93	13	27
S11	CPU	74	52	85	3	79	95	39	34	73	77	83	73	11	42	27	66	76	80	99	56	43	71	38	16	22	57	95	10	88	89	19	18	22	13	94	20	81	36	27	83	42	47	66	40	44	95	38	6	38	36
	Mem	84	67	79	60	73	88	2	99	60	98	99	7	86	58	69	88	46	91	78	37	92	82	44	10	34	46	9	73	82	1	29	39	47	75	39	61	97	3	88	72	90	98	29	3	39	33	83	8	73	52
S12	CPU	31	23	52	3	49	4	46	86	74	73	18	87	5	42	82	55	67	76	36	16	50	82	94	97	4	13	67	65	6	69	54	26	59	69	46	52	77	84	87	7	82	76	25	11	48	74	35	90	44	50
	Mem	1	53	78	90	54	14	27	24	72	5	67	61	5	45	12	23	29	21	93	49	73	2	99	90	91	38	14	82	17	50	81	67	48	84	14	10	26	80	81	96	63	29	49	65	97	56	5	5	0	57
S13	CPU	11	97	38	18	4	13	13	66	16	75	89	9	84	63	43	25	47	7	69	78	85	12	82	2	56	73	72	62	10	71	38	92	43	60	71	49	37	81	95	86	68	81	35	0	67	42	98	29	63	56
	Mem	80	13	6	11	87	57	22	67	55	36	39	95	81	93	94	98	73	83	34	14	10	31	45	31	32	49	50	64	38	94	86	60	67	23	67	59	7	87	49	1	91	77	14	50	6	99	2	20	52	86
S14	CPU	50	72	89	75	74	38	53	97	59	11	23	67	78	70	41	85	88	91	67	47	75	88	79	10	5	82	11	51	21	46	23	65	5	89	41	95	8	6	88	65	68	94	60	94	45	44	16	19	40	35
	Mem	34	43	54	88	35	70	14	61	58	69	45	68	91	36	70	72	23	22	99	47	71	60	71	54	88	50	27	3	14	76	89	20	90	47	87	93	82	0	51	42	76	87	51	22	70	58	4	14	19	24
S15	CPU	94	48	10	74	65	40	80	45	98	36	15	91	40	94	71	85	59	38	87	14	57	58	44	46	35	63	36	74	70	72	38	10	3	27	61	53	5	3	32	44	12	43	51	22	80	12	78	34	50	3
	Mem	84	82	54	70	49	17	74	34	45	19	33	27	42	78	51	66	3	73	70	73	24	9	38	20	19	19	40	37	13	8	65	52	98	10	91	54	18	9	37	28	94	99	62	74	31	80	77	53	12	6
S16	CPU	17	19	81	50	17	89	79	71	53	32	46	55	32	91	40	26	92	69	0	11	14	84	3	0	78	39	34	13	37	95	16	1	77	76	65	67	17	37	75	17	92	29	70	71	80	98	80	82	72	98
	Mem	35	53	30	24	52	20	46	51	11	92	34	22	20	68	79	17	97	33	56	17	94	62	42	27	95	72	12	19	29	65	14	52	61	64	58	78	13	73	22	63	78	38	95	37	0	92	44	35	21	80
S17	CPU	76	83	91	68	41	84	53	98	27	11	97	53	68	61	22	51	21	74	67	40	20	43	88	12	84	70	21	77	94	25	21	40	96	0	49	34	35	3	20	51	0	90	79	65	85	31	44	59	45	6
	Mem	84	52	21	38	54	86	34	19	58	25	4	97	93	31	88	14	3	92	49	70	86	17	0	62	14	37	97	88	37	39	99	13	34	43	11	69	66	21	20	81	41	29	13	82	50	95	12	62	34	32
S18	CPU	59	0	85	20	4	88	25	51	2	93	95	48	77	53	58	34	36	57	55	59	18	63	72	2	24	16	34	53	23	19	10	88	48	9	41	15	99	71	85	37	73	7	71	62	24	3	85	14	20	28
	Mem	40	13	96	26	16	42	4	97	70	35	50	75	17	12	97	72	78	58	45	66	99	87	6	98	34	20	26	62	82	69	38	66	93	73	41	77	89	47	72	71	20	64	9	74	42	76	70	36	54	71
S19	CPU	8	86	99	36	74	74	74	8	87	66	42	24	80	76	87	45	99	95	10	74	79	73	18	77	79	54	86	87	35	72	73	7	21	50	85	78	35	78	35	88	21	77	95	3	75	14	0	66	95	99
	Mem	21	46	19	20	54	94	6	36	6	5	56	30	79	20	92	58	9	85	74	89	30	60	39	6	73	90	75	75	38	32	53	10	84	63	93	33	28	25	46	0	38	20	95	34	28	35	54	98	14	31
S20	CPU	63	26	35	53	2	65	10	84	17	64	61	54	25	60	93	49	83	21	79	30	6	42	63	15	11	28	26	18	40	80	83	48	95	11	41	90	33	11	12	52	95	69	57	75	23	69	95	76	54	5
	Mem	43	56	20	58	41	6	20	57	98	88	4	67	1	69	59	98	2	71	24	69	41	86	64	52	9	70	22	7	84	60	99	94	21	18	1	1	47	92	52	2	88	40	1	62	49	21	99	94	71	58
S21	CPU	76	51	54	13	13	57	15	42	67	26	88	37	90	48	24	84	17	56	58	70	17	87	39	69	87	3	38	52	0	76	62	25	8	97	1	43	66	71	52	99	90	15	74	53	79	12	73	11	65	8
	Mem	10	47	43	59	73	59	57	19	79	6	34	84	5	90	53	15	90	7	13	26	28	93	56	59	38	94	24	69	1	56	37	62	73	49	15	21	77	28	23	42	29	19	53	73	90	44	45	53	97	85
S22	CPU	1	86	97	46	60	49	99	80	45	93	12	20	13	85	19	32	6	92	66	57	16	68	94	32	76	20	68	2	96	57	45	35	19	25	94	71	97	15	71	21	27	24	58	38	71	1	49	33	76	48
	Mem	51	65	70	14	21	38	30	95	63	17	96	25	53	97	78	47	24	11	71	3	16	5	87	83	43	77	72	97	64	88	83	70	49	29	81	30	91	57	13	0	1	91	13	76	96	41	75	41	40	97
S23	CPU	5	7	95	34	33	80	26	87	60	96	42	64	43	25	17	6	25	48	7	96	68	93	68	74	10	14	73	22	67	7	65	77	22	57	50	98	93	27	15	27	96	74	48	48	31	64	94	78	23	87
	Mem	39	39	32	61	47	19	70	5	17	73	69	86	48	29	40	86	26	7	77	95	35	61	18	87	65	30	58	90	39	20	27	19	89	41	98	29	61	9	61	9	83	54	24	60	59	83	44	75	17	1
S24	CPU	31	93	52	93	73	24	82	42	23	34	40	46	36	91	62	32	55	68	46	48	31	73	65	9	58	66	63	37	99	11	48	78	21	99	31	3	39	56	49	60	25	16	20	15	33	8	11	39	87	22
	Mem	15	8	72	87	56	81	7	65	62	77	85	30	67	2	13	87	78	86	6	58	77	31	37	92	67	98	19	81	74	51	90	22	24	37	7	47	81	69	37	55	55	61	95	5	27	75	66	25	79	88
S25	CPU	55	21	74	25	99	32	94	86	71	95	72	17	51	14	39	49	43	96	58	43	83	83	55	67	29	85	75	16	63	61	92	4	34	65	47	71	55	20	30	35	96	23	49	75	86	24	12	69	57	19
	Mem	39	16	72	9	34	84	94	15	2	14	51	56	84	32	48	22	19	99	83	32	0	50	9	11	89	44	81	8	7	23	20	34	52	19	60	40	77	58	29	44	67	16	63	86	82	76	48	19	75	12
S26	CPU	94	64	34	89	62	99	96	70	34	20	63	22	75	27	65	67	5	69	97	54	5	2	63	90	94	4	20	55	56	12	65	47	32	2	11	97	5	42	57	73	99	1	6	21	26	91	85	18	44	17
	Mem	53	30	86	6	85	7	46	78	70	25	98	23	47	2	56	57	27	66	18	31	64	43	78	83	2	38	91	72	9	47	47	99	95	63	44	81	99	11	79	56	1	31	52	81	87	94	3	88	76	64
S27	CPU	48	70	38	28	51	21	66	27	41	61	35	99	28	79	39	71	62	27	35	91	20	35	54	35	41	23	98	23	41	21	4	71	53	38	20	70	68	18	29	61	70	93	23	2	8	53	11	57	0	31
	Mem	32	18	67	6	53	86	66	2	4	68	76	40	78	14	4	63	99	14	72	70	38	73	94	30	34	99	64	96	81	28	45	88	30	49	79	79	66	30	74	87	26	85	90	13	25	22	74	88	93	98
S32	CPU	3	4	43	47	72	25	43	39	14	22	35	28	47	60	6	9	62	47	30	52	52	99	86	61	52	43	35	93	49	2	80	24	1	44	46	51	85	82	39	28	79	40	23	88	14	88	83	30	26	10
	Mem	75	40	1	88	18	20	44	24	69	6	91	12	48	75	61	9	52	49	79	66	20	23	8	12	19	38	88	78	16	21	43	9	14	49	34	19	24	20	40	71	87	37	55	80	16	43	92	57	63	99
S33	CPU	76	93	24	81	46	8	93	90	2	67	86	22	92	30	13	17	45	54	54	53	90	90	90	72	29	89	84	99	41	13	85	46	86	89	39	23	34	31	29	30	91	41	71	81	63	75	12	56	95	33
	Mem	96	60	54	28	46	65	16	38	65	82	93	84	68	18	85	27	44	80	49	95	60	2	86	96	71	71	35	60	66	18	64	26	22	17	92	96	2	16	96	60	83	27	14	68	3	46	60	97	20	73
S34	CPU	95	40	19	14	7	67	79	26	14	99	32	88	24	6	21	55	8	78	61	8	9	74	37	41	39	4	5	93	55	34	59	37	27	60	3	92	26	34	28	0	82	88	34	40	13	23	1	21	6	2
	Mem	3	93	95	27	91	96	64	4	71	45	76	19	52	19	45	74	60	55	84	9	70	13	49	72	76	77	15	9	77	0	64	48	99	17	56	72	85	17	27	2	20	18	17	23	92	8	13	54	86	49
S35	CPU	59	9	30	60	8	75	78	63	88	15	26	18	2	78	6	87	2	57	25	54	95	20	88	89	6	29	62	3	5	85	52	27	97	3	76	49	37	85	60	62	76	98	2	64	46	79	44	77	7	90
	Mem	94	73	96	60	11	90	71	60	73	43	92	50	42	13	95	39	11	75	73	84	69	5	48	76	94	79	99	58	35	17	99	26	64	99	60	21	89	82	75	93	11	88	27	48	90	49	38	15	9	14
S36	CPU	15	94	90	93	66	80	27	93	10	63	15	29	15	2	90	98	64	99	46	64	71	38	94	68	54	6	72	40	42	91	28	64	17	90	99	9	40	82	32	29	8	97	67	52	4	90	18	13	69	4
	Mem	27	95	39	74	13	83	92	55	42	74	13	50	95	17	30	26	77	59	40	79	72	19	24	80	49	56	19	52	98	27	61	3	42	45	67	82	88	15	90	0	35	59	15	65	59	8	99	24	9	7
S37	CPU	40	80	2	72	54	37	36	31	56	76	22	37	8	97	21	55	2	68	72	97	28	59	81	95	43	91	55	55	49	4	19	87	29	43	85	73	53	38	72	21	33	31	34	97	23	0	49	69	12	38
	Mem	83	89	67	0	16	93	89	39	42	97	10	1	9	8	77	25	16	90	69	99	99	81	94	64	40	93	47	23	35	20	26	92	25	18	81	6	70	42	62	53	19	45	16	14	95	94	97	54	15	32
S38	CPU	45	22	5	83	2	47	26	39	19	91	67	69	88	66	39	78	76	50	12	19	94	63	9	35	5	84	85	33	39	31	22	36	41	98	98	82	63	87	27	38	12	77	82	54	90	33	83	77	17	29
	Mem	35	65	96	99	57	21	36	14	75	27	83	76	75	5	74	16	63	71	82	69	81	11	53	81	82	5	68	24	73	39	14	9	7	51	48	99	16	16	18	15	66	54	26	46	64	35	95	63	21	71
S39	CPU	36	18	21	40	53	77	55	35	83	54	94	88	45	54	47	25	25	68	97	37	17	74	66	80	12	74	28	58	96	65	52	36	56	55	21	0	34	54	29	70	20	82	12	49	54	90	60	87	96	99
	Mem	60	15	81	2	16	90	29	30	51	37	8	92	93	60	37	56	69	82	86	97	26	45	71	54	56	83	46	77	84	24	42	75	69	0	11	37	35	31	1	94	12	5	93	5	27	1	78	28	48	14
S44	CPU	18	64	76	81	38	68	85	52	11	80	83	36	14	68	61	23	31	65	88	7	87	50	6	22	96	77	66	31	30	49	16	96	10	61	21	88	31	20	2	48	79	16	18	20	29	99	34	10	67	82
	Mem	9	86	82	40	54	60	70	17	37	40	2	74	57	92	31	42	58	20	1	72	7	64	49	87	75	24	57	61	8	69	42	28	20	65	93	99	11	27	51	92	93	94	58	91	2	58	98	74	68	29
S45	CPU	89	30	95	4	13	45	38	39	12	34	7	12	33	36	92	1	98	14	6	89	57	17	90	77	59	95	34	37	22	35	55	31	76	60	0	98	67	66	31	34	72	43	38	23	33	0	18	74	65	5
	Mem	99	41	73	70	24	30	62	30	50	26	6	93	97	24	64	54	36	31	31	77	61	38	95	62	12	32	3	99	26	60	41	92	37	32	56	13	96	54	8	13	38	10	87	68	64	43	94	73	39	37
S46	CPU	42	44	12	19	85	57	41	67	60	38	28	80	87	70	1	10	19	3	89	87	41	99	38	35	35	5	40	98	6	20	78	36	30	56	21	75	74	6	19	32	4	12	70	5	39	87	22	81	99	89
	Mem	46	19	28	81	15	84	54	90	90	34	21	32	16	13	95	35	63	5	69	5	2	2	61	91	18	55	7	50	45	42	93	69	99	68	18	21	81	48	58	60	28	4	59	75	6	56	21	98	38	83
S47	CPU	46	41	99	94	14	33	16	66	4	45	40	46	48	96	65	29	77	99	0	44	42	66	19	32	18	28	42	51	12	50	34	28	84	30	9	87	98	11	49	28	59	84	99	5	99	14	20	16	77	42
	Mem	78	21	81	51	6	3	99	26	80	69	0	4	78	39	30	57	48	31	18	89	4	75	15	96	16	12	3	4	96	85	24	88	41	22	84	32	45	90	92	76	86	37	95	97	75	57	40	54	69	30

A.2 Real World Benchmark

Table A.5: Google cluster data benchmark scenario #1

(a) Workload pattern

Workload		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
S1	CPU	1	3	3	33	41	41	41	44	47	47	47	44	44	44	43	40	37	42	46	51	51	51	61	69	69	69	76	79	79	79
	Mem	1	1	1	8	10	10	10	13	18	19	20	25	24	25	23	17	10	12	11	10	10	10	15	18	18	18	30	36	36	36
S2	CPU	36	36	36	36	19	16	20	16	18	33	36	36	36	35	29	23	21	24	26	30	38	38	38	38	38	44	44	41	34	34
	Mem	11	11	11	11	7	9	9	8	8	19	25	25	25	24	21	22	21	22	24	25	26	26	26	26	27	32	32	31	27	27
S3	CPU	35	35	35	35	35	35	35	31	27	27	27	27	27	27	27	27	27	27	27	27	23	15	13	13	13	13	13	13	13	13
	Mem	7	7	7	7	7	7	7	6	6	6	6	6	6	6	6	6	6	6	6	6	5	3	3	3	3	3	3	3	3	3
S4	CPU	32	32	33	33	33	33	33	33	33	33	33	32	33	33	34	34	35	34	32	32	32	33	33	33	33	33	33	33	32	31
	Mem	10	10	10	10	11	10	10	10	11	11	11	10	11	11	11	11	12	11	11	10	10	11	11	11	11	10	11	11	11	11

(b) Workload distribution #1

vDom0		vDom1		vDom3		vDom4		vDom5		vDom7	
s4	s4	n/a	n/a	s4	s4	s1	s1	s2	s4	s2	s3
s3	s3	n/a	n/a	s3	s4	s3	s3	s4	s1	s2	s2
s2	s3	n/a	n/a	s2	s2	s2	s4	s2	s2	s3	s4
s1	s1	n/a	n/a	s1	s3	s1	s4	s1	s3	s1	s1

(c) Workload distribution #2

vDom0		vDom1		vDom3		vDom4		vDom5		vDom7	
s1	s3	n/a	n/a	s2	s2	s3	s2	s3	s4	s4	s1
s2	s1	n/a	n/a	s4	s2	s1	s4	s4	s1	s1	s1
s1	s4	n/a	n/a	s2	s2	s3	s4	s4	s4	s3	s1
s2	s2	n/a	n/a	s3	s3	s2	s3	s1	s4	s3	s3

(d) Workload distribution #3

vDom0		vDom1		vDom3		vDom4		vDom5		vDom7	
s4	s4	n/a	n/a	s4	s2	s1	s2	s3	s1	s4	s3
s3	s2	n/a	n/a	s4	s4	s1	s4	s3	s3	s4	s3
s1	s2	n/a	n/a	s2	s2	s3	s4	s4	s1	s1	s2
s3	s1	n/a	n/a	s1	s2	s3	s2	s1	s2	s3	s1

Table A.6: Google cluster data benchmark scenario #2

(a) Workload pattern

Workload		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
S1	CPU	1	3	3	33	41	41	41	44	47	47	47	44	44	44	43	40	37	42	46	51	51	51	61	69	69	69	76	79	79	79
	Mem	1	1	1	8	10	10	10	13	18	19	20	25	24	25	23	17	10	12	11	10	10	10	15	18	18	18	30	36	36	36
S2	CPU	38	48	75	86	82	92	85	87	91	93	91	93	89	73	36	25	26	26	28	28	25	24	26	21	19	21	23	31	18	31
	Mem	15	18	21	27	28	32	30	30	31	34	31	31	31	29	19	17	16	15	16	17	16	15	14	13	10	10	10	15	11	22
S3	CPU	16	9	5	4	7	47	80	92	78	83	91	86	89	87	79	91	91	93	93	90	91	91	84	86	66	59	69	79	62	47
	Mem	6	4	3	2	2	9	15	16	15	16	18	14	15	15	16	16	16	17	15	14	13	12	14	18	16	19	18	16	12	12
S4	CPU	85	82	83	69	67	84	49	63	81	92	90	76	70	58	18	14	14	15	15	14	14	12	10	15	24	24	25	28	27	69
	Mem	16	14	16	23	29	35	22	18	35	51	58	39	24	21	10	6	7	8	7	7	7	6	5	5	7	7	7	9	8	17
S5	CPU	27	25	60	79	82	85	84	84	85	87	83	60	62	77	80	84	82	84	85	89	82	69	19	13	12	13	13	15	15	25
	Mem	14	12	19	23	25	29	25	26	26	28	26	21	22	25	27	28	28	28	29	30	30	28	13	10	9	9	9	11	10	17
S6	CPU	28	28	26	25	26	25	27	30	29	28	25	24	24	25	27	26	31	28	29	29	28	25	24	23	23	26	25	25	28	26
	Mem	18	20	20	17	15	14	17	18	19	17	15	15	15	19	21	18	21	20	22	22	21	19	17	13	15	16	16	17	17	18
S7	CPU	35	55	40	38	49	66	38	23	19	25	24	20	22	34	31	37	35	28	32	28	27	22	35	43	24	27	21	25	30	28
	Mem	14	19	20	17	21	29	19	12	10	14	12	10	10	13	16	17	14	15	18	14	13	14	13	22	10	12	10	10	11	13

(b) Workload distribution #1

vDom0		vDom1		vDom3		vDom4		vDom5		vDom7	
s4	s6	n/a	n/a	s4	s5	s3	s5	s5	s5	s5	s7
s3	s5	n/a	n/a	s3	s4	s6	s5	s4	s4	s2	s1
s2	s3	n/a	n/a	s2	s4	s2	s4	s3	s6	s3	s7
s1	s1	n/a	n/a	s1	s2	s1	s3	s1	s2	s1	s2

(c) Workload distribution #2

vDom0		vDom1		vDom3		vDom4		vDom5		vDom7	
s3	s7	n/a	n/a	s4	s3	s5	s4	s7	s3	s7	s7
s5	s5	n/a	n/a	s6	s7	s1	s2	s1	s4	s5	s3
s3	s6	n/a	n/a	s2	s2	s6	s6	s6	s2	s1	s2
s4	s5	n/a	n/a	s5	s1	s3	s1	s1	s4	s6	s4

(d) Workload distribution #3

vDom0		vDom1		vDom3		vDom4		vDom5		vDom7	
s2	s4	n/a	n/a	s5	s4	s4	s6	s4	s3	s1	s1
s7	s6	n/a	n/a	s2	s7	s1	s2	s6	s5	s1	s2
s1	s5	n/a	n/a	s7	s5	s7	s3	s5	s2	s7	s6
s6	s1	n/a	n/a	s2	s3	s4	s3	s3	s5	s6	s3

Table A.7: Google cluster data benchmark scenario #3

(a) Workload pattern

Workload		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
S1	CPU	1	3	3	33	41	41	41	44	47	47	47	44	44	44	43	40	37	42	46	51	51	51	61	69	69	69	76	79	79	79
	Mem	1	1	1	8	10	10	10	13	18	19	20	25	24	25	23	17	10	12	11	10	10	10	15	18	18	18	30	36	36	36
S2	CPU	38	48	75	86	82	92	85	87	91	93	91	93	89	73	36	25	26	26	28	28	25	24	26	21	19	21	23	31	18	31
	Mem	15	18	21	27	28	32	30	30	31	34	31	31	31	29	19	17	16	15	16	17	16	15	14	13	10	10	10	15	11	22
S3	CPU	51	51	44	45	49	51	58	40	39	41	40	38	30	25	22	23	24	23	22	35	44	43	43	54	38	19	39	57	52	50
	Mem	9	9	10	9	8	9	10	6	6	8	7	5	7	7	7	7	6	6	6	6	6	7	8	10	7	5	9	12	9	8
S4	CPU	91	91	90	87	84	86	87	86	73	12	3	0	0	0	0	0	2	31	1	3	4	2	2	21	26	28	28	28	28	28
	Mem	38	38	39	38	37	37	38	36	30	1	0	0	0	0	0	0	1	43	1	1	1	1	1	3	3	3	3	3	3	3
S5	CPU	27	25	60	79	82	85	84	84	85	87	83	60	62	77	80	84	82	84	85	89	82	69	19	13	12	13	13	15	15	25
	Mem	14	12	19	23	25	29	25	26	26	28	26	21	22	25	27	28	28	28	29	30	30	28	13	10	9	9	9	11	10	17
S6	CPU	1	1	1	10	30	36	38	40	40	39	40	41	40	40	40	40	41	35	25	8	1	1	1	1	1	1	1	1	1	1
	Mem	1	1	1	2	3	6	7	5	5	5	5	5	7	8	5	5	5	4	3	2	2	1	1	1	1	1	1	2	1	1
S7	CPU	1	1	1	26	58	67	57	37	35	31	31	32	35	39	44	49	39	29	20	12	10	8	14	16	16	16	14	14	11	10
	Mem	1	1	1	7	15	16	13	9	9	9	9	9	10	10	13	17	13	9	6	4	4	3	5	6	6	6	5	5	3	3

(b) Workload distribution #1

vDom0		vDom1		vDom3		vDom4		vDom5		vDom7	
s4	s6	n/a	n/a	s4	s5	s3	s5	s5	s5	s5	s7
s3	s5	n/a	n/a	s3	s4	s6	s5	s4	s4	s2	s1
s2	s3	n/a	n/a	s2	s4	s2	s4	s3	s6	s3	s7
s1	s1	n/a	n/a	s1	s2	s1	s3	s1	s2	s1	s2

(c) Workload distribution #2

vDom0		vDom1		vDom3		vDom4		vDom5		vDom7	
s4	s6	n/a	n/a	s3	s7	s3	s4	s3	s5	s6	s7
s6	s1	n/a	n/a	s1	s4	s2	s1	s2	s6	s2	s5
s7	s6	n/a	n/a	s3	s6	s1	s5	s1	s5	s3	s2
s4	s7	n/a	n/a	s2	s5	s5	s3	s4	s2	s1	s4

(d) Workload distribution #3

vDom0		vDom1		vDom3		vDom4		vDom5		vDom7	
s2	s5	n/a	n/a	s7	s3	s6	s3	s1	s6	s2	s7
s1	s4	n/a	n/a	s3	s1	s1	s3	s5	s6	s4	s5
s6	s2	n/a	n/a	s1	s4	s4	s7	s3	s4	s7	s6
s4	s2	n/a	n/a	s6	s5	s7	s2	s5	s3	s5	s2

Table A.8: Google cluster data benchmark scenario #4

(a) Workload pattern

Workload		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
S1	CPU	33	39	59	54	47	66	74	83	83	82	69	51	47	45	34	32	31	33	36	34	33	23	23	45	49	58	58	57	47	43
	Mem	18	19	24	23	21	21	21	30	30	27	26	24	25	25	20	20	17	18	19	19	18	17	17	22	23	25	26	27	23	22
S2	CPU	26	27	28	27	25	27	28	26	26	27	28	29	26	25	28	28	27	28	31	31	33	36	31	29	29	31	31	31	22	20
	Mem	7	12	6	6	6	6	7	6	6	7	7	7	7	7	7	7	7	8	9	9	9	9	9	8	8	8	8	8	7	6
S3	CPU	70	67	56	49	40	52	57	67	63	45	58	71	74	76	64	64	79	76	71	56	45	59	62	71	65	54	61	68	64	63
	Mem	25	23	18	16	14	18	20	20	19	12	16	20	21	22	18	18	19	18	17	17	16	21	22	24	22	17	21	24	24	24
S4	CPU	26	27	28	27	25	27	28	26	26	27	28	29	26	25	28	28	27	28	31	31	33	36	31	29	29	31	31	31	22	20
	Mem	7	12	6	6	6	6	7	6	6	7	7	7	7	7	7	7	7	8	9	9	9	9	9	8	8	8	8	8	7	6
S5	CPU	56	55	54	48	40	47	49	44	44	45	54	53	39	41	38	37	37	50	48	59	59	49	29	37	32	22	21	41	27	22
	Mem	11	11	11	10	10	8	7	5	5	6	8	10	8	8	9	9	8	9	9	10	10	9	7	8	7	4	5	8	8	8
S6	CPU	48	47	57	48	37	40	41	34	33	32	28	23	25	26	32	33	25	27	31	28	26	30	31	38	42	53	48	43	37	28
	Mem	18	19	22	20	17	20	21	15	15	19	17	15	13	12	15	20	15	13	14	15	16	17	18	23	22	20	19	19	16	15
S7	CPU	19	21	26	25	23	18	16	23	22	18	17	14	16	17	15	15	15	15	16	16	15	19	20	17	17	18	20	22	18	16
	Mem	15	15	15	15	15	16	16	20	20	17	17	16	17	18	18	18	14	15	17	18	18	20	21	17	17	18	18	17	14	13

(b) Workload distribution #1

vDom0		vDom1		vDom3		vDom4		vDom5		vDom7	
s4	s6	n/a	n/a	s4	s6	s3	s5	s5	s5	s5	s7
s3	s5	n/a	n/a	s3	s6	s6	s5	s4	s4	s2	s1
s2	s3	n/a	n/a	s2	s4	s2	s4	s3	s6	s3	s7
s1	s1	n/a	n/a	s7	s2	s1	s3	s1	s2	s1	s2

(c) Workload distribution #2

vDom0		vDom1		vDom3		vDom4		vDom5		vDom7	
s6	s5	n/a	n/a	s7	s2	s2	s1	s6	s7	s4	s2
s6	s7	n/a	n/a	s7	s3	s4	s7	s4	s4	s2	s5
s2	s3	n/a	n/a	s3	s3	s1	s3	s6	s1	s6	s6
s5	s3	n/a	n/a	s1	s2	s4	s7	s1	s5	s5	s1

(d) Workload distribution #3

vDom0		vDom1		vDom3		vDom4		vDom5		vDom7	
s3	s1	n/a	n/a	s3	s6	s6	s5	s3	s2	s6	s5
s2	s6	n/a	n/a	s4	s5	s7	s7	s7	s1	s1	s6
s3	s1	n/a	n/a	s4	s5	s6	s7	s4	s3	s2	s7
s5	s4	n/a	n/a	s1	s2	s4	s5	s4	s2	s1	s2

Table A.9: Google cluster data benchmark scenario #5

(a) Workload patterns

Workload		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
S1	CPU	46	44	35	39	43	44	44	44	45	53	48	40	41	42	40	40	42	42	43	43	45	45	46	57	57	57	56	56	40	35
	Mem	31	29	23	25	26	28	29	29	30	39	32	22	25	27	25	25	26	27	29	30	33	35	35	41	41	40	41	41	28	25
S2	CPU	59	57	52	53	53	55	56	60	61	66	70	75	66	61	50	49	47	47	48	50	51	69	73	70	69	68	68	68	60	32
	Mem	21	21	21	20	19	20	21	23	23	25	26	26	26	25	19	18	18	18	18	19	20	25	27	26	25	24	25	25	25	21
S3	CPU	27	26	22	21	20	18	17	20	21	22	19	19	16	14	20	21	21	21	22	23	23	26	27	28	28	27	22	17	13	12
	Mem	11	11	11	11	6	7	8	8	8	8	7	6	5	5	5	5	5	4	4	4	5	6	7	8	8	8	7	7	5	4
S4	CPU	42	41	37	37	37	37	37	26	28	40	38	40	42	43	35	37	35	36	39	42	44	42	42	40	39	38	34	30	29	36
	Mem	22	22	20	20	19	19	19	11	12	19	20	22	23	24	20	19	16	18	22	23	24	22	21	22	22	20	19	18	18	21
S5	CPU	20	20	19	17	38	53	60	45	44	44	44	50	56	57	44	44	44	39	18	15	18	14	16	15	16	12	12	31	15	10
	Mem	7	7	8	10	9	11	13	8	8	8	8	11	11	11	8	8	9	9	8	6	6	5	5	8	8	6	7	10	7	7
S6	CPU	43	40	38	39	39	39	40	47	49	53	53	52	51	50	13	13	46	47	49	50	51	54	53	52	51	51	51	52	53	53
	Mem	8	7	6	6	6	6	7	7	7	9	9	9	9	8	5	5	10	10	10	10	10	10	10	10	7	7	7	7	7	7
S7	CPU	73	69	54	52	49	47	46	40	42	54	47	37	27	21	19	18	18	18	19	20	20	24	25	28	27	25	24	22	21	20
	Mem	27	25	19	18	17	18	18	15	16	19	17	13	11	9	8	8	7	7	8	9	9	12	13	18	16	12	12	12	10	9

(b) Workload distribution #1

vDom0		vDom1		vDom3		vDom4		vDom5		vDom7	
s2	s5	n/a	n/a	s4	s6	s2	s7	s1	s7	s7	s2
s1	s3	n/a	n/a	s2	s6	s7	s5	s5	s1	s4	s4
s4	s3	n/a	n/a	s5	s1	s4	s1	s4	s2	s5	s3
s1	s2	n/a	n/a	s6	s2	s4	s3	s7	s3	s6	s3

(c) Workload distribution #2

vDom0		vDom1		vDom3		vDom4		vDom5		vDom7	
s7	s4	n/a	n/a	s3	s5	s2	s6	s1	s1	s3	s7
s2	s4	n/a	n/a	s3	s6	s5	s7	s4	s5	s7	s2
s7	s2	n/a	n/a	s7	s5	s1	s6	s4	s1	s6	s5
s6	s5	n/a	n/a	s3	s3	s6	s2	s3	s1	s4	s1

(d) Workload distribution #3

vDom0		vDom1		vDom3		vDom4		vDom5		vDom7	
s3	s2	n/a	n/a	s5	s3	s6	s7	s4	s5	s5	s6
s1	s3	n/a	n/a	s4	s5	s6	s2	s3	s7	s6	s7
s7	s4	n/a	n/a	s2	s1	s1	s2	s3	s1	s7	s2
s5	s4	n/a	n/a	s4	s7	s1	s6	s6	s4	s5	s1

Table A.10: Google cluster data benchmark scenario #6

(a) Workload patterns

Workload		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
S1	CPU	24	24	20	19	20	37	44	43	44	46	47	48	46	46	19	19	41	32	17	17	18	40	45	47	46	45	46	48	24	17
	Mem	10	10	8	7	7	6	6	8	8	10	11	13	11	9	7	6	5	6	6	8	9	8	8	10	10	10	9	9	8	7
S2	CPU	22	22	30	30	23	12	12	15	17	22	22	22	24	26	34	35	34	31	28	25	21	15	14	11	13	19	19	19	15	19
	Mem	7	7	8	7	6	5	5	5	5	6	6	7	7	7	8	8	8	8	7	6	5	5	6	5	6	9	10	11	7	6
S3	CPU	27	31	41	32	22	32	42	30	29	41	44	48	35	27	30	29	26	24	21	23	24	23	23	20	20	19	16	14	23	26
	Mem	24	22	12	10	7	11	15	9	9	16	29	15	13	11	16	16	11	11	11	11	11	9	9	10	10	9	9	9	24	27
S4	CPU	56	53	42	47	52	60	63	50	51	58	59	60	48	42	43	42	40	37	30	37	43	44	44	34	43	64	60	56	66	69
	Mem	30	28	23	24	26	38	43	27	26	25	26	29	27	26	25	24	18	18	17	23	27	30	30	30	29	26	27	28	29	30
S5	CPU	25	21	10	10	11	14	15	17	16	11	15	21	14	11	9	12	13	16	21	10	10	28	25	12	17	29	29	28	16	13
	Mem	11	10	7	11	15	11	9	5	5	6	7	8	7	6	6	7	5	7	10	7	7	11	12	7	8	10	10	14	8	6
S6	CPU	93	91	84	70	54	45	42	43	51	88	88	86	86	86	90	89	77	63	38	68	90	84	72	57	58	60	63	65	65	65
	Mem	23	20	10	9	9	6	5	7	9	16	16	15	19	21	14	13	10	9	6	12	16	16	15	7	10	15	16	17	18	19
S7	CPU	19	20	19	19	13	16	17	20	20	20	19	20	18	17	18	20	27	27	27	29	31	32	32	31	27	19	20	21	21	20
	Mem	8	8	7	6	4	6	6	5	5	4	5	8	8	8	5	5	8	8	9	9	9	8	8	7	8	8	7	6	7	7
S8	CPU	30	31	33	31	28	27	28	26	28	38	34	30	31	32	28	28	26	28	30	36	37	32	31	30	30	30	32	35	31	29
	Mem	20	21	24	23	20	19	18	18	18	18	19	20	20	21	19	18	16	17	19	22	24	21	20	20	19	19	21	22	21	20
S9	CPU	26	26	26	26	39	33	26	26	24	23	24	20	19	18	19	19	20	18	15	20	22	27	28	25	25	24	25	26	22	21
	Mem	20	20	22	21	21	19	17	18	17	16	16	16	17	17	16	16	16	16	16	17	17	21	22	21	20	19	21	23	19	18
S10	CPU	94	93	92	94	95	95	96	96	95	95	95	95	95	96	96	95	95	94	92	92	93	90	90	92	91	90	90	90	92	92
	Mem	53	54	58	54	49	45	44	48	49	50	51	52	54	55	46	46	49	52	57	56	57	47	46	48	47	45	51	56	51	49

(b) Distribution #1

vDom0		vDom1		vDom3		vDom4		vDom5		vDom7	
s7	s6	n/a	n/a	s1	s6	s4	s5	s3	s10	s5	s9
s8	s4	n/a	n/a	s2	s1	s4	s2	s3	s10	s6	s9
s8	s3	n/a	n/a	s5	s2	s1	s3	s7	s8	s9	s8
s7	s10	n/a	n/a	s4	s5	s1	s2	s6	s7	s10	s9

(c) Distribution #2

vDom0		vDom1		vDom3		vDom4		vDom5		vDom7	
s5	s2	n/a	n/a	s1	s4	s2	s4	s8	s10	s9	s5
s10	s2	n/a	n/a	s2	s7	s6	s6	s9	s1	s5	s4
s10	s1	n/a	n/a	s3	s8	s6	s10	s8	s7	s7	s3
s6	s5	n/a	n/a	s3	s9	s8	s7	s9	s1	s3	s4

(d) Distribution #3

vDom0		vDom1		vDom3		vDom4		vDom5		vDom7	
s6	s6	n/a	n/a	s7	s7	s5	s9	s3	s5	s6	s1
s10	s10	n/a	n/a	s5	s3	s4	s8	s9	s2	s5	s7
s9	s1	n/a	n/a	s7	s8	s4	s9	s1	s8	s10	s8
s4	s4	n/a	n/a	s2	s3	s3	s1	s2	s10	s6	s2

Table A.11: Google cluster data benchmark scenario #7

Workload		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
S00	CPU	37	34	32	34	36	29	27	36	36	31	38	47	40	35	46	48	50	52	57	51	47	42	41	36	35	36	40	43	43	43
	Mem	16	16	16	15	13	10	9	15	16	17	19	22	19	18	21	21	18	18	19	21	22	21	21	19	18	18	20	21	19	18
S01	CPU	21	20	13	15	17	20	21	21	23	30	29	27	26	26	31	31	30	31	33	32	31	20	18	18	19	20	24	28	16	12
	Mem	6	6	7	5	4	6	7	6	6	7	7	7	7	6	8	8	7	7	7	7	7	7	7	7	7	7	7	7	7	6
S02	CPU	22	20	19	23	27	20	18	15	15	17	19	21	19	16	15	15	14	13	13	15	16	24	25	29	29	25	27	33	30	29
	Mem	14	13	13	13	14	12	12	10	10	10	11	12	9	7	7	7	11	10	10	11	11	15	15	15	16	17	18	20	19	18
S03	CPU	37	38	41	41	41	41	41	38	37	34	34	35	36	36	30	30	29	31	35	43	46	44	43	38	38	37	41	41	38	37
	Mem	21	22	24	22	20	21	21	17	17	19	21	22	20	18	17	17	17	21	27	27	26	22	21	22	20	17	21	25	22	22
S04	CPU	34	31	35	39	43	40	38	42	40	28	30	34	33	33	35	36	41	38	32	39	44	35	33	31	33	35	32	30	32	33
	Mem	15	15	16	18	19	19	18	19	18	13	14	15	14	14	15	16	18	17	14	17	18	15	14	12	13	14	13	12	14	14
S05	CPU	29	31	33	32	30	34	35	30	34	32	31	30	29	28	36	37	34	30	33	31	30	31	31	33	35	33	36	39	35	34
	Mem	14	14	16	16	16	15	14	14	13	10	12	14	13	12	16	17	14	14	14	15	15	15	15	15	15	15	14	13	15	16
S06	CPU	61	65	81	77	83	84	85	75	69	78	80	85	88	88	90	90	85	79	68	64	69	55	38	58	63	73	73	78	80	79
	Mem	36	41	56	45	35	29	26	40	41	44	39	31	34	36	32	31	29	28	25	32	52	27	24	27	27	25	27	29	29	29
S07	CPU	38	39	37	35	33	38	39	51	49	37	37	37	42	45	45	44	35	34	32	35	38	48	50	42	42	43	48	56	60	62
	Mem	6	6	4	4	4	4	4	5	5	5	5	4	5	5	5	5	4	4	3	4	4	5	6	6	5	4	6	8	7	7
S08	CPU	39	42	53	47	40	41	41	63	63	57	61	66	44	32	57	57	31	46	46	47	50	49	48	28	34	48	54	59	49	47
	Mem	22	22	21	17	12	12	13	25	26	25	21	16	13	11	17	17	9	12	16	17	16	17	17	10	12	16	19	21	18	17
S09	CPU	23	23	24	24	24	23	23	23	23	24	24	23	23	24	24	24	27	27	29	29	28	27	27	26	26	25	25	25	25	25
	Mem	16	17	18	18	19	17	16	17	17	16	17	18	17	17	16	16	18	18	17	19	20	19	19	18	18	19	19	20	19	18
S10	CPU	55	53	49	50	50	48	48	46	45	42	48	57	55	53	52	52	48	49	49	53	54	53	53	58	57	57	55	54	51	43
	Mem	46	45	40	37	34	36	37	38	38	38	44	52	49	47	40	39	38	39	41	46	47	45	46	55	54	52	48	45	44	35
S11	CPU	50	47	41	43	45	39	37	34	33	31	36	38	38	38	34	33	34	42	57	56	55	46	44	45	48	43	36	31	37	38
	Mem	34	32	28	28	27	22	20	19	20	21	22	23	23	23	20	20	18	23	33	34	34	30	29	30	29	27	24	21	26	27
S12	CPU	74	73	68	64	60	56	54	62	64	69	69	69	66	65	61	64	64	57	42	51	58	69	57	54	55	58	54	51	50	50
	Mem	7	7	7	7	7	8	8	8	9	10	11	11	10	10	9	9	9	9	8	10	11	10	9	8	8	9	8	7	7	7
S13	CPU	40	39	35	32	28	35	38	35	32	21	22	25	37	41	19	16	23	24	26	27	27	28	28	46	42	31	31	31	26	24
	Mem	5	5	4	4	3	5	6	5	5	5	5	7	8	9	5	4	5	5	5	5	6	6	6	10	9	7	7	7	7	7
S14	CPU	45	45	43	43	42	49	52	42	41	42	41	39	40	41	45	45	40	41	42	42	43	45	45	39	40	44	46	49	49	49
	Mem	22	22	22	21	19	25	27	19	19	21	20	20	21	22	24	23	20	20	21	22	24	22	22	18	20	23	24	25	22	21
S15	CPU	64	62	56	57	58	48	44	47	48	54	53	51	51	51	47	46	43	51	49	55	60	50	48	48	47	46	52	57	55	50
	Mem	22	21	22	29	35	30	28	27	27	25	29	35	35	35	29	28	28	29	27	32	36	30	27	28	28	28	28	28	19	16
S16	CPU	54	54	55	50	45	53	56	42	42	43	42	42	41	40	42	41	37	36	35	41	45	40	33	34	33	32	32	32	32	32
	Mem	24	24	27	25	23	27	26	23	23	24	24	23	23	23	23	23	23	24	25	27	26	21	23	23	22	22	22	22	22	22
S17	CPU	35	35	36	34	32	31	32	41	40	33	33	32	34	35	40	40	46	63	71	66	64	44	33	42	41	34	37	40	35	34
	Mem	9	9	9	9	9	10	10	11	10	9	10	11	10	10	9	9	12	19	27	30	32	16	9	11	11	11	11	11	10	9
S18	CPU	56	54	47	46	45	46	46	50	50	48	46	43	40	38	43	46	51	48	43	42	41	44	43	29	33	43	40	37	32	31
	Mem	38	37	33	32	31	32	33	38	39	40	37	32	31	31	35	37	38	34	28	25	23	26	27	26	25	22	25	29	26	27
S19	CPU	24	26	31	29	31	34	35	31	31	30	28	27	30	31	27	27	30	32	32	35	37	30	29	32	32	32	31	30	27	27
	Mem	12	12	13	21	30	18	14	12	11	10	11	11	19	24	13	13	23	64	14	21	26	13	11	10	11	13	12	11	11	11
S20	CPU	50	49	48	45	38	35	36	33	34	41	40	40	40	38	34	41	38	39	40	43	46	45	44	41	40	38	38	37	37	37
	Mem	27	31	42	31	19	19	19	19	19	22	20	19	22	23	21	22	23	23	23	24	25	28	29	27	26	24	24	24	24	24
S21	CPU	23	24	26	30	35	37	38	42	39	43	41	36	46	50	30	27	28	30	37	38	41	32	29	30	29	27	28	29	29	29
	Mem	14	14	13	14	14	16	17	16	16	18	19	22	22	22	17	16	13	18	14	15	15	14	14	15	13	10	12	13	12	12
S22	CPU	71	68	57	50	43	57	63	54	54	59	63	67	46	35	13	11	19	20	21	20	20	18	18	18	20	27	26	25	18	16
	Mem	16	14	9	9	10	15	17	16	16	15	19	24	13	7	4	3	5	5	6	7	7	6	6	8	8	8	7	7	4	3
S23	CPU	30	30	29	29	29	29	29	29	29	29	29	29	28	28	29	29	28	28	28	28	29	29	29	29	29	29	29	29	29	30
	Mem	13	12	12	12	12	11	11	11	11	11	11	12	11	11	11	11	11	11	11	11	12	12	12	12	12	11	11	11	12	12
S24	CPU	46	46	47	46	45	45	46	51	52	52	51	49	49	49	48	48	47	48	50	49	48	47	47	48	49	50	47	47	52	54
	Mem	12	12	12	12	12	11	11	13	13	12	12	11	11	12	12	12	14	14	13	13	13	13	13	13	13	13	12	12	12	12
S25	CPU	51	53	62	52	39	33	30	58	55	35	45	59	51	46	47	47	43	42	39	52	61	56	55	55	50	66	56	48	56	59
	Mem	19	19	21	17	12	13	13	16	15	10	13	18	19	20	17	17	15	16	17	19	20	19	18	19	18	20	19	18	17	16
S26	CPU	51	51	60	56	51	52	57	52	54	57	62	63	39	23	57	60	54	54	53	60	69	68	67	73	71	69	69	70	62	60
	Mem	8	9	13	10	7	7	8	8	9	11	12	12	8	6	8	8	7	7	7	12	15	10	9	12	11	10	10	10	8	8
S31	CPU	56	62	80	87	95	90	93	70	69	75	70	64	59	56	56	55	45	40	30	38	44	40	29	57	55	50	56	61	54	52
	Mem	6	7	10	11	13	11	11	8	9	11	10	9	9	9	7	7	5	4	4	4	4	3	3	8	8	8	8	9	8	8
S32	CPU	59	31	31	31	32	30	30	26	31	58	70	73	62	58	74	73	54	52	48	56	62	46	43	43	14	16	15	14	17	18
	Mem	8	6	6	6	5	6	6	7	7	8	14	18	16	15	17	17	12	12	11	12	12	12	11	10	4	4	5	5	7	8
S33	CPU	81	82	84	81	77	78	78	76	74	65	64	63	62	62	59	60	63	60	60	65	74	73	68	71	71	73	70	68	68	67
	Mem	26	25	23	22	21	21	20	22	22	21	21	20	22	22	20	20	20	21	22	23	25	23	22	24	25	26	25	23	22	22
S34	CPU	40	32	36	35	32	36	40	46	47	50	52	55	55	56	54	53	53	55	57	58	58	59	59	58	58	58	58	59	61	61
	Mem	22	21	21	21	22	23	24	28	29	31	34	38	38	39	35	32	35	35	36	37	38	35	35	38	38	38	38	39	40	40
S35	CPU	37	46	60	66	69	62	59	54	56	65	65	64	61	59	57	57	60	57	54	52	52	54	55	46	48	52	41	40	38	37
	Mem	25	26	25	24	22	20	20	20	21	24	24	25	19	17	19	19	18	19	19	20	20	18	18	17	19	23	18	15	16	16
S36	CPU	27	26	26	26	25	26	26	24	24	24	25	27	27	27	27	27	27	27	27	27	28	28	28	29	29	28	27	27	28	29
	Mem	4	4	5	5	5	5	5	5	5	4	4	4	5	5	4	4	5	5	4	5	5	5	5	5	4	5	5	6	6	6
S37	CPU	31	32	38	34	28	31	33	31	33	37	36	34	33	32	35	36	40	40	39	36	37	34	33	34	33	30	34	38	41	42
	Mem	5	5	6	5	3	4	4	7	7	5	5	6	5	5	5	5	5	5	5	5	5	5	5	6	6	6	6	5	7	7
S38	CPU	48	45	45	47	44	38	36	39	40	43	39	33	30	30	29	28	26	27	30	31	31	31	30	31	33	39	34	29	32	33
	Mem	16	15	14	15	15	16	16	14	14	14	14	15	13	13	11	11	13	12	12	12	12	12	12	10	11	12	12	11	11	12
S43	CPU	36	35	35	34	33	33	33	32	33	36	37	38	37	37	34	33	34	40	44	47	49	47	46	47	47	45	46	46	47	47
	Mem	19	19	18	18	18	18	17	17	18	19	20	20	20	17	17	15	19	23	25	26	25	25	24	24	25	25	26	27	27	27
S44	CPU	49	50	53	57	62	53	49	46	47	52	50	47	45	43	55	56	52	52	51	51	51	53	53	61	59	53	53	53	53	52
	Mem	33	34	34	35	37	39	40	34	35	38	40	42	42	41	31	30	36	38	41	44	46	44	43	37	37	37	38	39	39	39
S45	CPU	70	70	71	73	75	77	77	75	76	78	76	74	72	71	68	67	67	66	65	66	67	67	66	66	67	71	68	66	66	67
	Mem	31	31	32	33	33	32	31	31	31	30	32	34	32	30	30	30	30	29	27	29	30	30	30	29	30	33	31	30	30	30
S46	CPU	32	37	58	55	51	36	30	46	44	33	45	57	53	51	46	45	42	40	41	34	35	49	52	22	26	37	43	62	49	43
	Mem	8	10	15	13	11	13	14	14	13	9	11	13	13	12	11	11	12	15	13	13	14	16	14	6	8	12	12	14	11	9
S47	CPU	25	26	29	29	32	29	28	29	29	29	28	29	29	29	32	32	30	29	28	30	31	36	31	30	30	30	30	30	33	33
	Mem	17	17	15	23	19	16	15	17	17	17	18	19	17	16	18	18	16	16	17	17	16	18	18	17	18	19	16	14	17	18

Table A.12: Google cluster data benchmark scenario #8

Workload		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
S00	CPU	36	37	42	48	55	57	58	63	64	69	66	63	60	57	53	53	60	60	60	60	61	63	64	64	59	48	49	51	61	64
	Mem	10	11	13	13	13	14	14	15	15	15	15	15	16	15	13	13	13	13	12	12	12	13	14	20	19	15	15	15	22	24
S01	CPU	25	31	48	49	50	56	58	78	78	74	61	43	45	46	18	15	16	16	15	25	32	22	23	43	40	43	42	41	61	65
	Mem	18	18	21	21	20	24	25	24	24	24	22	20	20	20	17	17	22	19	13	17	20	16	17	19	20	22	18	14	24	28
S02	CPU	76	73	67	59	50	60	65	50	51	57	61	67	65	63	58	56	49	48	47	56	63	66	42	33	33	30	29	28	28	28
	Mem	28	28	26	26	25	26	26	24	24	26	28	30	30	29	23	22	24	24	23	25	27	28	26	25	24	21	21	20	20	21
S03	CPU	54	50	36	38	41	42	43	49	52	47	46	45	52	56	57	56	47	51	60	54	50	39	37	39	43	52	46	55	55	45
	Mem	12	12	13	12	12	11	11	18	21	11	19	29	20	15	17	17	13	14	15	13	12	11	11	11	11	12	12	17	14	10
S04	CPU	60	60	59	53	47	49	49	40	39	39	42	42	43	40	35	31	29	31	32	34	38	36	32	29	30	31	31	44	35	33
	Mem	33	33	32	31	29	28	28	27	27	27	27	28	29	29	27	27	26	26	26	27	28	27	27	27	27	26	26	27	27	27
S05	CPU	8	8	8	9	9	10	10	9	9	8	8	8	9	9	9	9	8	8	8	9	9	10	10	10	10	10	9	9	10	10
	Mem	6	7	7	7	7	8	8	7	7	6	7	7	7	7	7	7	7	7	7	7	7	8	8	7	8	8	7	7	5	4
S06	CPU	48	46	38	37	37	40	41	33	33	32	34	33	34	34	36	35	31	38	34	36	39	38	37	36	36	34	36	37	33	32
	Mem	26	24	20	22	24	21	20	17	17	18	21	19	18	18	20	20	17	18	19	19	20	23	24	26	26	26	25	24	21	21
S07	CPU	29	40	42	39	34	27	25	35	37	40	36	30	36	39	28	32	28	26	22	22	22	34	37	45	44	35	39	36	39	34
	Mem	13	18	21	19	17	15	14	14	15	16	16	15	17	18	12	14	14	15	17	12	8	13	15	18	17	14	16	15	16	16
S08	CPU	35	36	37	48	60	32	21	38	43	64	55	42	29	21	9	8	10	9	8	9	10	9	9	9	9	9	17	30	45	50
	Mem	7	8	10	9	8	4	3	13	14	15	13	9	8	7	3	2	3	3	3	3	2	4	5	4	4	4	10	16	17	18
S09	CPU	33	40	63	67	74	73	75	65	64	63	62	60	66	67	69	75	71	71	72	61	54	48	48	61	59	55	61	65	68	69
	Mem	16	16	16	16	16	16	21	17	15	14	14	15	17	16	15	16	14	15	16	13	12	12	13	16	16	14	15	16	16	16
S10	CPU	63	56	46	45	44	35	34	38	37	35	39	46	47	48	29	26	18	22	29	21	17	19	20	31	26	13	16	18	16	15
	Mem	20	18	16	16	16	13	12	12	12	11	12	14	15	15	12	11	8	10	13	10	8	9	9	12	11	6	7	8	8	7
S11	CPU	22	22	24	23	22	23	23	20	19	19	20	25	25	22	23	23	24	33	29	24	23	23	23	24	24	23	18	13	12	12
	Mem	8	8	8	7	7	9	9	8	8	8	8	9	9	9	8	8	11	12	11	9	8	8	8	8	8	8	8	8	7	7
S12	CPU	40	39	28	27	26	21	18	22	24	29	26	23	25	27	41	41	32	29	24	34	41	34	33	29	33	41	36	32	28	27
	Mem	14	13	9	9	10	9	9	11	11	12	11	10	11	12	14	14	10	11	12	14	15	12	11	11	12	14	13	11	10	10
S13	CPU	53	50	43	43	42	42	42	38	37	37	37	37	39	41	39	38	36	37	39	40	41	43	36	54	50	35	33	32	32	31
	Mem	13	11	10	8	7	6	6	7	7	6	6	6	6	6	6	6	6	6	6	7	7	8	8	12	12	11	9	7	6	5
S14	CPU	15	9	18	12	6	10	14	12	12	11	15	21	22	22	20	13	22	23	25	18	14	20	20	8	10	15	21	10	14	16
	Mem	4	2	3	2	1	2	3	15	13	2	3	5	5	5	4	7	6	6	5	3	2	4	4	3	3	2	5	2	3	4
S15	CPU	62	62	62	55	47	54	57	38	40	48	53	60	60	60	53	52	48	49	51	56	60	58	48	46	45	43	46	48	46	45
	Mem	19	19	21	19	17	17	17	14	15	16	19	22	20	19	17	16	14	15	17	20	21	19	17	15	15	13	16	18	16	15
S16	CPU	63	65	70	68	67	44	34	50	54	67	63	57	47	42	57	61	77	70	58	60	61	54	52	45	47	50	60	69	55	51
	Mem	15	16	23	21	19	9	5	12	13	15	15	16	10	6	17	18	18	17	15	15	16	12	12	10	11	14	14	14	13	13
S17	CPU	29	29	29	27	25	25	25	25	25	22	24	24	26	26	24	23	24	29	31	32	32	27	26	27	26	24	25	26	26	26
	Mem	10	9	8	9	8	8	8	10	8	8	8	8	8	9	9	9	8	9	10	11	12	9	9	9	9	8	9	9	9	9
S18	CPU	35	35	33	36	39	39	41	52	56	62	60	55	53	57	57	57	54	58	64	57	52	60	61	54	55	57	54	51	55	56
	Mem	2	2	2	1	1	2	2	3	5	7	7	6	6	7	6	6	7	7	7	6	5	6	6	5	5	5	5	5	5	5
S19	CPU	42	37	22	22	21	32	36	32	34	45	40	33	45	52	33	32	45	43	40	41	42	32	30	40	33	23	33	42	35	32
	Mem	9	8	5	5	6	8	9	9	9	11	11	10	11	12	10	10	9	9	10	11	11	9	9	12	10	8	9	10	9	8
S20	CPU	44	43	42	40	39	43	45	43	40	29	31	34	41	45	39	38	37	38	39	40	41	43	44	45	46	49	47	44	39	36
	Mem	20	19	17	15	13	15	15	14	13	10	10	11	14	16	12	12	13	13	15	15	15	18	18	20	21	23	20	17	13	11
S21	CPU	24	23	24	25	26	26	26	29	29	29	28	25	26	26	25	25	27	27	26	25	25	23	22	27	26	25	26	27	30	30
	Mem	17	17	16	16	16	16	16	17	17	14	14	14	16	16	16	16	17	17	17	16	16	16	15	16	16	15	15	15	16	16
S22	CPU	54	53	48	51	54	55	56	55	54	53	48	41	41	40	41	42	49	54	63	61	62	65	61	52	49	44	48	53	50	49
	Mem	12	11	10	10	10	11	12	11	11	10	9	7	8	8	7	7	8	11	15	17	29	19	18	14	13	12	16	19	13	11
S23	CPU	49	47	38	40	40	40	39	42	42	42	44	48	50	51	46	45	41	42	45	42	41	46	52	42	45	52	52	52	49	48
	Mem	46	44	38	37	37	36	35	41	41	38	40	43	42	41	39	39	40	39	36	36	36	45	46	43	44	45	44	43	39	37
S24	CPU	9	9	9	10	11	11	12	11	11	10	10	10	10	11	11	11	10	10	9	10	11	12	12	12	11	11	11	10	11	11
	Mem	9	9	10	10	10	11	11	10	10	10	10	9	10	10	11	11	10	10	9	10	11	11	11	11	11	11	11	11	11	11
S25	CPU	34	35	35	34	32	35	51	39	36	28	30	33	33	34	37	37	34	32	28	30	32	34	34	30	30	30	31	34	30	27
	Mem	8	8	9	8	7	10	15	11	10	9	10	12	12	13	13	12	12	12	11	11	11	11	10	10	10	12	12	10	9	
S26	CPU	56	56	56	50	44	54	60	51	50	48	49	50	45	43	48	48	43	39	32	46	55	52	24	20	21	23	20	35	25	21
	Mem	11	11	9	10	11	12	13	14	13	9	9	12	8	7	9	9	10	9	8	11	13	13	10	7	7	7	6	9	7	6
S31	CPU	60	59	56	52	49	54	56	34	33	34	34	35	35	34	41	40	31	32	33	35	33	37	40	34	34	34	33	41	35	33
	Mem	20	19	17	18	19	17	16	13	14	15	16	20	21	17	17	16	14	14	14	15	15	15	15	15	15	15	15	17	15	14
S32	CPU	55	54	54	51	54	49	48	37	37	35	44	57	43	34	46	48	48	44	37	46	53	51	51	53	53	54	45	37	41	42
	Mem	40	38	34	32	33	30	29	28	26	19	33	52	30	18	30	32	32	28	21	33	41	35	34	39	39	38	28	20	32	36
S33	CPU	50	53	63	68	74	86	92	78	76	58	61	58	77	87	91	88	65	53	32	31	30	36	26	40	46	60	73	85	72	67
	Mem	14	13	11	15	19	15	14	13	13	10	11	13	14	14	11	10	7	7	8	11	13	10	9	12	11	9	10	10	8	8
S34	CPU	38	36	31	28	25	22	20	30	31	32	35	40	43	44	28	25	21	23	26	29	31	29	29	35	32	28	29	41	33	30
	Mem	20	21	24	23	22	24	25	20	20	19	20	21	23	25	25	25	20	18	15	19	22	17	15	18	18	21	23	25	21	20
S35	CPU	44	43	44	44	45	44	44	43	43	47	52	47	51	54	43	41	38	40	44	50	55	53	49	46	45	40	42	44	46	46
	Mem	27	26	24	23	23	20	19	23	24	29	29	29	31	32	31	30	21	22	23	27	30	27	26	25	25	25	26	31	27	25
S36	CPU	25	21	22	23	27	18	15	25	27	33	29	23	25	24	22	21	32	31	28	33	31	24	22	19	20	26	21	16	16	16
	Mem	50	50	51	53	56	55	54	52	52	55	56	53	59	62	48	46	40	42	47	50	54	51	49	48	46	41	45	50	54	55
S37	CPU	39	38	34	37	38	37	36	38	38	39	43	44	44	43	42	42	54	63	75	71	69	45	38	37	35	34	41	43	42	42
	Mem	10	10	10	10	10	8	8	20	19	12	11	10	11	11	10	10	12	12	11	12	11	11	11	8	8	10	8	7	7	6
S38	CPU	32	33	41	35	33	42	46	31	31	33	36	40	44	46	38	36	30	31	33	36	38	28	27	30	29	27	36	43	31	28
	Mem	19	19	16	17	18	18	17	19	19	19	20	21	20	20	20	21	27	32	39	41	42	24	19	19	18	16	19	21	19	18
S43	CPU	73	74	74	68	60	67	70	58	52	45	42	38	41	43	50	50	41	40	39	42	48	49	43	42	43	44	41	39	41	42
	Mem	8	8	10	7	7	11	12	8	8	11	12	14	13	12	10	9	8	9	10	11	11	7	7	9	9	10	9	8	7	7
S44	CPU	21	20	23	29	30	25	40	28	25	18	29	30	30	29	26	27	34	33	29	23	18	29	30	21	25	34	30	27	21	20
	Mem	27	28	30	27	25	27	28	28	27	26	25	24	22	21	24	24	23	24	25	28	31	31	30	26	26	27	28	28	26	26
S45	CPU	54	55	57	55	54	64	68	69	52	65	75	88	89	89	60	52	23	26	37	42	48	34	26	25	25	24	25	39	31	27
	Mem	9	9	9	10	10	11	14	11	10	8	10	12	11	11	10	10	12	12	12	10	8	11	12	9	10	13	12	11	10	10
S46	CPU	61	61	62	57	51	59	62	51	51	52	51	50	49	50	54	52	33	34	53	58	62	37	30	28	26	28	27	35	31	30
	Mem	27	27	27	26	25	30	32	32	29	25	30	36	35	35	29	27	22	23	26	25	25	23	21	20	19	18	18	21	19	18
S47	CPU	25	26	29	29	32	29	28	29	29	29	28	29	29	29	32	32	30	29	28	30	31	36	31	30	30	30	30	30	33	33
	Mem	4	4	5	4	4	5	5	3	3	4	4	4	4	4	4	4	3	3	3	4	58	3	2	1	1	1	1	2	2	1

Table A.13: Google cluster data benchmark scenario #9

Workload		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
S00	CPU	23	27	45	40	29	24	22	27	24	23	24	24	23	26	22	22	26	28	32	27	23	22	22	26	28	32	25	20	26	27
	Mem	18	17	15	12	8	9	7	11	10	7	7	8	8	9	7	7	9	9	10	11	13	7	6	13	12	9	8	6	26	32
S01	CPU	37	44	64	64	63	46	39	55	59	71	73	77	78	78	59	58	64	72	78	72	69	49	47	81	78	72	65	59	59	59
	Mem	22	23	28	27	26	20	18	31	30	23	25	26	28	29	22	21	22	26	31	27	25	20	19	20	22	25	23	21	20	20
S02	CPU	54	58	69	52	33	52	60	38	37	41	39	36	39	41	51	53	51	46	36	45	51	36	22	13	14	17	18	19	10	7
	Mem	15	14	10	9	9	9	8	4	4	6	6	7	5	5	11	11	8	7	6	7	7	6	5	5	5	3	6	8	3	1
S03	CPU	27	32	29	36	46	45	45	37	37	40	31	18	43	49	23	18	12	12	13	15	16	29	32	41	46	52	36	28	49	55
	Mem	9	11	8	10	12	12	11	9	9	8	7	4	13	13	7	6	7	6	3	4	5	12	13	12	14	16	19	24	13	9
S04	CPU	30	29	28	26	25	28	29	25	25	28	31	29	33	35	28	27	25	28	33	33	34	30	30	35	40	35	31	28	28	28
	Mem	24	24	24	22	21	21	21	20	21	23	24	23	26	28	23	22	21	23	28	28	29	27	27	29	28	25	25	24	24	24
S05	CPU	42	39	31	33	35	38	39	39	39	39	37	34	33	33	33	33	32	34	36	38	38	42	43	43	43	47	43	41	35	33
	Mem	20	19	16	16	16	17	17	18	18	17	17	16	16	15	16	16	15	16	16	17	18	18	19	20	22	25	22	20	17	16
S06	CPU	22	22	28	23	24	24	24	19	20	24	24	24	24	24	27	27	24	23	22	25	24	26	27	38	34	24	19	30	18	14
	Mem	16	15	16	14	14	15	15	14	15	16	16	16	16	15	17	17	16	16	16	17	16	17	18	20	19	15	15	16	16	16
S07	CPU	45	43	35	34	33	33	33	35	35	32	33	35	35	35	33	34	36	36	36	33	31	37	38	38	39	42	39	35	26	23
	Mem	39	37	28	27	26	30	31	34	32	25	26	26	30	32	31	32	35	33	29	32	35	37	38	36	36	36	35	31	29	24
S08	CPU	49	49	52	53	54	57	69	53	51	45	49	50	49	51	56	47	23	27	35	30	26	23	27	27	36	53	36	34	37	38
	Mem	19	20	27	28	18	20	17	17	17	17	23	24	19	17	26	23	12	13	14	13	12	13	14	10	13	21	16	15	15	15
S09	CPU	36	36	37	42	47	46	45	38	41	57	54	51	51	52	33	31	33	40	52	54	55	41	40	33	32	30	34	38	37	37
	Mem	19	20	21	21	22	19	18	20	21	25	24	24	24	24	21	20	18	21	25	27	28	22	21	23	23	21	22	23	22	22
S10	CPU	37	43	50	36	21	27	31	41	41	36	37	38	44	26	32	33	41	38	31	31	39	51	52	49	49	49	47	46	34	30
	Mem	17	19	20	15	8	10	11	16	17	17	16	15	19	10	11	13	17	14	12	12	17	24	25	22	21	18	18	18	13	11
S11	CPU	30	28	25	25	25	24	24	24	24	25	25	25	25	26	25	25	24	25	27	28	29	29	30	31	30	27	27	27	27	27
	Mem	29	26	23	23	23	23	23	23	23	23	23	24	25	26	25	24	22	23	24	26	27	29	29	29	28	24	24	25	25	25
S12	CPU	35	37	34	35	36	33	32	30	30	34	32	30	30	30	30	31	35	34	34	36	38	35	33	32	32	34	35	43	51	51
	Mem	18	25	45	34	19	30	34	26	25	26	24	23	25	27	20	20	23	23	23	26	28	31	30	22	22	22	24	27	24	23
S13	CPU	71	71	69	69	69	72	73	68	69	73	69	70	74	77	73	79	79	77	74	75	76	70	69	78	77	75	75	76	73	72
	Mem	17	17	17	18	19	18	18	16	17	19	17	17	19	20	19	21	20	20	19	19	18	17	17	21	20	19	20	20	18	17
S14	CPU	47	52	67	62	57	64	67	41	37	40	29	17	24	22	12	10	8	9	19	23	25	28	29	39	41	45	41	37	37	37
	Mem	10	12	16	15	13	18	19	15	14	13	11	9	13	8	8	8	6	6	13	17	10	14	14	14	14	14	14	13	13	13
S15	CPU	29	29	32	32	32	32	32	31	30	29	31	32	32	31	31	31	29	29	30	31	32	32	32	32	32	32	32	31	23	20
	Mem	9	9	9	9	9	9	9	9	9	12	22	9	9	9	9	9	10	9	9	9	9	9	9	9	9	9	9	9	8	7
S16	CPU	81	81	83	85	87	86	86	87	88	90	89	88	88	88	88	88	88	87	84	82	80	65	63	78	80	86	83	79	73	72
	Mem	33	34	35	35	35	34	34	37	36	30	32	35	38	39	38	37	34	34	35	37	39	30	28	32	34	37	38	38	35	34
S17	CPU	39	39	38	39	40	41	42	39	41	40	38	37	36	35	35	35	35	37	41	36	32	29	28	33	35	31	33	36	39	40
	Mem	14	15	16	15	14	12	11	18	18	13	13	14	19	22	16	15	13	13	15	17	19	12	10	12	14	16	17	18	17	17
S18	CPU	42	38	24	28	32	47	43	64	62	41	45	51	52	45	42	39	21	25	33	20	10	28	31	24	24	44	41	39	20	13
	Mem	5	5	3	3	3	6	5	8	8	10	9	8	8	7	6	6	2	3	5	8	10	4	4	5	3	4	4	3	3	2
S19	CPU	57	58	62	66	71	67	66	54	56	70	72	75	77	77	78	77	71	67	60	55	51	49	49	63	64	68	71	74	76	77
	Mem	51	48	36	36	35	35	35	35	35	35	36	37	38	38	36	35	31	34	40	41	42	36	34	38	38	37	36	35	37	37
S20	CPU	66	66	63	67	72	73	74	66	58	60	57	53	60	64	56	53	43	43	45	48	55	52	49	51	55	62	60	58	68	71
	Mem	34	34	33	33	32	33	33	30	29	30	29	29	31	32	26	25	21	22	24	29	33	32	31	32	33	37	38	38	37	36
S21	CPU	20	19	22	33	46	50	53	42	45	62	65	69	75	78	72	69	51	45	34	37	40	39	40	57	53	45	42	38	33	31
	Mem	5	4	6	12	18	20	20	19	19	20	18	14	15	16	17	17	17	19	25	21	19	16	16	23	22	19	19	19	17	17
S22	CPU	51	51	50	51	52	50	49	52	52	52	52	53	30	17	50	50	17	31	55	34	19	48	52	20	20	19	38	55	56	56
	Mem	13	13	14	13	13	11	11	14	14	14	14	14	12	11	15	15	10	12	16	13	11	14	14	12	11	10	13	15	15	15
S23	CPU	19	20	24	23	21	22	21	17	18	23	27	40	51	56	50	51	20	32	34	25	21	16	16	26	24	19	20	21	16	13
	Mem	12	12	13	12	10	10	9	10	10	12	14	19	30	37	34	32	11	15	18	16	15	13	13	15	14	11	13	15	10	8
S24	CPU	77	74	63	58	53	46	43	54	55	57	58	61	70	76	78	77	72	65	53	63	71	70	70	70	67	61	63	66	55	52
	Mem	39	36	26	24	22	23	24	28	29	33	35	38	41	43	35	34	34	33	31	33	34	35	35	34	36	39	39	39	32	30
S25	CPU	6	5	6	5	4	6	7	7	7	5	6	8	8	8	5	4	6	6	14	17	14	20	18	16	16	15	16	17	16	15
	Mem	1	1	2	1	0	2	2	1	1	1	2	2	2	1	1	1	1	1	3	5	5	6	4	4	4	4	4	4	4	4
S26	CPU	38	37	36	39	42	39	38	42	43	42	44	48	47	50	48	48	43	44	45	49	52	43	42	42	44	48	40	32	30	29
	Mem	10	10	10	11	12	11	11	10	11	12	12	11	12	12	13	14	16	14	12	12	12	12	12	12	12	12	11	10	9	9
S31	CPU	27	27	27	25	22	22	21	21	21	25	26	26	28	30	24	23	21	22	24	29	36	30	28	26	26	24	25	26	24	24
	Mem	20	19	18	17	16	15	15	16	16	17	18	19	20	18	17	17	17	18	21	25	22	21	20	19	17	19	20	18	17	
S32	CPU	89	89	88	85	80	82	83	67	64	60	61	64	65	66	40	41	85	72	59	57	55	56	57	56	57	60	59	58	59	59
	Mem	28	29	29	27	24	24	24	22	21	20	19	19	21	22	14	15	27	25	21	32	19	19	19	18	18	19	20	20	20	20
S33	CPU	29	28	27	23	20	35	41	22	21	21	22	19	20	22	23	23	26	28	39	38	37	55	44	45	42	37	36	35	36	36
	Mem	13	11	5	5	6	9	10	8	8	7	7	6	7	8	8	8	10	10	12	13	17	15	13	13	15	19	16	12	10	9
S34	CPU	44	47	47	39	32	38	47	36	40	53	44	31	33	37	25	27	49	44	36	51	61	27	18	23	21	18	21	24	14	11
	Mem	13	13	7	6	5	6	9	7	8	11	9	6	6	7	5	5	11	10	8	12	14	7	5	9	7	6	6	6	3	2
S35	CPU	27	28	31	29	27	37	40	41	39	34	31	26	35	40	35	36	44	42	40	43	45	36	36	57	49	28	33	38	28	24
	Mem	8	8	8	8	7	9	10	9	8	29	27	6	9	11	7	8	11	11	10	11	11	7	7	14	12	6	7	8	7	7
S36	CPU	41	40	40	42	44	39	35	36	36	38	37	36	35	35	37	37	40	42	46	44	43	33	30	32	33	33	38	42	41	40
	Mem	14	13	11	12	12	11	10	10	11	15	14	13	13	13	14	15	16	16	15	14	13	13	14	13	11	13	15	14	14	
S37	CPU	57	56	55	56	58	58	58	66	66	60	61	62	64	65	66	66	62	66	73	66	61	65	63	66	62	58	57	56	53	52
	Mem	25	25	24	24	24	24	23	24	24	24	26	27	28	28	28	27	26	27	29	28	28	28	27	28	27	25	23	21	21	22
S38	CPU	32	32	32	32	33	31	31	34	35	36	35	34	40	44	37	36	37	38	39	39	36	35	35	51	46	36	36	35	36	36
	Mem	9	8	6	8	10	10	10	10	10	10	10	9	10	11	9	9	14	14	13	12	10	9	9	13	13	12	11	11	11	11
S43	CPU	18	21	45	46	45	43	42	46	47	51	50	49	50	48	33	29	14	14	13	12	11	16	18	38	35	27	37	55	52	50
	Mem	5	4	8	8	9	8	8	8	8	7	7	8	7	7	4	4	5	4	3	3	3	4	4	7	6	6	7	9	8	8
S44	CPU	44	43	40	43	47	40	40	44	44	45	40	34	34	33	40	42	47	44	41	48	53	60	58	49	42	38	35	45	38	34
	Mem	23	23	23	26	30	24	21	23	23	24	24	24	23	23	29	29	23	25	30	32	33	37	37	34	26	24	48	71	34	22
S45	CPU	54	54	55	54	53	51	50	55	55	51	36	15	49	68	59	58	56	63	63	67	71	69	70	71	69	66	66	66	64	63
	Mem	14	14	15	13	11	12	12	15	15	12	10	6	9	11	9	9	11	12	11	12	12	11	11	13	14	17	16	15	13	12
S46	CPU	41	42	60	73	85	85	85	59	56	50	55	62	78	87	84	79	44	37	25	30	33	45	49	56	55	53	58	62	81	87
	Mem	29	29	34	34	34	35	35	29	29	30	32	36	38	38	41	40	26	27	28	29	29	32	33	34	34	34	34	35	35	36
S47	CPU	60	60	60	56	53	62	66	62	64	66	62	55	54	52	52	51	52	55	48	44	51	53	37	35	34	31	29	41	33	29
	Mem	13	12	11	13	12	14	15	15	15	14	13	13	11	10	11	11	8	8	9	10	11	11	10	8	9	10	9	11	9	8

Table A.14: Google cluster data benchmark scenario #10

Workload		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
S00	CPU	46	47	44	43	43	42	42	48	49	50	51	53	51	50	50	51	47	48	47	47	48	48	48	48	50	54	53	51	52	52
	Mem	19	17	15	18	20	19	18	23	23	21	22	21	20	20	22	21	23	23	19	19	20	19	19	22	22	22	22	23	21	20
S01	CPU	36	38	45	44	43	43	43	39	39	41	42	43	46	48	43	42	38	40	45	50	54	52	51	46	44	38	44	49	49	49
	Mem	19	19	22	21	19	19	18	16	16	17	18	20	21	22	19	18	16	17	20	23	24	27	27	23	21	16	18	20	21	21
S02	CPU	74	74	74	65	56	63	66	40	36	41	41	41	41	39	41	40	33	21	0	0	0	0	0	0	2	9	12	30	30	30
	Mem	25	25	25	23	20	20	20	15	14	15	16	18	15	13	15	14	10	8	0	0	0	0	0	0	2	7	4	5	6	6
S03	CPU	40	42	49	44	38	39	40	43	42	37	43	52	45	41	52	54	58	55	50	52	54	48	46	38	44	57	45	35	44	47
	Mem	4	4	5	6	7	4	3	3	3	2	3	4	4	3	7	7	8	6	4	4	4	3	3	2	5	9	5	2	3	3
S04	CPU	24	25	26	26	25	24	23	27	27	27	29	31	26	24	29	29	26	25	24	34	37	36	37	37	37	37	32	28	26	25
	Mem	9	10	11	10	9	10	10	10	10	10	11	13	11	10	11	11	9	9	9	12	13	13	14	14	13	12	12	12	11	11
S05	CPU	19	20	22	21	20	22	22	13	13	16	18	20	17	15	20	20	12	11	9	12	11	9	9	15	14	13	19	24	15	14
	Mem	6	6	6	5	5	5	6	4	4	5	4	4	4	4	5	6	9	7	4	4	4	3	3	4	4	4	5	7	4	4
S06	CPU	46	42	42	44	47	37	33	47	47	44	44	44	43	42	38	37	34	32	29	35	39	43	44	42	40	38	43	48	44	42
	Mem	17	15	13	14	16	12	11	15	15	16	16	16	16	15	13	12	11	11	11	14	16	16	16	16	15	14	17	19	16	15
S07	CPU	38	36	42	46	51	47	45	35	38	46	62	86	88	88	44	35	14	14	15	17	17	24	27	37	42	55	54	53	39	36
	Mem	19	18	20	21	22	19	17	15	16	15	20	26	26	26	16	14	12	11	10	11	12	17	18	18	20	24	23	23	18	33
S08	CPU	43	41	41	40	38	45	50	49	52	57	60	64	58	55	49	49	54	55	55	57	58	73	74	55	52	46	56	65	59	58
	Mem	3	3	2	4	4	9	12	7	25	8	10	14	12	12	11	11	9	10	12	16	22	17	16	11	10	7	9	11	11	10
S09	CPU	17	18	19	20	19	21	21	19	20	23	23	21	20	19	24	20	18	18	19	18	18	20	20	18	18	19	19	20	19	19
	Mem	7	8	9	7	7	7	7	9	9	9	10	8	8	8	8	6	8	8	7	8	8	6	7	9	8	7	8	9	8	8
S10	CPU	89	88	86	83	79	82	84	71	68	68	72	78	78	77	72	71	69	67	68	73	81	86	72	66	67	68	65	63	62	61
	Mem	28	28	27	25	23	22	22	22	22	21	22	23	23	22	22	22	22	21	21	22	23	24	22	19	20	23	21	19	18	18
S11	CPU	49	51	58	60	62	61	60	56	57	62	65	68	65	63	62	62	65	66	67	65	64	57	56	61	60	59	59	59	69	72
	Mem	27	29	34	33	32	33	33	30	29	28	28	29	29	29	30	30	31	29	27	31	34	33	33	27	28	28	28	27	26	26
S12	CPU	35	34	35	38	40	37	36	36	36	39	39	39	38	30	29	27	13	19	29	29	30	29	30	31	26	14	23	30	18	14
	Mem	21	20	18	18	15	17	17	18	19	23	26	30	27	25	21	19	7	12	22	24	26	22	22	26	21	9	17	24	11	7
S13	CPU	18	18	19	20	22	30	31	31	29	21	28	29	25	25	20	19	18	26	29	31	32	31	31	32	31	30	31	31	31	31
	Mem	15	15	16	15	15	14	16	17	17	16	16	16	14	15	15	15	16	18	19	20	20	20	22	20	16	18	21	20	19	
S14	CPU	20	20	20	19	19	19	19	19	19	19	20	21	20	20	20	20	21	21	21	22	22	22	23	33	30	20	20	23	36	38
	Mem	15	15	15	14	14	14	14	14	14	14	14	14	14	16	15	13	14	15	16	16	15	15	16	16	15	14	14	15	15	
S15	CPU	61	61	60	57	54	61	64	48	50	55	51	48	47	47	48	47	45	45	54	61	64	64	54	53	54	54	45	37	36	35
	Mem	11	11	11	12	12	13	14	11	12	12	11	11	10	10	10	10	9	10	13	15	16	16	15	15	15	16	17	19	13	11
S16	CPU	25	24	23	24	25	26	27	22	21	18	51	50	49	48	39	38	38	37	37	43	46	45	45	46	45	42	45	46	48	47
	Mem	16	14	6	7	7	6	6	10	10	11	15	13	12	11	18	17	10	13	20	17	14	19	20	17	16	13	14	14	13	12
S17	CPU	60	59	54	56	57	58	58	52	53	56	50	43	45	47	49	50	50	53	59	62	65	55	45	32	36	44	40	36	43	45
	Mem	8	7	5	6	7	7	7	6	6	9	7	5	5	5	6	6	6	7	9	8	7	5	5	5	5	5	5	4	3	6
S18	CPU	78	81	91	93	95	95	95	93	92	90	87	85	90	91	94	94	92	90	89	79	72	75	58	73	71	65	67	70	75	76
	Mem	22	24	29	27	25	24	23	23	23	27	27	28	27	27	26	26	19	19	20	26	30	22	19	22	21	19	19	19	19	20
S19	CPU	58	60	68	72	75	72	71	59	61	71	73	77	86	88	86	85	80	71	54	55	56	52	51	44	48	58	56	55	75	81
	Mem	32	33	36	33	31	27	25	23	23	23	24	24	28	31	27	26	23	24	24	26	27	24	23	19	20	21	23	25	27	27
S20	CPU	40	40	40	39	39	50	51	47	45	39	39	39	40	39	39	40	40	40	39	43	42	41	41	42	42	42	46	45	42	41
	Mem	21	22	24	24	24	25	25	24	24	21	22	22	23	24	24	24	25	24	23	25	25	24	24	26	27	30	30	30	28	28
S21	CPU	47	47	49	49	50	45	43	42	42	41	42	44	43	43	38	36	22	22	23	23	24	23	23	31	29	21	20	19	18	17
	Mem	6	6	7	6	6	6	6	6	6	5	5	6	6	6	6	6	5	5	6	6	6	5	5	7	7	5	5	5	4	4
S22	CPU	48	49	50	55	61	54	51	47	48	55	52	47	51	53	58	59	56	57	58	56	54	52	53	65	61	51	58	67	62	60
	Mem	6	6	6	6	7	6	5	4	4	5	5	5	5	5	5	5	6	6	7	6	5	6	6	8	7	5	6	7	6	6
S23	CPU	60	58	52	41	28	23	21	24	25	31	35	42	37	33	24	26	33	32	32	29	25	30	32	41	34	16	19	21	36	38
	Mem	31	29	25	17	7	7	7	6	6	9	10	12	11	10	9	8	10	10	10	9	7	9	9	9	8	5	5	5	9	8
S24	CPU	50	55	70	70	69	83	91	92	86	66	69	73	75	75	70	69	62	61	64	68	67	60	59	71	72	74	76	77	78	78
	Mem	17	20	30	23	15	19	21	19	19	15	16	17	18	19	17	16	13	14	17	21	23	16	15	17	17	19	20	21	18	17
S25	CPU	15	15	15	15	14	14	14	14	14	15	15	15	17	16	17	17	16	19	16	16	16	15	15	15	15	15	14	15	23	
	Mem	13	14	17	15	12	14	15	13	14	15	14	12	13	13	12	12	12	11	9	11	12	12	12	14	14	14	13	12	13	16
S26	CPU	60	58	55	53	50	55	57	48	47	46	49	54	53	52	44	46	45	46	47	47	47	49	41	35	35	39	37	35	39	40
	Mem	12	12	11	11	11	11	12	14	15	18	17	14	14	14	14	14	12	14	17	15	14	14	13	13	13	13	12	11	12	13
S31	CPU	76	73	64	52	38	39	40	30	30	36	35	33	34	35	33	33	36	37	39	36	34	37	37	39	39	40	39	38	43	44
	Mem	75	73	65	54	42	41	40	36	36	38	38	37	39	40	37	37	35	37	40	39	38	43	44	47	45	41	41	41	44	45
S32	CPU	39	38	28	27	25	20	19	36	36	31	27	22	20	20	31	32	26	24	20	17	14	19	20	21	27	41	34	27	27	27
	Mem	7	9	13	10	8	6	6	9	9	8	7	5	5	5	9	10	12	11	8	7	6	5	5	8	8	10	9	7	8	9
S33	CPU	32	32	32	31	30	32	33	32	32	31	32	33	32	32	34	34	36	31	33	33	33	30	30	33	33	33	29	26	24	24
	Mem	12	12	12	12	11	12	12	11	11	11	11	11	11	10	11	11	12	11	11	11	11	10	10	11	11	10	10	10	8	8
S34	CPU	42	41	26	19	12	35	51	47	43	30	28	24	38	45	39	38	34	30	22	33	40	30	28	27	34	53	37	25	25	25
	Mem	4	5	1	2	2	4	8	6	5	2	2	1	4	5	4	4	5	4	2	17	27	7	3	2	3	4	3	2	3	3
S35	CPU	22	22	24	23	23	14	16	23	23	21	18	14	19	22	21	21	21	29	41	29	21	20	21	35	32	24	29	33	17	12
	Mem	4	5	5	5	4	3	4	4	5	7	6	4	5	6	4	4	5	6	8	7	8	7	5	7	7	5	6	7	4	4
S36	CPU	34	33	41	45	36	21	20	15	18	27	26	26	24	22	22	22	18	24	24	22	20	29	31	33	29	22	20	18	18	18
	Mem	7	7	9	9	8	6	6	5	5	4	5	6	6	6	4	4	5	7	9	8	8	9	9	10	8	6	6	6	6	6
S37	CPU	13	13	17	13	13	13	13	13	13	13	13	13	14	14	13	13	13	14	14	16	15	14	14	13	13	13	13	13	13	13
	Mem	3	3	4	3	3	3	2	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	2	2	2	3	3
S38	CPU	28	33	53	57	56	75	83	85	81	58	63	70	82	88	82	77	44	36	22	31	36	28	28	54	54	55	66	75	83	87
	Mem	14	15	21	21	20	21	22	26	26	21	24	29	28	28	25	24	20	19	17	17	17	14	14	20	19	18	22	26	24	24
S43	CPU	8	8	17	19	17	11	9	8	8	9	9	8	8	8	10	10	9	9	9	9	9	11	11	10	11	12	11	10	9	8
	Mem	2	2	3	3	3	2	2	1	1	1	1	1	2	2	2	2	1	2	2	2	2	3	3	4	4	4	3	2	2	2
S44	CPU	46	43	40	39	38	44	49	57	58	60	59	58	60	61	61	61	59	59	58	58	57	59	61	63	62	57	54	52	42	39
	Mem	15	14	12	11	9	11	12	13	14	15	15	15	16	17	16	16	15	14	12	13	13	15	16	25	26	24	21	19	15	13
S45	CPU	25	25	25	26	27	25	24	24	24	23	24	25	25	25	24	23	22	23	26	28	29	26	25	24	23	20	26	31	28	27
	Mem	15	16	19	18	16	16	16	15	15	15	16	17	16	16	16	16	15	16	19	19	19	18	17	16	16	14	15	16	18	18
S46	CPU	44	43	40	37	33	25	21	23	24	26	26	29	28	29	32	33	34	33	31	38	40	40	39	15	20	31	36	39	37	35
	Mem	13	14	14	12	10	9	9	12	12	12	11	10	13	15	13	13	14	13	11	12	12	10	9	9	11	14	15	15	15	15
S47	CPU	48	48	45	44	42	43	43	47	47	46	45	45	48	49	46	46	48	47	45	47	48	47	47	48	48	47	47	46	46	46
	Mem	11	11	11	11	11	11	10	10	10	9	10	11	12	13	12	12	12	12	11	12	12	13	13	11	11	11	11	11	11	12

Table A.15: Google cluster data benchmark scenario #11

Workload		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
S00	CPU	91	90	84	71	57	56	55	56	55	49	56	65	35	18	45	49	51	49	36	32	29	34	33	15	14	13	11	16	29	3
	Mem	20	20	19	15	10	10	9	9	9	8	10	12	6	2	7	8	10	9	5	4	3	4	4	2	3	3	3	4	7	2
S01	CPU	26	26	25	24	25	28	40	37	28	16	16	16	24	23	21	20	16	19	28	27	28	22	24	29	29	25	20	29	19	15
	Mem	10	10	11	9	7	9	12	11	9	7	7	8	8	8	6	7	16	13	9	12	14	24	26	19	16	7	6	7	12	14
S02	CPU	37	41	49	47	49	54	55	53	39	35	36	38	40	41	36	35	31	33	36	38	40	36	29	23	23	21	25	43	31	27
	Mem	15	16	19	17	16	17	18	17	14	13	14	16	18	18	15	15	12	13	15	16	17	14	13	12	12	10	13	18	14	13
S03	CPU	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	Mem	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
S04	CPU	63	61	56	53	48	47	47	45	46	53	53	55	62	63	61	60	57	55	52	51	51	53	54	64	60	49	50	52	51	50
	Mem	28	27	26	25	24	42	50	22	21	22	24	28	29	29	25	25	22	21	18	21	24	21	21	25	24	22	21	20	21	22
S05	CPU	56	57	58	56	54	48	45	58	57	51	55	59	57	56	55	55	57	58	61	60	60	54	53	59	58	56	56	55	52	51
	Mem	45	44	42	40	38	34	33	35	36	37	37	38	41	43	36	35	38	39	41	43	45	43	43	46	44	38	40	42	38	36
S06	CPU	93	93	92	94	96	94	95	95	94	92	90	86	82	80	89	90	89	91	95	96	93	93	93	93	90	90	90	89	89	89
	Mem	27	27	27	26	26	35	29	25	25	26	25	25	21	19	22	22	22	23	24	23	23	27	27	24	23	23	24	24	26	26
S07	CPU	33	31	26	29	33	26	23	27	29	40	40	39	31	26	20	19	17	18	21	18	14	34	39	48	45	37	33	29	22	20
	Mem	27	25	19	18	18	16	15	14	15	18	18	18	18	18	15	14	11	12	13	12	11	24	28	33	31	26	23	21	16	14
S08	CPU	11	11	11	11	10	13	14	10	10	10	10	10	11	11	10	10	10	10	11	11	11	10	10	9	10	11	11	12	11	11
	Mem	10	10	9	9	9	6	5	7	7	6	7	7	6	5	6	6	7	7	7	7	7	7	7	6	7	7	7	7	6	5
S09	CPU	49	50	48	47	46	45	45	39	36	40	41	42	41	40	37	37	38	40	44	46	46	44	45	57	54	45	46	48	45	44
	Mem	32	32	29	27	24	24	24	25	26	31	33	36	33	31	26	26	29	30	32	32	31	34	35	41	39	34	33	33	31	30
S10	CPU	51	52	55	51	46	50	51	47	48	54	51	48	51	52	46	45	50	48	45	47	51	60	62	61	59	54	57	59	54	52
	Mem	6	6	5	4	4	3	3	4	4	4	5	5	5	4	4	3	3	4	4	4	5	7	7	7	7	8	7	6	6	5
S11	CPU	66	66	68	58	48	45	44	43	43	46	46	46	41	39	44	45	46	46	46	47	52	34	28	56	48	26	25	28	27	25
	Mem	20	21	22	20	19	17	16	15	15	15	16	17	15	14	15	16	15	15	14	16	16	10	9	20	17	10	10	9	9	8
S12	CPU	56	55	53	53	52	50	49	56	56	54	48	45	47	48	47	47	47	48	49	50	51	51	51	49	49	50	49	47	50	51
	Mem	46	47	50	45	39	38	38	44	45	49	42	37	42	44	41	40	38	40	42	42	42	38	38	44	42	39	37	36	49	53
S13	CPU	42	39	31	33	35	38	39	39	39	39	37	34	33	33	33	33	32	34	36	38	38	42	43	43	43	47	43	41	35	33
	Mem	20	19	16	16	16	17	17	18	18	17	17	16	16	15	16	16	15	16	16	17	18	18	19	20	22	25	22	20	17	16
S14	CPU	16	16	15	17	19	20	20	19	19	19	19	20	21	20	15	15	15	16	16	16	17	19	16	17	17	16	16	16	23	16
	Mem	4	4	6	6	7	7	7	7	7	7	7	7	6	5	5	6	6	6	6	6	6	6	6	6	6	6	6	6	7	6
S15	CPU	27	30	38	42	47	50	52	59	59	56	57	58	63	66	59	55	27	36	29	25	25	37	40	43	46	53	57	60	57	56
	Mem	12	14	18	16	14	17	18	15	15	16	18	20	21	21	20	19	10	14	20	17	15	16	16	17	18	19	19	19	19	19
S16	CPU	54	54	52	49	45	48	50	46	45	44	46	47	48	48	41	40	45	45	44	44	47	46	38	38	39	42	42	41	44	45
	Mem	22	22	22	22	23	22	21	21	21	21	22	22	22	22	21	21	21	21	20	21	22	22	21	20	21	23	23	24	23	23
S17	CPU	81	73	48	55	62	61	60	51	56	58	59	60	62	61	59	59	51	50	43	46	49	50	43	43	44	43	42	55	45	41
	Mem	28	24	10	16	23	21	21	18	19	23	23	23	22	20	24	24	18	19	21	22	22	21	20	18	18	17	17	19	16	15
S18	CPU	50	46	33	34	35	29	26	37	37	31	36	44	40	37	39	40	42	36	25	33	38	48	50	43	43	41	40	40	34	32
	Mem	13	12	10	8	6	7	8	8	8	7	9	11	10	10	11	12	11	12	14	15	16	16	16	15	13	9	11	12	11	11
S19	CPU	62	61	59	51	43	42	41	34	32	29	33	39	39	38	33	33	38	41	46	38	28	33	33	30	29	24	25	38	31	28
	Mem	22	21	19	17	14	14	14	13	12	12	13	14	15	15	11	11	11	11	11	11	11	14	14	12	11	9	9	12	11	10
S20	CPU	32	34	40	41	43	45	47	48	47	42	54	57	57	57	39	38	53	61	64	55	51	63	66	66	67	68	60	53	58	61
	Mem	6	13	36	56	79	72	69	60	60	66	65	62	56	53	58	59	60	63	70	69	69	66	65	59	57	53	67	79	70	67
S21	CPU	73	68	59	49	38	32	29	31	35	45	40	34	38	40	28	25	13	15	19	23	24	22	21	21	21	19	19	19	17	16
	Mem	31	30	27	23	19	16	14	14	16	19	19	18	22	23	16	14	9	10	13	16	17	16	15	15	15	14	13	13	11	11
S22	CPU	73	73	72	67	61	74	79	66	67	67	67	68	69	70	61	59	54	51	45	52	59	66	46	56	54	51	51	62	47	14
	Mem	22	22	23	23	23	25	26	25	25	25	26	28	26	25	22	21	19	19	20	21	20	23	22	23	22	19	21	25	19	14
S23	CPU	22	22	24	23	22	23	23	20	19	19	20	25	25	22	23	23	24	33	29	24	23	23	23	24	24	23	18	13	12	12
	Mem	8	8	8	7	7	9	9	8	8	8	8	9	9	9	8	8	11	12	11	9	8	8	8	8	8	8	8	8	7	7
S24	CPU	46	47	50	52	54	41	42	46	46	44	44	45	46	47	50	51	58	60	69	67	61	47	44	46	48	53	57	60	58	57
	Mem	19	19	18	19	21	13	10	14	15	16	16	16	17	17	20	21	27	28	31	32	33	18	15	16	18	22	23	24	26	26
S25	CPU	50	49	47	46	46	44	43	48	49	46	44	45	47	49	46	46	51	47	41	47	51	46	45	54	54	54	52	50	46	45
	Mem	19	19	18	19	21	19	18	20	20	20	20	20	19	18	19	19	23	21	17	20	21	20	20	19	20	23	22	21	20	19
S26	CPU	43	45	51	42	30	29	29	38	39	43	43	40	40	30	21	18	16	12	12	21	27	22	23	11	17	31	27	22	32	34
	Mem	8	8	8	7	5	10	11	7	7	7	7	5	7	4	5	5	6	4	4	7	8	5	7	3	4	7	5	3	6	7
S31	CPU	23	21	13	21	29	23	21	9	11	24	22	19	18	17	15	18	48	34	10	10	10	32	36	24	27	36	25	14	14	13
	Mem	4	5	7	8	9	6	4	2	3	6	5	5	3	3	4	5	14	10	1	2	2	9	11	5	6	10	7	4	3	3
S32	CPU	79	80	82	80	78	77	77	66	60	32	32	31	33	34	34	34	35	34	32	34	35	40	40	31	32	34	38	42	36	34
	Mem	38	38	37	37	37	37	36	28	25	12	12	11	11	10	9	9	10	10	9	10	11	11	11	9	10	12	12	12	11	10
S33	CPU	38	37	37	34	31	33	34	34	35	36	34	31	35	37	32	30	25	26	28	34	39	34	33	33	33	32	35	38	38	38
	Mem	26	26	26	23	21	23	24	25	24	23	23	22	25	27	26	25	19	20	21	26	29	25	24	23	23	23	23	23	23	23
S34	CPU	60	64	75	76	77	81	79	79	73	78	81	87	87	86	85	85	82	72	54	51	54	52	41	52	58	70	64	59	72	77
	Mem	15	16	19	20	22	22	21	20	19	21	21	22	22	22	21	21	19	18	17	17	18	16	15	18	19	21	19	18	20	20
S35	CPU	94	94	93	90	87	93	96	96	95	91	91	85	82	81	79	80	85	77	64	68	73	86	39	35	36	37	37	37	37	37
	Mem	12	12	12	13	15	15	15	16	16	15	17	12	13	14	12	11	11	10	19	9	64	69	51	5	5	5	5	5	4	4
S36	CPU	51	49	46	51	54	39	34	33	33	30	30	29	17	10	45	54	58	52	41	41	40	50	52	51	50	50	51	57	64	60
	Mem	24	24	27	27	26	22	20	18	18	19	18	18	10	6	30	33	27	25	20	19	17	25	26	25	24	22	23	26	24	23
S37	CPU	43	45	48	36	22	31	34	51	49	34	34	34	36	35	28	30	54	43	22	17	13	42	46	26	39	68	43	21	41	48
	Mem	11	11	10	8	5	9	10	9	9	5	6	7	7	8	6	6	10	8	5	5	5	10	11	10	12	16	14	11	10	10
S38	CPU	39	42	51	45	31	28	30	28	28	31	30	28	34	37	30	29	28	29	32	40	41	39	38	39	38	36	38	42	44	46
	Mem	28	29	33	30	25	24	24	25	24	20	21	23	23	22	27	27	20	18	17	23	26	22	21	25	25	25	28	32	28	29
S43	CPU	35	38	53	50	41	43	46	37	36	36	47	48	47	47	53	51	32	32	43	41	39	33	41	42	46	40	42	45	36	34
	Mem	21	25	38	32	23	33	37	27	26	26	40	40	30	24	35	35	24	21	32	33	33	22	31	31	32	27	35	44	30	26
S44	CPU	22	22	21	21	21	21	21	23	24	24	24	24	26	27	26	26	27	27	27	26	26	26	26	25	24	23	24	25	25	25
	Mem	24	24	24	24	24	22	22	25	25	26	30	36	33	32	38	37	26	27	28	33	37	43	43	26	26	27	32	36	30	28
S45	CPU	15	15	17	15	12	13	13	11	11	14	16	13	13	12	12	12	13	12	11	12	10	12	13	14	13	12	13	30	20	16
	Mem	7	7	8	8	7	7	7	5	6	7	8	7	6	6	7	7	6	6	6	17	14	6	6	7	6	6	23	10	9	9
S46	CPU	37	39	45	40	35	34	33	39	40	43	41	38	39	39	43	44	45	45	45	46	48	46	46	45	44	43	43	44	37	35
	Mem	3	4	5	4	4	4	4	5	5	5	5	6	6	6	6	6	7	6	6	5	5	6	6	6	6	5	5	5	4	4
S47	CPU	35	36	41	36	29	45	51	46	46	39	31	25	22	20	31	34	38	37	35	35	37	41	40	42	41	56	52	49	49	41
	Mem	9	10	14	14	14	15	16	16	19	19	14	13	12	11	14	14	11	11	10	11	12	13	13	17	16	19	19	18	16	13

Bibliography

[1] Luiz André Barroso, Kourosh Gharachorloo, Robert McNamara, Andreas Nowatzyk, Shaz Qadeer, Barton Sano, Scott Smith, Robert Stets, and Ben Verghese. 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] Brent Bohnenstiehl, Aaron Stillmaker, Jon J. Pimentel, Timothy Andreas, Bin Liu, Anh T. Tran, Emmanuel Adeagbo, and Bevan M. Baas. KiloCore: A 32-nm 1001-Processor Computational Array. *IEEE Journal of Solid-State Circuits*, PP(99):1–12, 2017.

[3] Shekhar Borkar. Thousand core chips: A technology perspective. In *Proceedings of the 44th Annual Design Automation Conference*, DAC '07, pages 746–749, New York, NY, USA, 2007. ACM.

[4] Shekhar Borkar and Andrew A. Chien. The future of microprocessors. *Communications of the ACM*, 54(5):67–77, May 2011.

[5] Thomas D. Burd and Robert W. Brodersen. Energy efficient cmos microprocessor design. In *Proceedings of the Twenty-Eighth Hawaii International Conference on System Sciences*, volume 1, pages 288–297, Jan 1995.

[6] Qiong Cai, 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 *Proceedings of the 17th IEEE/ACM International Symposium on Low-power Electronics and Design*, ISLPED '11, pages 379–384, Piscataway, NJ, USA, 2011. IEEE Press.

[7] Saurabh Dighe, Sriram R. Vangal, Paolo Aseron, Shasi Kumar, Tiju Jacob, Keith A. Bowman, Jason Howard, James Tschanz, Vasantha Erraguntla, Nitin Borkar, Vivek K. De, and Shekhar Borkar. 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.

[8] 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.

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

[10] Xing Fu and Xiaouri Wang. Utilization-controlled task consolidation for power optimization in multi-core real-time systems. In *2011 IEEE 17th International Conference on Embedded and Real-Time Computing Systems and Applications*, volume 1, pages 73–82, Aug 2011.

[11] Mohammad Ghasemazar, Hadi Goudarzi, and Massoud Pedram. Robust optimization of a chip multiprocessor's performance under power and thermal constraints. In *2012 IEEE 30th International Conference on Computer Design (ICCD)*, pages 108–114, Sept 2012.

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

[13] Vinay Hanumaiah and Sarma Vrudhula. Energy-efficient operation of multicore processors by dvfs, task migration, and active cooling. *IEEE Transactions on Computers*, 63(2):349–360, Feb 2014.

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

[15] Jason Howard, Saurabh Dighe, Yatin Hoskote, Sriram Vangal, David Finan, Gregory Ruhl, David Jenkins, Howard Wilson, Nitin Borkar, Gerhard Schrom, Fabrice Pailet, Shailendra Jain, Tiju Jacob, Satish Yada, Sraven Marella, Praveen Salihundam, Vasantha Erraguntla, Michael Konow, Michael Riepen, Guido Droege, Joerg Lindemann, Matthias Gries, Thomas Apel, Kersten Henriss, Tor Lund-Larsen, Sebastian Steibl, Shekhar Borkar, Vivek De, Rob Van der Wijngaart, and Timothy Mattson. 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, Feb 2010.

[16] Intel. Intel turbo boost technology 2.0. `http://www.intel.com/content/www/us/en/architecture-and-technology/turbo-boost/turbo-boost-technology.html`, 2013. Online, accessed March 2015.

[17] Nikolas Ioannou, Michael Kauschke, Matthias Gries, and Marcelo Cintra. Phase-based application-driven hierarchical power management on the single-chip cloud computer. In *Proceedings of the 2011 International Conference on Parallel*

Architectures and Compilation Techniques, PACT '11, pages 131–142, Washington, DC, USA, 2011. IEEE Computer Society.

[18] 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*, MICRO 39, pages 347–358, Washington, DC, USA, 2006. IEEE Computer Society.

[19] Vaibhav Jain. Fast Process Migration on Intel SCC using Lookup Tables (LUTs). Technical Report Masters Thesis, Arizona State University, May 2013.

[20] Sudhanshu Shekhar Jha, Wim Heirman, Ayose Falcón, Jordi Tubella, Antonio González, and Lieven Eeckhout. Shared resource aware scheduling on power-constrained tiled many-core processors. *Journal of Parallel and Distributed Computing*, 100:30–41, 2017.

[21] Chanseok Kang, Seungyul Lee, Yong-Jun Lee, Jaejin Lee, and Bernhard Egger. Scheduling for better energy efficiency on many-core chips. In *19th Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP) In Conjunction with IPDPS 2015*, 19th JSSPP. Springer-Verlag, Hyderabad, India, May 2015.

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

[23] 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.

[24] Jian Li and José F. Martínez. Power-performance implications of thread-level parallelism on chip multiprocessors. In *IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS 2005)*, pages 124–134, March 2005.

[25] 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.

[26] David Meisner, Brian T. Gold, and Thomas F. Wenisch. Powernap: Eliminating server idle power. In *Proceedings of the 14th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS XIV, pages 205–216, New York, NY, USA, 2009. ACM.

[27] David Meisner and Thomas F. Wenisch. Dreamweaver: Architectural support for deep sleep. In *Proceedings of the Seventeenth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS XVII, pages 313–324, New York, NY, USA, 2012. ACM.

[28] 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*, PACT '08, pages 177–186, New York, NY, USA, 2008. ACM.

[29] Trevor Mudge. Power: A first-class architectural design constraint. *Computer*, 34(4):52–58, 2001.

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

[31] Andreas Olofsson. Epiphany-V: A 1024 processor 64-bit RISC System-On-Chip. https://arxiv.org/abs/1610.01832, Oct 2016.

[32] Jean-Marc Pierson and Henri Casanova. On the utility of dvfs for power-aware job placement in clusters. In *Proceedings of the 17th International Conference on Parallel Processing - Volume Part I*, Euro-Par'11, pages 255–266, Berlin, Heidelberg, 2011. Springer-Verlag.

[33] Krishna K. Rangan, Gu-Yeon Wei, and David Brooks. Thread motion: Fine-grained power management for multi-core systems. In *Proceedings of the 36th Annual International Symposium on Computer Architecture*, ISCA '09, pages 302–313, New York, NY, USA, 2009. ACM.

[34] 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*, MICRO 42, pages 459–468, New York, NY, USA, 2009. ACM.

[35] Niraj Tolia, Zhikui Wang, Manish Marwah, Cullen Bash, Parthasarathy Ranganathan, and Xiaoyun Zhu. Delivering energy proportionality with non energy-proportional systems: Optimizing the ensemble. In *Proceedings of the 2008 Conference on Power Aware Computing and Systems*, HotPower'08, pages 2–2, Berkeley, CA, USA, 2008. USENIX Association.

[36] John Wilkes. More Google cluster data. Google research blog, November 2011. Posted at http://googleresearch.blogspot.com/2011/11/more-google-cluster-data.html.

요약

대칭형 다중 처리 운영체제를 실행 시키는 캐쉬 일관성을 가지는 공유 메모리 아키텍처를 위한 전통적인 접근 방법은 전력관리가 가장 중요한 문제 중 하나로 존재하는 미래의 매니코어 시스템에는 적합하지 않다. 본 논문에서는 매니코어 시스템을 위한 계층적 전력관리 프레임워크를 소개한다. 제안한 프레임워크는 캐쉬 일관성을 가지는 공유 메모리가 필요 없으며, 다수의 코어들이 전압/주파수를 공유하고 다중 전압/다중 주파수를 지원하는 아키텍처에서 사용 가능하다. 이 프레임워크는 NUMA-인지 계층적 전력관리 기술로 동적 전압 및 주파수 교환(DVFS)과 워크로드 마이그래이션을 사용한다. 여기서 워크로드 마이그래이션 계획을 위해 사용된 탐욕 알고리즘은 서로 상충하는 비슷한 작업량의 패턴을 가진 작업을 같은 전압 영역으로 모으는 목표와 작업을 데이터가 있는 위치와 가까운 곳으로 이동하는 목표를 고려한다. 제안된 프레임워크는 소프트웨어로 구현되어 캐쉬 일관성이 없는 48 코어의 칩 레벨 멀티프로세서 하드웨어에서 평가되었다. 본 논문의 프레임워크를 데이터 센터 작업 패턴으로 광범위에 걸친 실험을 수행한 결과 최첨단의 DVFS 기술과 DVFS와 NUMA-비인지 워크로드 마이그래이션을 같이 사용한 전력관리 기술에 비해 상대적으로 각각 30%와 5%의 전력소모당 처리 작업량 향상을 큰 성능손실 없이 이루었다.

주요어: 매니코어 아키텍쳐, 불균일 기억 장치 접근, 스케쥴링, 동적 전압 및 주파수 변경, 에너지 효율

학번: 2015-22902