OPTIMAL POWER ALLOCATION FOR MULTIPROGRAMMED WORKLOADS ON SINGLE-CHIP HETEROGENEOUS PROCESSORS

Euijin Kwon[1,2], Jae Young Jang[2], Jae W. Lee[2], and Nam Sung Kim[2,3]
[1]Samsung Electronics, Yongin, Korea [2]Sungkyunkwan University, Suwon, Korea
[3]University of Wisconsin-Madison, WI, USA
[1]euijin.kwon@samsung.com [2]{euijinkwon, rhythm2jay, jaewlee, nskim3}@skku.edu [3]nskim3@wisc.edu

Abstract

Due to the growing requirements of performance and energy efficiency, processor manufacturers now integrate both CPU and GPU cores onto a single chip and support fine-grain dynamic voltage/frequency scaling (DVFS). To effectively utilize abundant hardware resources, the CPU and the GPU on a single-chip heterogeneous processor (SCHP) are required to execute multiple programs simultaneously via both space and time sharing. Since the performance is primarily limited by the available power budget, it is crucial to optimally allocate it across the CPU and the GPU by considering both workload characteristics and evaluation metrics. This paper advocates adaptive, workload-aware power allocation for multiprogrammed workloads on an SCHP with fine-grain DVFS capabilities. Using a detailed cycle-level SCHP simulator, we first demonstrate that workload-aware power allocation can improve throughput and energy efficiency by an average of 12% and 18%, respectively, over workload-oblivious uniform power allocation for 11 multiprogrammed workloads. We also propose two efficient run-time algorithms that find an optimal voltage/frequency setting for the two metrics (i.e., throughput and energy efficiency), which allow the SCHP to reach the optimal or near-optimal setting within four iterations.

1. Introduction

A single-chip heterogeneous processor (SCHP), which is comprised of various types of processing elements such as CPUs, GPUs, and accelerators, has been widely adopted by all computing segments [1, 2, 3]. In particular, integrating both CPU and GPU cores onto a single chip can greatly reduce the performance and energy penalties incurred by communications between them. Furthermore, the number of CPU and GPU cores placed on a chip is expected to increase continuously in the foreseeable future with technology scaling.

Due to the growing requirements of performance and energy efficiency, per-core voltage/frequency (V/F) domains began to be supported for CPUs. Extrapolating this trend, we expect that the GPU in a future SCHP will be provided with more than one voltage/frequency (V/F) domains. Moreover, to maximize resource utilization, those GPUs will be required to execute multiple programs simultaneously via both space and time sharing [4].

The limited chip power budget often prevents SCHPs from further increasing the overall throughput. Thus, it is crucial to carefully allocate the power budget across all processing elements. Although the performance of the CPU or the GPU in an SCHP is typically proportional to the allocated power budget, the amount of improvement depends on both workload characteristics and evaluation metrics (i.e., throughput and throughput/Watt for performance and energy efficiency, respectively). Moreover, a complex interplay among multiple programs running concurrently poses additional challenges in determining an optimal V/F setting for each processing element. Therefore, adaptive, workload-aware power allocation is highly desirable to optimize multiprogrammed workloads on an SCHP.

Various techniques have been proposed to improve the overall throughput of both single-chip and multi-chip heterogeneous platforms. Some recent studies investigated workload and/or power budget partitioning between the CPU and the GPU [5, 6]. Wang *et al.* investigated optimal workload and power budget allocation between the CPU and the GPU. However, their work focuses only on single-programmed workloads without consideration of spatial multitasking on a GPU. Adriaens *et al.* demonstrated the benefit of spatial multitasking on a GPU by partitioning GPU cores among multiple programs running simultaneously [4]. Although this approach significantly improves the overall throughput, it targets a discrete GPU without considering the coupling between the CPU and the GPU sharing the chip power budget on an SCHP.

By contrast, we explore optimal power allocation schemes for *multiprogrammed* workloads on an SCHP with *the GPU* having two independent V/F domains in this paper. Our premise is that programs exhibit non-uniform performance sensitivities to operating frequencies (hence allocated power budget). To make the best use of a limited chip power budget, it is necessary to search for an optimal V/F setting at run-time by considering both workload characteristics and evaluation metrics. The key contributions of this paper are summarized as follows:

- We analyze potential throughput improvement of adaptive, workload-aware power allocation schemes for multiprogrammed workloads. We find that the optimal V/F

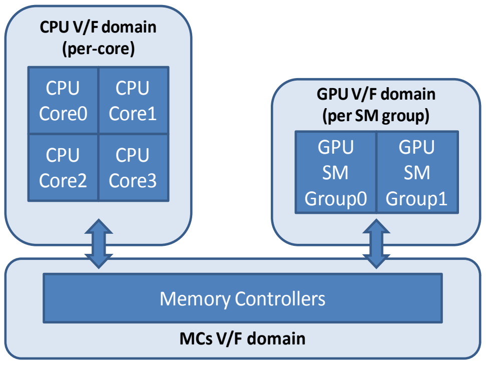

Figure 1. Block diagram of single-chip heterogeneous processor

settings for the CPU and the GPU vary greatly depending on workload characteristics and evaluation metrics.

- We propose and evaluate two run-time algorithms that can maximize throughput and energy efficiency, respectively by determining optimal V/F settings for multiple programs running concurrently.

2. Motivation

2.1 Single-chip Heterogeneous Processors

Recent trends indicate that the GPU performance improves more rapidly than the CPU performance on SCHPs. For example, Apple's A5X system-on-a-chip (SoC) has twice as many GPU cores compared to its predecessor (A5), while reusing the same ARM Cortex-A9 based a dual-core CPU [7]. This trend is likely to continue due to growing performance demands from data-parallel applications on both mobile and desktop platforms.

Since a single program may fall short of keeping all GPU cores busy, Adriaens *et al.* proposed spatial multitasking on GPU for multiple applications to share the GPU's hardware resources and run concurrently [4]. This is a promising approach that ensures high utilization of the GPU. In this paper, as a simple realization of this approach, we statically partition the GPU into two groups of stream multiprocessors (SMs) and assign one V/F domain for each group.

Figure 1 shows a block diagram of an SCHP with four CPU cores and two GPU SM groups, each of which has 8 SMs. To maximize energy efficiency, we assume each CPU core is supported by its own V/F domain. For the rest of this paper, we will use this SCHP as our baseline.

2.2 Opportunities for Multiprogrammed Workloads

Multiple V/F domains enable independent operating frequencies for CPU cores and GPU SM groups in an SCHP. This introduces an optimization problem of allocating the chip power budget across processing elements while maximizing the overall throughput or energy efficiency. An optimal power allocation depends on the characteristics of a given set of programs since different programs have non-uniform performance sensitivities to changes in allocated power budget (i.e., the number and types of processing elements and their V/F setting). It is also affected by the evaluation metric (e.g., throughput vs. throughput/Watt).

In a multiprogrammed environment where two or more programs are running concurrently, the chip-level power

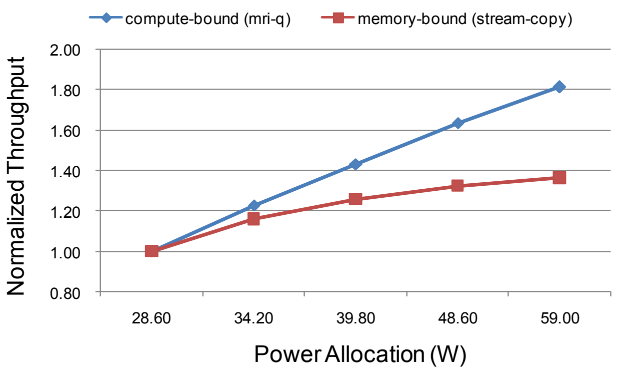

Figure 2. Difference of throughput improvement between compute-bound and memory-bound

budget should be allocated to those programs. By exploiting different performance sensitivity to allocated power budget, we can find optimal, possibly asymmetric, V/F settings that can maximize overall throughput or throughput/Watt. Figure 2 evidences the existence of such optimal settings with two programs: *mri-q* and *stream-copy*. The throughput of *mri-q* increases more rapidly than *stream-copy* as the allocated power budget increases. It is because the former is compute-bound, whereas the latter is memory-bound. Thus, increasing power allocation for the CPU and/or GPU cores that execute *stream-copy* does not benefit as much as *mri-q* since the shared memory system becomes the throughput bottleneck.

Once per-program power budget is determined, there is another level of power budget allocation between CPUs and GPUs running the same program. If a program has complex data dependences and control flows, it is likely to run more efficiently on the CPU, hence more power budget should be allocated to the CPU; if a program has abundant data-level parallelism, more power budget should be allocated to the GPU. Even if a program is suitable for either a CPU or a GPU, neither of them can consume the entire chip power budget due to thermal and reliability constraints and thus allocating workload on both the CPU and the GPU leads to higher throughput [5]. Hence, it is highly desirable to dynamically allocate the chip power budget by considering the characteristics of each executed program. The rest

Table 1. Configuration parameters

CPU freq/volt	1.67-3.44GHz/0.72-0.99V
CPU core fetch/issue/retire	4/4/4
CPU IL1 and DL1	64KB/2-way/64B 2 cycles
CPU L2 per core	1MB/16-way/64B 20 cycles
CPU store buffer	16 per core
CPU branch mis-pred, penalty	14 cycles
CPU MSHR	20 per core
# of GPU SMs / V/F domains	16/2
GPU freq/volt	350-710MHz/0.72-0.99V
GPU # of registers per SM	16384
GPU # of threads per SM	1024
GPU # of CTAs per SM	8
GPU L1$ per SM	32KB
GPU SIMD Width	8
GPU Warp Size	32
GPU branch divergence	Immediate post dominator
# of MC/scheduling policy	2/FR-FCFS
Memory frequency	1600MHz (DDR3)

Table 2. Voltage/frequency and power consumption

Level	Voltage (V)	Frequency (GHz)		Power (Watts)	
		CPU	GPU	CPU (2cores)	GPU (8 SMs)
1	0.75	1.89	0.39	17.40	11.20
2	0.81	2.31	0.48	24.80	16.80
3	0.87	2.70	0.56	34.20	22.40
4	0.93	3.07	0.64	46.40	31.20
5	0.99	3.44	0.71	62.80	41.60

of this paper explores optimal power allocation for various multiprogrammed workloads with two programs running on an SCHP with multiple V/F domains shown in Figure 1.

3. Methodology

3.1 Baseline Processor Model

The baseline SCHP configuration in this study is based on the previous work [5], which is configured to model an SCHP with 4 CPU cores and NVIDIA's GT260M GPU. This GPU is comparable to that of an integrated GPU on Intel's and AMD's SCHPs. The original model has one GPU with 12 SMs but we modify it to have 16 SMs to reflect recent SCHP design trends. To run multiple programs, we assume a static SM partitioning, evenly dividing 16 SMs between programs. Important configuration parameters of the SCHP are tabulated in Table 1.

We also calculate the power consumption of the CPU and the GPU as a function of V/F using the method in [5]. The total power budget is assumed to be 100W. The maximum power consumption allowed is 80W for the CPU and 64W for the GPU, respectively. We assume that a multiprogrammed workload is composed of 2 programs and each program uses a half of the available CPU and GPU cores (i.e,. 2 CPU cores and 8 GPU SMs). Table 2 shows the frequency and calculated power consumption for one CPU group (2 cores) and one GPU group (8 SMs) with varying V/F at their thermal design power (TDP) point, which is independent from a program's varying behavior at run-time.

In the baseline configuration with the total power budget of 100W, two CPU cores as a group operate at 0.81V/2.31GHz consuming 24.8W. Eight GPU SMs as a group operate at 0.87V/0.56GHz consuming 22.4W, and the total power budget is almost equally allocated across all CPU cores and GPU groups. Consequently, the 4 CPU cores and the 16 GPU SMs consume 49.6W and 44.8W, respectively. Hence, in a two-program setup, each program can use 47.2W assuming equal partitioning between the CPU and the GPU with the total power consumption of 94.4W.

We use a SCHP simulator which integrates gem5 [8] and GPGPU-Sim [9] to model the CPU and the GPU, respectively. It has two memory controllers which service all memory requests from the CPU and the GPU. This simulator is based on a simulator released by Wang *et al.* [5], which can execute single-programmed workloads. We adapt this simulator to support per-core DVFS for the CPU, where the V/F of each CPU core can be independently adjusted to control its power allocation. Furthermore, we also augment the simulator to provide multiple V/F domains for multiple GPU SM groups.

3.2 Workloads

We use six benchmarks taken from Parboil [10], Rodinia [11], and NVIDIA developer zone [12], as summarized in Table 3. We divide these benchmarks into three groups:

Table 3. The information of benchmark programs

Benchmark	Acronym	Source	Characteristics
Magnetic Resonance Imaging Q	MRQ	Parboil	Compute-bound
Stream Cluster	SCL	Rodinia	Compute-bound
Hotspot	HOT	Rodinia	Neutral
Sum of Absolute Difference	SAD	Parboil	Neutral
Stencil	STN	Parboil	Memory-bound
Stream Copy	SCP	NVIDIA dev zone	Memory-bound

compute-bound, memory-bound, and neutral (i.e., falling between compute- and memory-bound ones). To analyze benchmark characteristics, we either reuse the classifications from [13] (for MRQ, HOT, SAD, and STL), or perform simulation (for SCL and SCP). For SCP, we only use *copy* operations and discard the other three (*scale*, *add*, and *triad*) to make it more memory intensive.

4. Evaluation

To demonstrate the potential benefit of workload-aware, asymmetric power allocation among programs, we create 11 workloads, each of which consists of two programs from one of the three benchmark groups: compute-bound (C), memory-bound (M), and neutral (N). One program uses a half of the available computing resources for execution, one CPU group (2 cores) and one GPU group (8 SMs). We evaluate the performance using two metrics: (loop) iterations per second (IPS) for throughput and IPS per Watt (IPS/Watt) for energy efficiency. Note that both numbers are normalized to the corresponding numbers of the baseline discussed in Section 3.1

4.1 Optimal Power Allocation

First, we perform an exhaustive search to find the optimal power allocation between two programs for each of the 11 workloads, which is summarized in Table 4. All of the six benchmark programs we use for evaluation (Table 2) have ample data-level parallelism to favor the GPU over the CPU. However, a single GPU group cannot use more than 41.6 W due to its power constraint. Thus, it is beneficial to exploit the unused power budget of CPU cores for executing part of the work. Furthermore, the optimal power allocation between the CPU and the GPU, as well as between two programs is affected not only by workload characteristics but also by the evaluation metric.

Figure 3 shows the IPS and IPS/Watt improvements for the 11 workloads with the optimal power allocation. All numbers are normalized to the baseline case where the power budget is almost equally distributed among the four CPU and GPU groups. On average, the optimal configuration improves IPS and IPS/Watt by 12% and 18%, respectively. Thus, we expect that workload-aware, asymmetric power allocation among processing elements will lead to further IPS and IPS/Watt improvements.

4.2 Case Studies

Figure 4 shows the results of exhaustive search for 4 representative cases out of the 11 workloads, which are shaded in Table 4. For the case studies, we use four programs taken from the three groups: MRQ (C), HOT (N), STN (M) and SCP (M). The X- and Y-axes show the power allocated to the two co-scheduled programs, respectively. Note that there are five levels of power allocation for either CPU or

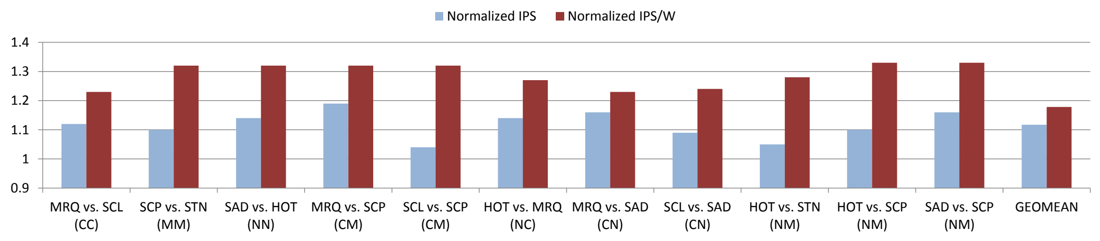

Figure 3. IPS and IPS/Watt for 11 multiprogrammed workload cases

Table 4. Optimal power allocation for 11 workloads

P1	P2	Metric 1: IPS				Metric 2: IPS/Watt			
		P1 (Watt)		P2 (Watt)		P1 (Watt)		P2 (Watt)	
		CPU	GPU	CPU	GPU	CPU	GPU	CPU	GPU
MRQ (C)	SCL(C)	17.40	31.20	17.40	31.20	17.40	16.80	17.40	16.80
SCP (M)	STN (M)	17.40	31.20	17.40	31.20	17.40	11.20	17.40	11.20
SAD (N)	HOT (N)	17.40	31.20	17.40	31.20	17.40	11.20	17.40	16.80
MRQ (C)	SCP (M)	17.40	41.60	17.40	22.40	17.40	22.40	17.40	16.80
SCL (C)	SCP (M)	17.40	41.60	17.40	22.40	17.40	11.20	17.40	11.20
HOT (N)	MRQ(N)	17.40	31.20	17.40	31.20	17.40	11.20	17.40	22.40
MRQ (C)	SAD (N)	17.40	31.20	17.40	31.20	17.40	16.80	17.40	22.40
SCL (C)	SAD (N)	17.40	31.20	17.40	31.20	17.40	16.80	17.40	11.20
HOT (N)	STN (M)	17.40	41.60	17.40	22.40	17.40	11.20	17.40	11.20
HOT (N)	SCP (M)	17.40	41.60	17.40	22.40	17.40	11.20	17.40	11.20
SAD (N)	SCP (M)	17.40	41.60	17.40	22.40	17.40	11.20	17.40	22.40

GPU as in Table 3, so there are 25 levels of power configuration for each program. Therefore, both X- and Y-axes have 25 points sorted by the GPU power level first, and then by the CPU power level as follows: (GPU, CPU) = (L1, L1)→…→(L1, L5)→(L2, L1)→…→(L5, L5). However, 25 combinations are not entirely usable since there are constraints on maximum power in a CPU level, a GPU level, and the entire chip level as discussed in Section 3.1

We use both IPS and IPS/Watt for metrics, and all numbers are normalized to the baseline case as before. For each configuration, the higher the number is, the darker the corresponding box is. A circle marks the optimal power configuration; the dotted line in each graph indicates a line of uniform power allocation between the two programs.

Case 1. MRQ (C) vs. SCP (M): Figure 4(a) and 4(b) show geometric means of IPS and IPS/Watt, respectively, when running MRQ (compute-bound) and SCP (memory-bound) simultaneously. As expected, allocating more power to MRQ improves both IPS and IPS/Watt since MRQ has a higher marginal throughput gain to additional power allocation than SCP. By doing so, the optimal configuration improves IPS by 19% and IPS/Watt by 32% over the baseline. Optimizing energy efficiency favors configurations in the lower left corner because of diminishing returns of energy efficiency as the allocated power budget increases.

Case 2. SCP (M) vs. STN (M) : Figure 4(c) and 4(d) plot the IPS and IPS/Watt for a workload with two memory-bound programs: SCP and STN. Since both exhibit similar throughput characteristics when power budget changes, the optimal configuration is fair distribution. For IPS/Watt, we observe that the best energy efficiency is achieved at the minimum power configuration because the performance of a memory-bound program is bottlenecked by memory system performance. This configuration achieves 10% higher IPS, and 32% higher IPS/Watt than the baseline.

Case 3. HOT (N) and SCP (M): Figure 4(e) and 4(f) show the IPS and IPS/Watt curves when running HOT and SCP simultaneously. HOT is relatively more compute-intensive but less memory-intensive than SCP. Therefore, to optimize IPS, more power budget should be allocated to HOT, which has a higher marginal gain. For IPS/Watt, there are multiple optimal configurations whose IPS/Watt numbers are almost the same. Although the optimal point is placed on the fair distribution line in Figure 4(f), most of the other optimal points are on the HOT side. We observe 10% and 33% improvements over the baseline case for IPS and IPS/Watt, respectively.

Case 4. HOT (N) vs. MRQ (C): Figure 4(g) and 4(h) show the IPS and IPS/Watt curves for a workload composed of HOT (neutral) and MRQ (compute-bound) . We find that the optimal configuration for IPS falls on the uniform distribution line. It is because, although HOT is categorized as *neutral*, the difference in performance characteristics is relatively small. However, there is only a slight difference between the first and second optimal points, and the second and third optimal configurations are placed towards the MRQ side as expected. We observe 14% and 27% gains of IPS and IPS/Watt, respectively, over the baseline configuration.

5. Run-time Algorithms

The results in Section 4 support our argument that workload-aware, asymmetric power allocation can significantly improve the throughput and energy efficiency for multiprogrammed workloads. Thus, we describe two heuristic-based run-time algorithms that find an optimal power configuration efficiently to maximize IPS and IPS/Watt, respectively. We evaluate these algorithms and Table 5 shows the experimental results with 11 workloads. In experiments, we assume the threshold value of 0.0005 IPS/Watt in finding optimal IPS point and set (GPU_{P1}, GPU_{P2}) = (L1, L1) as the starting point in searching an optimal IPS/Watt point.

Algorithm to optimize IPS: According to our experiments, an optimal configuration is always found on the perimeter of the configuration space. More specifically, there are only three distinct optimal points for all 11 workloads represented by the following pairs of GPU power levels for the two programs P1 and P2: (GPU_{P1}, GPU_{P2}) = (L5, L3), (L4, L4), and (L3, L5). Depending on (1) which program is more compute-bound and (2) how much different the two programs' throughput sensitivities to the allocated power budget, the system may allocate more power to one program or the other; if their difference in throughput sensitivities is smaller than a threshold, both programs receive the same power budget (of Level 4). Note that the CPU power is always set to Level 1 since our workloads favor the GPU over the CPU; however, this may change for different types of workloads.

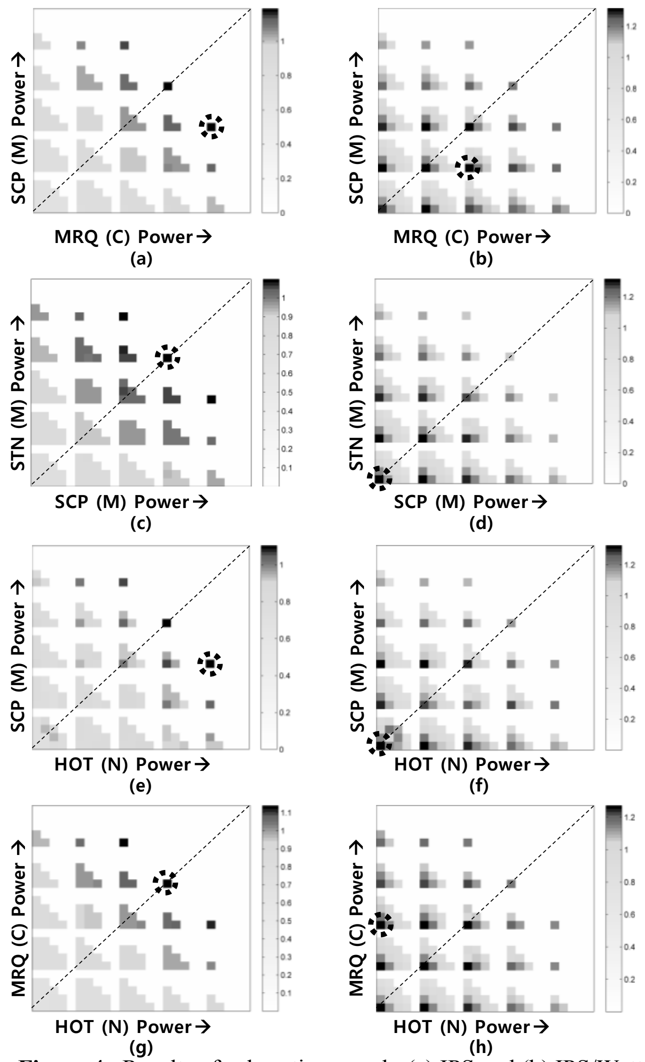

Figure 4. Results of exhaustive search: (a) IPS and (b) IPS/Watt for MRQ vs. SCP; (c) IPS and (d) IPS/Watt for SCP vs. STN; (e) IPS and (f) IPS/Watt for HOT vs. SCP; (g) IPS and (h) IPS/Watt for HOT vs. MRQ

Figure 5(a) presents an optimization algorithm based on run-time performance monitoring. As in Figure 2, the run-time monitor measures the slopes of performance improvement when the power budget for GPU increases from L1 to L5 for both programs. These slopes are denoted by SLOPE_P1 and SLOPE_P2. If their difference is greater than a specific threshold (denoted by THRESHOLD), the algorithm allocates more power budget to the program that has a higher slope. This algorithm can be generalized to deal with multiple, Pareto optimal points along the perimeter of the configuration space by considering the amount of difference between SLOPE_P1 and SLOPE_P2. Although the algorithm finds an optimal configuration only once at program invocation, it can be also extended to deal with the phase behavior of a workload over time. We can achieve this by monitoring IPS periodically and triggering recalibration as needed, for example. Columns 3-5 of Table 5 show the search results. The 7 of 11 workloads reach the optimal points correctly. For the four non-optimal cases, average IPS loss is 0.49%.

Algorithm to optimize IPS/Watt: In most cases, the optimal power configuration for IPS/Watt is placed near the minimum power configuration (i.e., the case of all L1's). Therefore, Figure 5(b) shows a simple gradient search algorithm. Again, as GPU power allocation dominantly determines the program's throughput, we only search for an optimal configuration of GPU power. The algorithm starts searching at the minimum power configuration and the measure energy efficiency of the next power level for both P1 and P2. If one of the two yields higher energy efficiency, the system moves to that configuration. The optimization loop continues iteration until all of the neighboring configuration points have lower energy efficiencies than the current configuration. Columns 6-8 of Table 5 show the results of searching optimal IPS/Watt points for 11 workloads. This algorithm takes 1 to 4 iterations with an average of 2.27 iterations to reach the true optimal power configuration for IPS/Watt with 100 % accuracy.

6. Related Work

As SCHPs become the mainstream computing devices in all computing segments, there are proposals to make the best use of abundant hardware resources offered by both the CPU and the GPU. Luk *et al.* proposed a technique to partition a given workload adaptively between the CPU and GPU to minimize its total execution time [6]. Wang *et al.* proposed a run-time algorithm to jointly optimize workload and power partitioning between the CPU and GPU [5]. Like our work both studies pursue dynamic throughput optimization exploiting platform heterogeneity. However, they are limited to single-programmed workloads, where a single parallel program is partitioned into multiple tasks running on both the CPU and GPU.

Jeong *et al*. proposed an effective partitioning scheme of DRAM bandwidth to optimize a pair of co-scheduled programs running on the CPU and the GPU, respectively [14]. The target workload contains a program with real-time constraints placed on the GPU. By dynamically adjusting the priorities to access DRAM, the SCHP meets the real-time constraints for the GPU program, and subject to that, maximizes the throughput of the CPU program. Kim *et al*. also co-optimized CPU and GPU programs on an SCHP platform integrated with hybrid DRAM/PRAM-based main memory [15]. However, both proposals aimed to maximize the utilization of limited memory bandwidth without considering power budget allocation, and the program runs on either the CPU or the GPU, but not on both. Instead, our work optimizes the allocation of a different shared resource, chip power budget, and exploits spatial multiplexing on both the CPU and the GPU.

Adriaens *et al*. advocate spatial multitasking on GPU to allow multiple programs to run simultaneously and maximize resource utilization [4]. Unlike our work their proposal is to partition SMs on a single, discrete GPU. Therefore, they do not take into account complex tradeoffs in power budget allocation between the CPU and the GPU on SCHP.

7. Conclusion

Modern computing systems in all scales are demanded to process complex, multiprogrammed workloads, and embrace heterogeneity in processing elements for both higher performance and energy-efficiency. This paper proposed optimal power allocation schemes for multiprogrammed workloads running on an SCHP. The SCHP we target inte-

```
while (every regular time intervals)
    IPS_MIN = get_IPS (P1_CPU_L1, P1_GPU_L1,
                       P2_CPU_L1, P2_GPU_L1)
    IPS_P1_MAX = get_IPS (P1_CPU_L1, P1_GPU_L5,
                          P2_CPU_L1, P2_GPU_L1)
    IPS_P2_MAX = get_IPS (P1_CPU_L1, P1_GPU_L1,
                          P2_CPU_L1, P2_GPU_L5)
    SLOPE_P1 = cal_slope (IPS_MIN, IPS_P1_MAX)
    SLOPE_P2 = cal_slope (IPS_MIN, IPS_P2_MAX)

    if (|SLOPE_P1-SLOPE_P2|) < (THRESHOLD)
        set_IPS(P1_CPU_L1, P1_GPU_L4,
                P2_CPU_L1, P2_GPU_L4)
    else if (SLOPE_P1 > SLOPE_P2)
        set_IPS(P1_CPU_L1, P1_GPU_L5,
                P2_CPU_L1, P2_GPU_L3)
    else
        set_IPS(P1_CPU_L1, P1_GPU_L3,
                P2_CPU_L1, P2_GPU_L5)
end while
```

(a)

```
P1_CPU_FINAL = P1_CPU = P1_CPU_L1
P1_GPU_FINAL = P1_GPU = P1_GPU_L1
P2_CPU_FINAL = P2_CPU = P2_CPU_L1
P2_GPU_FINAL = P2_GPU = P2_GPU_L1
IPSW_BASE = get_IPSW (P1_CPU, P1_GPU,
                      P2_CPU, P2_GPU)

while (1)
    P1_GPU = P1_GPU_FINAL + 1 level
    P2_GPU = P2_GPU_FINAL
    IPSW_P1 = get_IPSW (P1_CPU, P1_GPU,
                        P1_CPU, P2_GPU)

    P1_GPU = P1_GPU_FINAL
    P2_GPU = P2_GPU_FINAL + 1 level
    IPSW_P2 = get_IPSW (P1_CPU, P1_GPU,
                        P1_CPU, P2_GPU)

    if (IPSW_BASE >= IPSW_P1 &&IPSW_BASE >= IPS
    W_P2)
        exit
    else if (IPSW_P1 >= IPSW_P2)
        IPSW_BASE = IPSW_P1
        P1_GPU_FINAL = P1_GPU_FINAL + 1 level
    else
        IPSW_BASE = IPSW_P2
        P1_GPU_FINAL = P1_GPU_FINAL
end while
```

(b)

Figure 5. Run-time algorithms for: (a) IPS (b) IPS/Watt

grates multiple CPU and GPU cores on a single chip, where multiple V/F domains are supported to improve energy efficiency. Depending on the workload and evaluation metric, the optimal power allocation between the CPU and the GPU and between two programs (via adjusting V/F of each domain) varies greatly. This motivates us to devise two run-time algorithms that determine an optimal power allocation for maximizing throughput (IPS) and/or energy efficiency (IPS/Watt), respectively, when multiple programs are running concurrently. Our evaluation shows that the optimal power allocation among four groups of CPU and GPU cores improves IPS and IPS/Watt by 12% and 18%, respectively, compared to the baseline configuration that equally distributes the power budget among those four groups. In the future, we plan to evaluate these algorithms on platforms with a larger search space and with a wider variety of workloads such as those more suitable for CPUs.

Table 5. Search results with run-time algorithms

P1	P2	Metric 1 : IPS			Metric 2 : IPS/Watt		
		Predict-ed	Actual	Miss Penalty (%)	Desti-nation	Actual	# of Iter.
MRQ (C)	SCL(C)	(L5,L3)	(L4,L4)	0.16	(L2,L2)	(L2,L2)	3
STN (M)	SCP (M)	(L4,L4)	(L4,L4)	0	(L1,L1)	(L1,L1)	1
SAD (N)	HOT (N)	(L4,L4)	(L4,L4)	0	(L1,L2)	(L1,L2)	2
MRQ (C)	SCP (M)	(L5,L3)	(L5,L3)	0	(L3,L2)	(L3,L2)	4
SCL (C)	SCP (M)	(L5,L3)	(L5,L3)	0	(L1,L1)	(L1,L1)	1
MRQ (C)	HOT (N)	(L5,L3)	(L4,L4)	0.61	(L3,L1)	(L3,L1)	3
MRQ (C)	SAD (N)	(L4,L4)	(L4,L4)	0	(L2,L3)	(L2,L3)	4
SCL (C)	SAD (N)	(L5,L3)	(L4,L4)	0.71	(L2,L1)	(L2,L1)	2
HOT (N)	STN (M)	(L5,L3)	(L5,L3)	0	(L1,L1)	(L1,L1)	1
HOT (N)	SCP (M)	(L4,L4)	(L5,L3)	0.47	(L1,L1)	(L1,L1)	1
SAD (N)	SCP (M)	(L5,L3)	(L5,L3)	0	(L1,L3)	(L1,L3)	3
Average		-	-	0.49	-	-	2.27

Acknowledgements

We thank W. Hao for providing the baseline cycle-accurate simulator for SCHP and helping us to implement multiple V/F domains. This work was supported in part by the IT R&D program of MKE/KEIT [KI001810041244, Smart TV 2.0 Software Platform], a research grant from Samsung Electronics, and NSF CNS-1217102.

References

[1] M. Yuffe, E. Knoll, M. Mehalel, J. Shor and T. Kurts, "A fully integrated multi-CPU, GPU and memory controller 32nm processor," in *ISSCC*, 2011.

[2] S. Nussbaum, "AMD Trinity Fusion APU," in *Proc. of the Hot Chips: A Symp. on HPC*, 2012.

[3] Qualcomm, "Snapdragon MDP MSM8660 datasheet," [Online]. Available: https://developer.qualcomm.com/.

[4] J. Adriaens, K. Compton, N. Kim and M. Schulte, "The Case for GPGPU Spatial Multitasking," in *HPCA*, 2012.

[5] H. Wang, V. Sathish, R. Singh, M. Schulte and N. Kim, "Workload and Power Budget Partitioning for Single-Chip Heterogeneous Processors," in *PACT*, 2012.

[6] C. Luk, S. Hong and H. Kim, "Qilin: Exploiting parallelism on heterogeneous multiprocessors with adaptive mapping," in *MICRO*, 2009.

[7] "Apple A5X - Wikipedia," [Online]. Available: http://en.wikipedia.org/wiki/Apple_A5X.

[8] N. Binkert et al., "The gem5 simulator," *ACM SIGARCH Comp.Arch.News,* pp. 1-7, Aug 2011.

[9] A. Bakhoda, G. Yuan, W. Fung, H. Wong and T. Aamodt, "Analyzing CUDA Workloads using a Detailed GPU Simulator," in *ISPASS*, 2009.

[10] "Parboil Benchmark Suite," [Online]. Available: http://impact.crhc.illinois.edu/parboil.php.

[11] C.Shuai, M.Boyer, J.Meng, D.tarjang, J.W.Sheaffer, S.H.Lee and K.Skadron, "Rodinia: A Benchmark Suite for Heterogeneous Computing," in *IISWC*, 2009.

[12] "Stream Benchmark from NVIDIA developer zone," [Online]. Available: https://devtalk.nvidia.com/.

[13] V. Sathish, M. Schulte and N. Kim, "Lossless and Lossy Memory I/O Link Compression for Improving Performance of GPGPU Workloads," in *PACT*, 2012.

[14] M.K.Jeong, M.Erez, C.Sudanthi and N.Paver, "A QoS-Aware Memory Controller for Dynamically Balancing GPU and CPU Bandwidth Use in an MPSoC," in *DAC*, 2012.

[15] D. Kim, S. Lee, J. Chung, D. Kim, D. Woo, S. Yoo and S. Lee, "Hybrid DRAM/PRAM-based main memory for single-chip CPU/GPU," in *DAC*, 2012.