ONAC: Optimal Number of Active Cores Detector for Energy Efficient GPU Computing

Xian Zhu, Mihir Awatramani, Diane Rover and Joseph Zambreno
Department of Electrical and Computer Engineering
Iowa State University
Ames, IA, USA
Email: {xian, mihir, drover, zambreno}@iastate.edu

***Abstract*—Graphics Processing Units (GPUs) have become a prevalent platform for high throughput general purpose computing. The peak computational throughput of GPUs has been steadily increasing with each technology node by scaling the number of cores on the chip. Although this vastly improves the performance of several compute-intensive applications, our experiments show that some applications can achieve peak performance without utilizing all cores on the chip. We refer to the number of cores at which performance of an application saturates as the optimal number of active cores (N_{opt}). We propose executing the application on N_{opt} cores, and power-gating the unused cores to reduce static power consumption.**

Towards this target, we present ONAC (Optimal Number of Active Cores detector), a runtime technique to detect N_{opt}. ONAC uses a novel estimation model, which significantly reduces the number of hardware samples taken to detect the optimal core count, compared to a sequential detection technique (Seq-Det). We implement ONAC and Seq-Det in a cycle-level GPU performance simulator and analyze their effect on performance, power and energy. Our evaluation shows that ONAC and Seq-Det can reduce energy consumption by 20% and 10% on average for memory-intensive applications, without sacrificing more than 2% performance. The higher energy savings for ONAC comes from reducing the detection time by 45% as compared to Seq-Det.

I. Introduction

Graphics Processing Units (GPUs) are increasingly becoming the preferred choice of accelerator for various scientific and engineering applications [17, 21]. A primary reason for their continued adoption is the higher floating point throughput GPUs offer compared to CPUs. Moreover, the amount of computational power in GPUs is increasing at a steady rate. The peak single and double precision floating point throughput of the current state of the art GPU, the GP-100 [22] is 2.1x and 3.15x compared to the GK-110 [18], released in 2012.

While compute intensive applications greatly benefit from higher floating point throughput, our experiments show that several GPU applications can achieve their peak performance without utilizing all the cores on the chip. Similar observations have been made previously [2, 11, 13]. Previous authors showed that, for memory-intensive applications, fewer threads can be executed per core without sacrificing performance. Consequently, they proposed techniques that detect the minimum number of threads required to achieve peak performance.

This work is supported in part by the National Science Foundation (NSF) under awards CNS-1116810 and CCF-1149539.

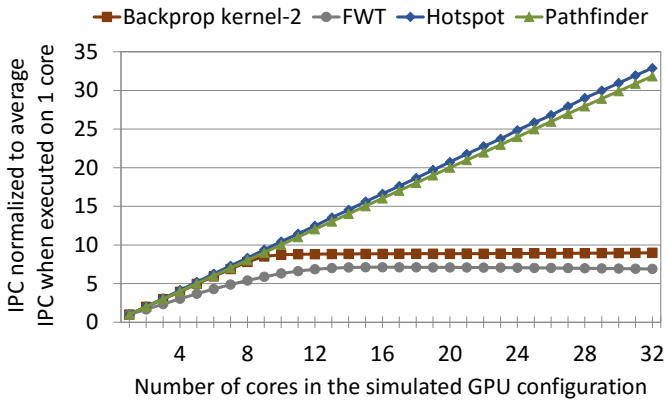


Fig. 1: The effect of number of cores on performance.

Executing fewer than the maximum threads reduces hardware resource utilization, which enables opportunities for reducing energy consumption through power-gating.

An orthogonal technique to reduce energy consumption is to execute the application on fewer cores. To observe the effect of number of cores on performance, we executed applications from the Rodinia benchmark suite and the CUDA SDK on a 32-core GPU simulated in GPGPU-Sim [3]. Fig. 1 plots the average IPC at each core count for 4 applications. Two clear performance trends can be observed. The performance of Hotspot and Pathfinder scales linearly with number of cores. Contrary to this, performance of Backprop and FWT scales linearly only for lower core counts. Beyond a certain number of cores, IPC scaling starts to decrease and performance eventually saturates. This shows that for Backprop-K2 and FWT, some of the cores can be power-gated without adversely affecting performance.

We refer to the number of active cores at which performance of an application saturates as its **optimal active core count**. The authors in [24] proposed a mechanism to detect the optimal active core count at runtime. Memory-intensive applications typically have high average memory latency due to DRAM queuing effects. Their technique marks a core as memory-sensitive if the average memory latency (sampled at runtime) is larger than an empirical threshold. The number of active cores are reduced sequentially (one per sample), until more than half the active cores are not memory-sensitive. Our

experiments show that there are two issues with this approach:
1. Reducing the number of active cores one at a time leads to a long detection time for applications which have a low optimal active core count. As all the cores consume static power during the detection period, a long detection time leads to lost opportunity for reducing energy consumption.
2. Memory latency is not a direct indicator of performance. Instead, the amount of latency overlapped by compute instructions is a better indicator. Moreover, the capability to hide memory latency varies across applications, and consequently a single threshold might not scale across newer applications. We implement the technique in [24] by directly using IPC as an indicator of performance, and comparing samples across cores instead of using a threshold. We refer to our implementation as sequential detection or Seq-Det.

To address these problems, we design ONAC: Optimal Number of Active Cores detector. ONAC uses an estimation model inspired by Roofline [25], and estimates the IPC versus core count curve described in Fig. 1. As the optimal active core count is estimated instead of searched, ONAC significantly reduces detection time and increases energy saving compared to Seq-Det. The specific contributions of this paper are:

- We thoroughly analyze the impact of number of cores on performance via case studies of two real-world applications. We show that performance saturation at a certain core count can be explained by measuring the amount of memory latency that is overlapped by computation.
- We propose a novel model to estimate the optimal number of active cores at runtime.
- We implement ONAC and Seq-Det in GPGPU-Sim [3], and analyze their effect on performance, power and energy consumption. Our evaluation shows that for memory-intensive applications, Seq-Det and ONAC reduce energy consumption by 10% and 20% respectively, with negligible impact on performance.

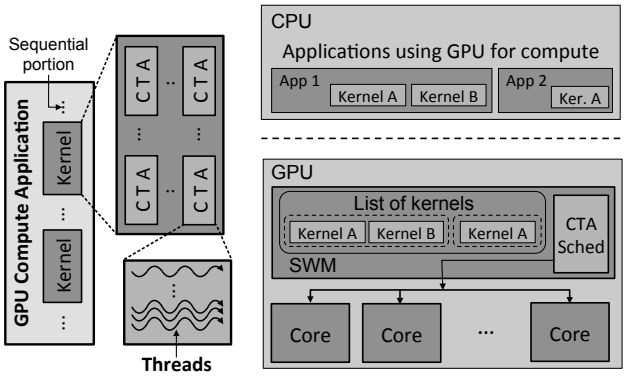
The remainder of the paper is organized as follows: Sect. 2 provides an overview of the GPU hardware architecture, Sect. 3 analyzes the impact of number of cores on application performance, Sect. 4 describes the estimation model used by ONAC and its hardware implementation, Sect. 5 studies the effect of ONAC and Seq-Det on application performance, power and energy consumption, Sect. 6 discusses related work, and Sect. 7 provides conclusions of our analysis.

II. Background

This section provides a brief overview of the GPU compute programming model and corresponding hardware architecture. Further details are available in [3, 6, 8, 18, 19, 22].

A. Programming Model

In our evaluation experiments, applications written in CUDA [19] were used. Nevertheless, our mechanism can be used as-is with applications written using other languages as well [8]. Fig. 2a depicts a highly simplified structure of a GPU compute application. The application is executed on the CPU, and portions of the computation to be offloaded to the



(a) Application (b) Hardware architecture

Fig. 2: GPGPU application hierarchy and GPU hardware architecture block diagram.

GPU are written as separate functions called kernels. Kernels use the Single Instruction Multiple Data (SIMD) programming model to capture concurrency in an algorithm. Several thousand threads typically perform the same set of computations, described by the kernel instructions, on independent pieces of data in parallel. Threads that share state are grouped into an abstraction called Cooperative Thread Arrays (CTAs). Threads from the same CTA can share data using SRAM memory on the core and synchronize using barriers. Each kernel has several thousand CTAs, collectively referred to as a grid.

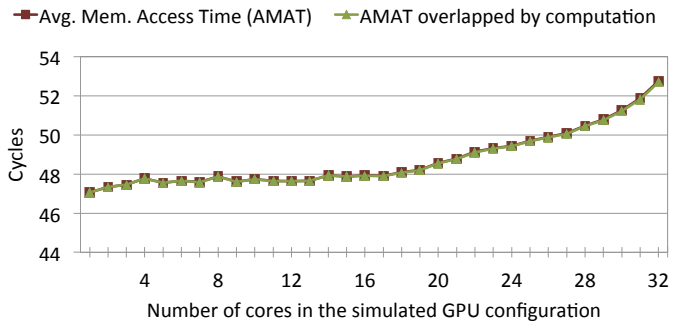
B. Overview of Scheduling

Fig. 2b depicts how the programming model abstractions described in the previous subsection are mapped to the GPU hardware architecture. At a high level, the GPU chip consists of several in-order SIMD cores called Streaming Multiprocessors (SMs). The SM Work Manager (SWM) has a list of kernels launched from the application. The SWM selects a kernel[1] and a hardware unit, referred to as the CTA scheduler, issues work from the selected kernel to the SMs.

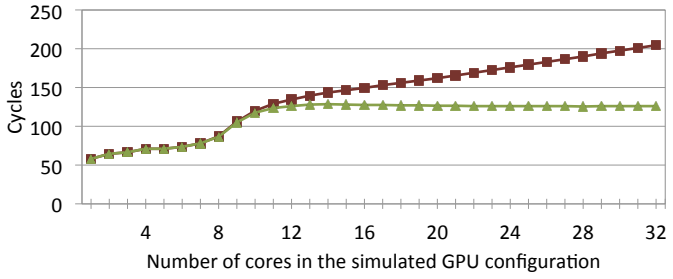
As threads within a CTA can use synchronization primitives, work is issued to SMs at the CTA granularity. A typical scheduling policy used by the CTA scheduler is round-robin. For example, CTA 1 is issued to SM 1, CTA 2 to SM 2, and so on. The maximum number of CTAs executing simultaneously on an SM depends on the number of registers and shared memory used per thread. The default policy used by the SWM is to issue CTAs to an SM until one of the hardware resources is exhausted. In this work, we show that for memory-intensive kernels, the SWM can selectively issue CTAs to fewer cores and enable opportunities for reducing energy consumption.

Within each SM, threads are executed in groups of 32 called warps. Warp scheduling policies have been extensively studied in the academic research community and several optimizations have been proposed [1, 16, 23]. In the remainder of this this paper, we will focus on scheduling at the CTA granularity only. We assume that only one kernel is active and the warp scheduling policy used is Greedy Then Oldest (GTO).

[1]Priority among concurrently launched kernels can be set by the driver or a hardware unit upstream of SWM.

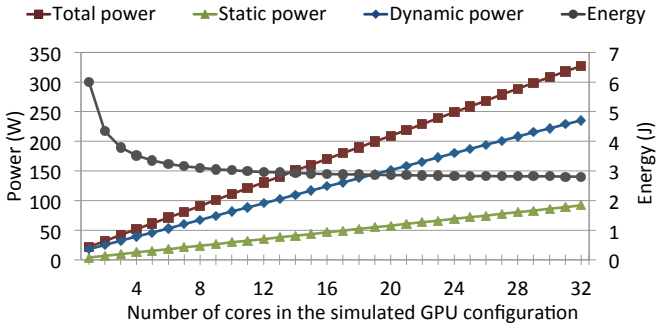


(a) Pathfinder

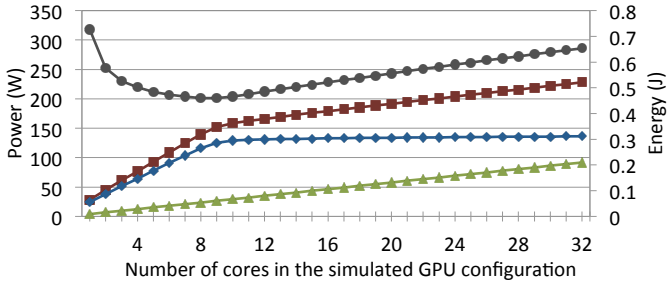


(b) Backprop

Fig. 3: Average Memory Access Time (AMAT) and the amount of AMAT overlapped by computation.



(a) Pathfinder



(b) Backprop

Fig. 4: The effect of N_{active} on power and energy.

III. Effect of Number of Cores on Performance

In this section we analyze the effect of the number of active cores (N_{active}) on performance for kernels from two applications: Pathfinder and Backprop. The Pathfinder kernel is compute-intensive, and obtains peak performance when all cores on the GPU are utilized. On the contrary, the Backprop kernel is memory-intensive, and its performance saturates after a specific number of active cores.

A. Effect on Memory Latency and Performance

The fundamental effect of N_{active} on performance can be deduced by analyzing its effect on non-overlapped memory latency (Fig. 3). Average memory access time (AMAT) is typically defined as the average memory latency observed by a memory instruction, irrespective of whether it hits in the L1 or L2 data caches. Fig. 3 plots the AMAT observed for the Backprop and Pathfinder kernels, at different core counts, normalized to the run on a configuration with just one core. As expected, as N_{active} increases, the bandwidth available to each core reduces, and consequently AMAT increases. Notice in the figure that AMAT increases for both kernels. Thus, by itself, AMAT is not an indicator of performance.

Fig. 3 also plots the amount of AMAT that is overlapped by computation ($AMAT_{overlapped}$). $AMAT_{overlapped}$ is the average number of cycles, for each memory request, when there is at least one arithmetic instruction in flight on its requesting core. Observe in Fig. 3a that for Pathfinder, $AMAT_{overlapped}$ increases with AMAT. Consequently, performance of Pathfinder keeps scaling until N_{active} equals 32. On the contrary, for the Backprop kernel, the difference between AMAT and $AMAT_{overlapped}$ starts to increase after N_{active} is higher than 9 (Fig. 3b). This indicates that an increasing amount of memory latency is exposed and contributes to the execution time of each core. Thus, the performance of Backprop starts to plateau around this point and eventually saturates when N_{active} reaches 12 (refer to Fig. 1). The threshold used to categorize a kernel's performance as saturated is when it is within 2% of the performance when N_{active} equals 32.

B. Effect on Power and Energy

Fig. 4 plots the static and dynamic power consumed by the two kernels with a varying number of active cores. While static power increases linearly with N_{active}, dynamic power consumption scales proportionally to IPC. As discussed in the previous subsection, Pathfinder's IPC continues to scale up until 32 cores. Consequently, dynamic power continues to increase as well. On the contrary, Backprop spends an increasing percentage of the total execution time on non-overlapped memory latency as N_{active} increases beyond 9 cores. As cores stay idle waiting for data for a larger portion of their execution time, the average dynamic power consumed per core reduces, and becomes almost constant after N_{active} is higher than 12 cores (Fig. 4b).

The energy consumed by a kernel is directly proportional to average power consumption and inversely proportional to IPC. Hence as N_{active} increases, if the increase in IPC is larger than the increase in power consumption, total energy consumption reduces. Notice in Fig. 4a that although the average power consumption of the Pathfinder kernel continues to increase until N_{active} = 32, energy consumption continues to decrease. This is due to a steady increase in IPC. On the contrary for Backprop (Fig. 4b), as the average power consumption

continues to increase (due to increase in static power), and IPC saturates after N_{active} = 12, the total energy increases.

IV. ONAC: Estimation Model and Hardware Implementation

In the previous section, we showed that some GPU kernels can achieve peak performance without using all the cores on the chip. The unused cores can be power gated to reduce static power consumption, and thereby energy. In this section we describe ONAC, our detection technique which detects the optimal core count at runtime. At its crux is our novel estimation model that significantly reduces the detection overhead.

A. Performance Estimation Model

Our estimation model leverages the following observations from the IPC trends described in the previous sections:
Observation 1: IPC achieved when all cores are used (IPC_{all}), is the maximum IPC of the kernel on the current configuration.
Observation 2: Let IPC_1 be the IPC achieved on the 1 core configuration. The IPC at a given core count N, is lower than or equal to N * IPC_1.
Observation 3: As the number of cores increase, IPC per core remains the same or decreases. For a configuration with x cores, let the average IPC per core be $IPC_{slope\,x}$. For any i $>$ j, $IPC_{slope\,i} \leq IPC_{slope\,j}$ (Fig. 5).

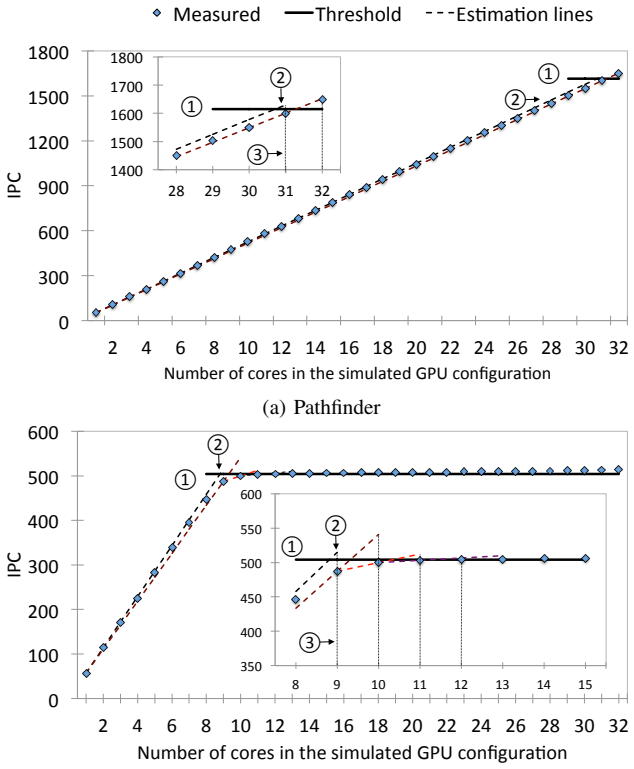
In the next subsection, we describe how our model uses these observations and illustrate the steps it takes to estimate the optimal active core count.

B. Optimal Core Detection Examples

At the beginning of a kernel's execution, a sample of IPC with all the cores active is taken (IPC_{all}). Following **observation 1**, our model assumes IPC_{all} to be the maximum achievable IPC for this kernel. Its objective is to find the minimum number of cores required to achieve an average IPC within a certain threshold of IPC_{all}. We refer to this as $IPC_{threshold}$, and is depicted by a solid line in Fig. 5 (1).

Next, all but 1 cores are put in a *paused* state and a sample of the IPC is taken with just 1 core active. The details of pausing cores and sampling are described in the next section. Following **observation 2**, the maximum performance at each core count can be projected by a line through the origin with a slope of IPC_1 (2). The intersection of this line with the constant line through $IPC_{threshold}$ is the least number of cores required to achieve peak IPC. This gives the model a first estimate of the optimal number of active cores. The first estimate calculated for the Pathfinder and Backprop kernels is 31 and 9 cores respectively (refer to Fig. 5a and Fig. 5b (3)).

An estimate of the optimal active core count is referred to as N_{est}. At this point, a sample of the average IPC is taken with N_{est} cores active (IPC_{est}). If the IPC_{est} is greater than or equal to $IPC_{threshold}$, the model concludes that the optimal number of cores has been detected. As expected, the first estimate is very optimistic. The IPC of Backprop with 9 cores and Pathfinder with 31 cores do not meet the required performance threshold.



(a) Pathfinder

(b) Backprop

Fig. 5: Example of optimal active core count detection using our model for the Backprop and Pathfinder kernels.

All the remaining estimates are calculated using **observation 3**. The model stores two values: a sample of the IPC at the current (IPC_{cur}) and previous (IPC_{prev}) estimates. The next estimate is calculated as the point at which a line passing through IPC_{prev} and IPC_{cur} intersects with $IPC_{threshold}$. For Backprop, a line through IPC_1 and IPC_9 intersects $IPC_{threshold}$ at 10 cores. The IPC sampled at 10 cores is still lower than $IPC_{threshold}$, and a new estimate is calculated using IPC_9 and IPC_{10}. In this way, the model keeps updating IPC_{cur} and IPC_{prev} until the IPC at the current estimate satisfies $IPC_{threshold}$. Our model converges at the fourth estimate for Backprop, and correctly detects the optimal core count as 12 cores. For Pathfinder, the optimal active core count is correctly detected as 32 cores at the second estimate.

C. Hardware Implementation

We implemented ONAC inside the SM Work Manager described in Sect. 2b. Fig. 6 illustrates a simplified block diagram. The unit inside the SWM which detects the optimal core count is shown as the ONAC control logic.

1) Core Status Table: The baseline SWM stores information required for scheduling in a structure referred to as the Core Status Table (CST). We add three new fields per core to this structure: active bit, paused bit and a field to record the IPCs sampled at runtime. CTAs are issued only to cores which have the active bit set and paused bit unset.

2) ONAC Control Logic: The control logic in Fig. 6 illustrates a state machine representation of the detection algorithm.

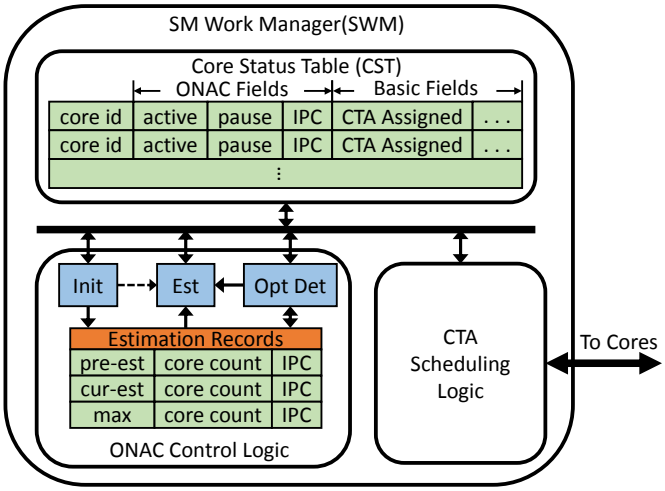


Fig. 6: Block diagram of ONAC's hardware implementation.

The Estimation Records (ER) table has three entries each at the previous estimate, current estimate and all cores active, to store the core count and average IPC. The state machine has three states:

Init: At the beginning of a kernel, all cores are active and unpasused. Each core takes a sample of its IPC and stores it in the CST (refer to Sect. IV-C3). When all the cores have taken their samples, the IPC is summed and stored in the ER. Next, the pause flag is set for 31 cores (refer to Sect. IV-C4). Once all cores are paused, a sample of IPC with one core is taken and Init invokes the **Est** logic to calculate the first estimate.

Est and Opt Det: The detection state machine stays in either Est or the Opt Det states for the remaining portion of the kernel. **Est** calculates the first estimate (N_{est}) using the samples of the IPC at max and 1, and stores it in the cur-est field of the ER. To get a sample of the IPC, N_{est} - 1 cores are unpaused by resetting their pause bits. Once the sample with N_{est} active cores is collected, **Opt Det** compares it with IPC for max and checks if the performance threshold is satisfied. If true, N_{est} is set as the optimal number of active cores and the detection process is terminated. If false, pre-est and cur-est entries are updated and **Est** calculates a new estimate.

3) Sampling: At the start of a sample, the SWM sends a request to all cores that are active but not paused. The requested cores wait for the instructions in-flight to complete, and then take a sample of the IPC over the next sampling period. Our experiments show that sampling period is a significant factor that affects the accuracy of the IPC samples, and consequently the accuracy of detection.

Sampling period: The total number of CTAs concurrently active on the GPU is referred to as a CTA wave. GPU compute kernels typically execute a similar set of instructions on a grid of input data, organized into CTAs. Consequently, the total computation performed by the kernel can be approximated as computations performed by a sequence of CTA waves. However, as all CTAs in a wave do not start and finish at the same time, the IPC fluctuates over time, over execution of a CTA wave. Our experiments show that the average over a set of four CTA waves is a good sampling period and results in a stable IPC value that is close to the kernel's average IPC.

TABLE I: The GPU configuration used in our experiments

Number of Cores	32
Warp Size	32
Warp schedulers per core	2, GTO scheduling policy
Execution units per core	32 ALUs, 4 SFUs,16 LD/ST units
Resources/Core	Max. 48 warps, Max. 8 thread blocks, 32768 Registers, 48KB Shared Memory
Core/ICNT/Memory Clock	1300MHz/1300MHz/1848MHz
Number of Mem. Partitions	12
DRAM Chip Model	32bits bus width/Memory Partition, 6 Banks/Memory Partition, GDDR5 timing
Processing Power	Single precision: Max. 2662.4 GFLOPs
Memory Bandwidth	Max. 177.4 GB/s

4) Core Pausing: During the detection period, the cores that are inactive are put in a paused state. To pause a core, the SWM sends a pause request, and the warp scheduler on the core stops issuing instructions. Once the instructions that were in-flight are completed, the core sends an acknowledgment to the SWM. Once all the required cores have paused, the SWM notifies the active cores to begin the next sample. We pause cores at the beginning to take a sample with 1 core. For the remaining portion, cores are unpaused and put back into the active state as the estimate of optimal core count increases. Pausing the cores instead of draining them, helps reduce the time required to wait before beginning the next sampling period. After the optimal core count is detected, the cores that are paused, are drained and power-gated.

V. Experimental Results

Two metrics are important in the design of a runtime technique for detecting optimal active core count (N_{opt}):
1: Accuracy: The accuracy of detection is important as overestimating N_{opt} reduces the amount of energy saved, while underestimating it negatively impacts performance.
2: Detection Time: A short detection time is important as it increases the amount of energy saved for memory-intensive kernels. Moreover, a long detection time can negatively impact performance, particularly for compute-intensive kernels.

In this section, we evaluate ONAC and the sequential detection technique (Seq-Det) on the basis of accuracy, detection time and their impact on performance, power and energy.

A. Methodology

We implemented ONAC and Seq-Det in GPGPU-Sim, a cycle level GPU architecture simulator [3]. The simulator was configured to match the ratio of peak computational throughput to peak memory bandwidth of an NVIDIA Tesla K40 GPU. Table I provides more details of the simulated GPU configuration.

The applications used for our evaluation were chosen from Rodinia [4], an open source benchmark suite for heterogeneous computing and the CUDA SDK [20]. The set of kernels used for our analysis is listed in Table II. We broadly group kernels into two categories. Kernels grouped under **type A** have N_{opt}

TABLE II: Benchmark Application Kernels

Kernel Name	Abbr.	Suite	Type	Total CTAs
LU Decomposition	LUD	Rodinia	A	65025
K-Nearest Neighbor	NN	Rodinia	A	50115
K-Means	KM	Rodinia	A	25600
B+tree Kernel 2	BT-K2	Rodinia	A	65535
Back Propagation kernel 2	BP-K2	Rodinia	A	65535
Speckle Reducing Anisotropic Diffusion	SRAD	Rodinia	A	32768
Discrete Cosine Transform kernel 1	DCT-K1	CUDA SDK	A	32768
Discrete Cosine Transform kernel 2	DCT-K2	CUDA SDK	A	32768
Transpose No Bank Conflicts	TP-K1	CUDA SDK	A	65536
Transpose Coarse Grained	TP-K2	CUDA SDK	A	65536
Fast Walsh Transform	FWT	CUDA SDK	A	32768
Convolution Separable	CVSEP	CUDA SDK	A	16384
Merge Sort	MS	CUDA SDK	A	24567
Pathfinder	PATH	Rodinia	B	46297
B+tree Kernel 1	BT-K1	Rodinia	B	65535
Hotspot	HS	Rodinia	B	29241
Back Propagation kernel 1	BP-K1	Rodinia	B	65535
Convolution Texture Kernel 1	CVT-K1	CUDA SDK	B	98304
Convolution Texture 6Kernel 2	CVT-K2	CUDA SDK	B	98304
DXT Compression	DXTC	CUDA SDK	B	16384

less than 32, and consequently are good candidates for saving energy, while **Type B** kernels have N_{opt} equal to 32.

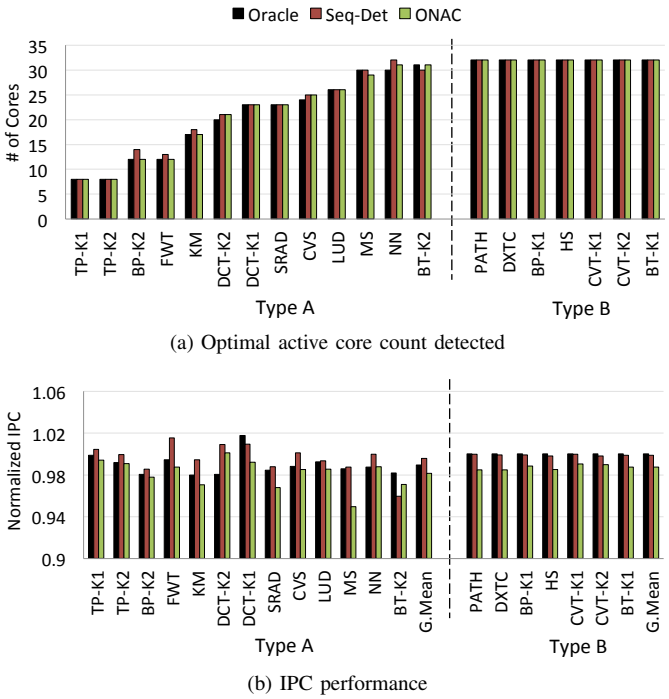
B. Accuracy of Detection and Performance

We detected the optimal number of cores for each kernel by executing them separately at each core count. We refer to this as N_{oracle}. Fig. 7a compares the N_{opt} detected by ONAC and Seq-Det to N_{oracle}. As expected, the N_{opt} detected at runtime has an impact on the kernel's performance. Consequently in Fig. 7b, we compare the performance achieved by kernels when executed with ONAC and Seq-Det, to when executed on N_{oracle} cores. The results are normalized to the performance achieved on the baseline configuration (referred to as N_{max}).

1) Type A Kernels: 13 of the 20 kernels analyzed in our experiments were type A kernels. Observe in Fig. 7a that the N_{opt} detected by Seq-Det is always higher than or equal to N_{oracle} for all of them, except BT-K2. Consequently, notice in Fig. 7b that the IPC achieved by Seq-Det is always higher than the performance threshold of 2% for all kernels except BT-K2, where N_{opt} is detected inaccurately due to a sampling variation. The sampled IPC taken with 30 cores active, is higher than the average IPC achieved when the kernel is executed on 30 cores, which leads to Seq-Det underestimating N_{opt}.

For ONAC, the N_{opt} value detected is higher than or equal to N_{oracle} for all type A kernels except MS. This detection inaccuracy is caused by a similar sampling variation problem. Consequently, the IPC achieved by ONAC is 95% of N_{max} for MS, and is close to the performance threshold for others (refer to Fig. 7b). On average Seq-Det and ONAC achieve 99% and 98% of the performance achieved with N_{max} cores. On the other hand, we show in Sect. V-C that they reduce the average energy consumption across kernels by 10% and 20% respectively.

2) Type B Kernels: In addition to detection accuracy, detection time also has an effect on performance. Notice in Fig. 7a that both techniques, Seq-Det and ONAC, detect N_{opt} with



(a) Optimal active core count detected

(b) IPC performance

Fig. 7: Optimal number of active cores (N_{opt}) detected by ONAC and Seq-Det, and the corresponding IPC achieved by the kernels. IPC results are normalized to the IPC achieved when the kernels are executed with all the cores active.

100% accuracy for all type B kernels. However, observe in Fig. 7b that the IPC achieved by them varies across kernels. As Seq-Det reduces the number of active cores one at a time, it takes one sample at 31 active cores, and switches back to 32. Consequently, the IPC achieved by Seq-Det for all type B kernels is close to that achieved by the baseline. On the other hand, ONAC has to take a sample with one active core, which causes the small impact on performance seen in the figure.

Although ONAC has this overhead for type B kernels, the effect on performance is less than 2% on average. On the other hand, ONAC saves significantly more energy compared to Seq-Det for type A kernels. We analyze the correlation between detection time and energy in detail in the next section. Notice in Fig. 7b that the effect of detection time on IPC is also observed in a few type A kernels. Although ONAC detects N_{opt} accurately for the BP-K2, KM, SRAD and BT-K2 kernels, the IPC achieved is a little below the 2% threshold.

C. Detection Time and its Effect on Power and Energy

In this section, we analyze the effect of ONAC and Seq-Det's detection time on average power and energy consumption. Fig. 8 plots the ratio of detection time to the total execution time, while Fig. 9a and Fig. 9b plot the average power and total energy consumed by the kernels. In summary, ONAC and Seq-Det reduce the energy consumption of type A kernels by 20% and 10% respectively, as compared to using all the cores on the chip, with very insignificant overhead on performance. The higher energy saving for ONAC, compared

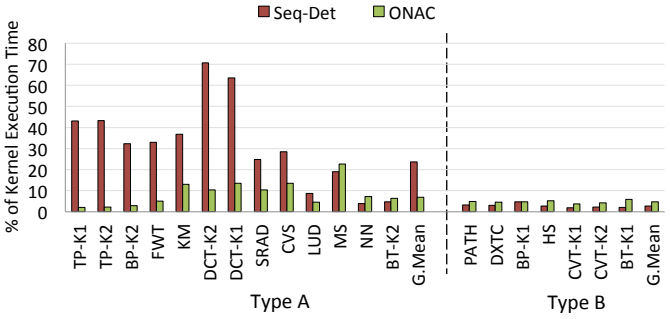
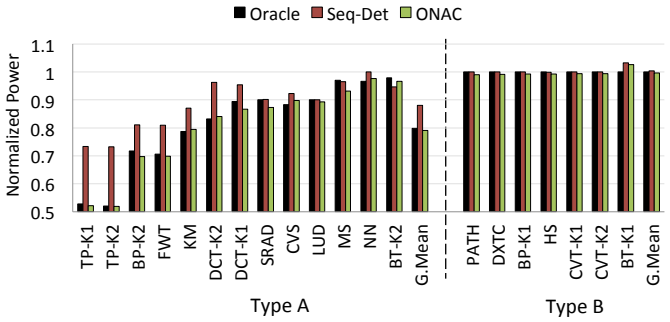
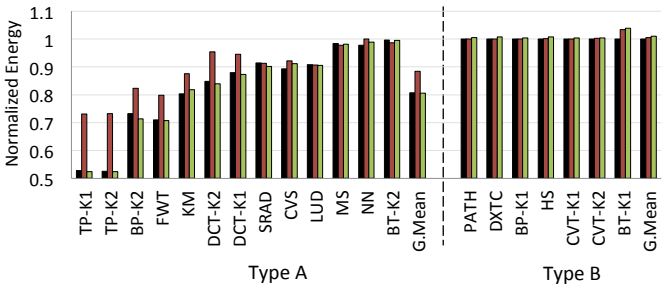


Fig. 8: The percentage of kernel's total execution time spent on detection by ONAC and Seq-Det.



(a) Average power consumption



(b) Total energy consumption

Fig. 9: Average power and total energy consumptions of the kernels when executed with oracle number of cores, Seq-Det and ONAC. The results are normalized to the power and energy consumption when executed with all the cores active.

to the sequential detection technique, comes from reducing the detection time by 45% on average across the type A kernels.

1) Type A kernels: The kernels in Fig. 8 and Fig. 9 are sorted in ascending order of the optimal number of active cores. The optimal core count of type A kernels ranges from 8 (transpose kernels) to 31 (B+Tree K2). Observe in Fig. 8 that as N_{opt} increases, the ratio of execution time spent on detection by Seq-Det decreases. The spikes in this trend for the DCT and MS kernels are because the total execution time of the kernels is relatively short. Hence, although the detection time of Seq-Det for the DCT kernels is lesser than the KM, FWT, BP-K2 and TP kernels, the ratio is higher. For ONAC, notice that the detection overhead increases as N_{opt} increases. This is because ONAC starts the estimation with 1 core, and increases the number of active cores as it converges to N_{opt}.

For all type A kernels, except MS, NN and BT-K2, Seq-Det takes longer to detect the optimal core count compared to ONAC. Its effect on the average power consumption can be clearly observed in Fig. 9a. For all kernels to the left BT-K2, the average power consumption with Seq-Det is higher compared to ONAC. The difference is larger for lower core counts. As N_{opt} increases, the detection time of Seq-Det decreases and the difference reduces, until they become comparable for the SRAD, CVS and LUD kernels.

With lower power consumption and comparable performance, the energy consumption with ONAC is lower than Seq-Det when N_{opt} is low. Similar to power, the difference reduces as the optimal core count increases. ONAC consumes a bit more energy compared to Seq-Det for the MS and BT-K2 kernels. Notice that for the MS kernel ONAC consumes lesser power compared to Seq-Det. The loss in IPC is more than the reduction in power consumption, causing the energy consumption to be higher.

2) Type B kernels: As kernels in the type B category have the optimal number of active cores as 32, the detection times for both ONAC and Seq-Det are modest (less than 4% of the execution time). Consequently, the power consumption of both ONAC and Seq-Det is similar to that of N_{max}. As ONAC takes a sample with 1 active core, it has a small impact on the IPC (refer to Fig. 7b). Consequently, ONAC increases the energy consumption of type A kernels by 2% on average.

VI. Related Work

A. Parallelism optimization on GPU

Kayıran et al. [11] observe that executing the maximal number of CTAs on GPU cores doesn't always achieve the best performance. They propose a mechanism to detect the optimal number of CTAs based on sampling internal data of the pipeline. When the sampled idle time is lower than a threshold and the sampled memory stall time falls between lower and upper bounds, the optimal number of CTAs is captured. The scheme developed by Lee et al. [13] leverages the behavior of greedy warp scheduler. Their scheme measures the number of instructions issued, and uses the ratio of instructions issued by the greediest CTA to instructions issued by all CTAs on the core as the optimal number of CTAs.

Besides modulating the number of CTAs per core, there are other papers focused on parallelism optimization at warp-level. Gebhart et al. [5] propose a warp scheduler that groups warps into two priorities and only issues instructions from the higher priority warps. Their evaluations demonstrate that significant energy saving can be achieved by power-gating unused in-core resources. Rogers et al. [23] adopt a cache-locality scoring system to throttle the number of warps adaptively. They show their mechanism can improve the performance for highly cache-sensitive workloads by reducing cache thrashing.

B. Energy-efficient computing on GPU

Although several works [10, 14] have explored thread-level and core-level parallelism optimizations for energy savings in the context of chip multiprocessors (CMP), these prior works

cannot be applied to GPU directly due to two distinct differences between CMP and GPU platforms: (1) The overhead of starting and shutting down a core on a GPU platform is much higher than on a CMP platform. (2) Also, the algorithms for CMP platform typically require operating system support. Thus, the energy optimizations specific to GPU platform are introduced.

In [9] Jiao et al. characterize the performance and power consumption of various GPU compute kernels at different GPU core and DRAM frequencies, and show their effect on performance and power consumption. In [12], Lee et al. adjust the number of cores and modulate the core's voltage and frequency to improve throughput under power constraints. While their objective is optimizing performance under given power constraints, we focus on optimizing power consumption under given performance constraints. In [15], Lin et al. utilize software prefetching and dynamic voltage scaling to achieve two objectives: energy optimization under performance constraints and performance optimization under power constraints.

In [7] Hong et al. propose an analytical model to predict power, performance and the optimal number of cores based on kernel's static information. They optimize for performance per watt, which has a side effect of losing performance. Our work is closest to that of Song et al. in [24]. The propose to sample memory latency to each core, and reduce the number of active cores one at a time, until the average latency is lower than an empirically found threshold. Our implementation of their technique is referred to as sequential detection or Seq-Det in this paper, and we have thoroughly compared the efficiency of both approaches on different types of compute kernels.

VII. Conclusion

In this work we show that certain GPU compute kernels can achieve peak performance without utilizing all the cores on the chip. For such applications, detecting the optimal number of active cores can enable energy savings by power-gating the unused cores. Using detailed analysis of two application kernels, we demonstrated the effect of number of active cores on performance, power and energy consumption. To this effect we design a mechanism to detect the optimal core count at runtime with high accuracy and low detection overhead. We implement the mechanism in a cycle level GPU simulator and analyze its efficiency compared to a sequential technique. Our results show that our mechanism reduces the detection time as much as 45% compared to the sequential detection technique. It reduces energy consumption by 20% on average compared to the baseline, with less than 2% impact on performance.

References

[1] M. Awatramani, X. Zhu, J. Zambreno, and D. Rover, "Phase aware warp scheduling: Mitigating effects of phase behavior in gpgpu applications," in *Proc. of The 24th Int. Conf. on Parallel Architecture and Compilation Techniques*, 2015, pp. 1–12.

[2] M. Awatramani, J. Zambreno, and D. Rover, "Perf-Sat: Runtime detection of performance saturation for GPGPU applications," in *Proc. of the 43rd Int. Conf. on Parallel Process. Workshops*, 2014, pp. 1–8.

[3] A. Bakhoda, G. L. Yuan, W. W. Fung, H. Wong, and T. M. Aamodt, "Analyzing CUDA workloads using a detailed GPU simulator," in *Proc. of the 2009 IEEE Int. Symp. on Performance Anal. of Syst. and Software*, 2009, pp. 163–174.

[4] S. Che *et al.*, "Rodinia: A benchmark suite for heterogeneous computing," in *Proc. of the 2009 IEEE Int. Symp. on Workload Characterization*, 2009, pp. 44–54.

[5] M. Gebhart *et al.*, "Energy-efficient mechanisms for managing thread context in throughput processors," in *Proc. of the 38th Annu. Int. Symp. on Computer Architecture*, 2011, pp. 235–246.

[6] P. N. Glaskowsky, "NVIDIA's Fermi: the first complete GPU computing architecture," White Paper, Nvidia, 2009. [Online]. Available: http://www.nvidia.com/content/PDF/fermi_white_papers/P.Glaskowsky_Nvidia's_Fermi-The_First_Complete_GPU_Architecture.pdf

[7] S. Hong and H. Kim, "An integrated GPU power and performance model," in *Proc. of the 37th Annu. Int. Symp. on Computer Architecture*, vol. 38, no. 3, 2010, pp. 280–289.

[8] L. Howes and A. Munshi. (2015) The OpenCL specification. Khronos OpenCL Working Group. [Online]. Available: https://www.khronos.org/registry/cl/specs/opencl-2.1.pdf

[9] Y. Jiao, H. Lin, P. Balaji, and W. Feng, "Power and performance characterization of computational kernels on the GPU," in *Proc. of 2010 IEEE/ACM Int. Conf. on Green Comput. and Commun. & Int. Conf. on Cyber, Physical and Social Comput.*, 2010, pp. 221–228.

[10] C. Jung, D. Lim, J. Lee, and S. Han, "Adaptive execution techniques for smt multiprocessor architectures," in *Proceedings of the 10th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, 2005, pp. 236–246.

[11] O. Kayıran, A. Jog, M. T. Kandemir, and C. R. Das, "Neither more nor less: Optimizing thread-level parallelism for GPGPUs," in *Proc. of the 22nd Int. Conf. on Parallel Architectures and Compilation Techniques*, 2013, pp. 157–166.

[12] J. Lee, V. Sathisha, M. Schulte, K. Compton, and N. S. Kim, "Improving throughput of power-constrained GPUs using dynamic voltage/frequency and core scaling," in *Proc. of the 20th Int. Conf. on Parallel Architectures and Compilation Techniques*, 2011, pp. 111–120.

[13] M. Lee *et al.*, "Improving GPGPU resource utilization through alternative thread block scheduling," in *Proc. of 20th IEEE Int. Symp. on High Performance Computer Architecture*, 2014, pp. 260–271.

[14] J. Li and J. F. Martinez, "Dynamic power-performance adaptation of parallel computation on chip multiprocessors," in *Proceedings of the 12th IEEE International Symposium on High-Performance Computer Architecture*, 2006, pp. 77–87.

[15] Y. Lin, T. Tang, and G. Wang, "Power optimization for GPU programs based on software prefetching," in *Proc. of the 10th Int. Conf. on Trust, Security and Privacy in Comput. and Commun.*, 2011, pp. 1339–1346.

[16] V. Narasiman, M. Shebanow, C. J. Lee, R. Miftakhutdinov, O. Mutlu, and Y. N. Patt, "Improving GPU performance via large warps and two-level warp scheduling," in *Proc. of the 44th Annu. IEEE/ACM Int. Symp. on Microarchitecture*, 2011, pp. 308–317.

[17] J. Nickolls and W. J. Dally, "The GPU computing era," *IEEE Micro*, vol. 30, no. 2, pp. 56–69, 2010.

[18] "NVIDIA Kepler GK110," White Paper, NVIDIA Corp., 2012. [Online]. Available: http://www.nvidia.com/content/PDF/kepler/NVIDIA-Kepler-GK110-Architecture-Whitepaper.pdf

[19] (2015) CUDA C programming guide. NVIDIA Corp. [Online]. Available: http://docs.nvidia.com/cuda/pdf/CUDA_C_Programming_Guide.pdf

[20] (2015) NVIDIA CUDA SDK. NVIDIA Corp. [Online]. Available: https://developer.nvidia.com/cuda-downloads

[21] (2015) NVIDIA Developer Zone. NVIDIA Corp. [Online]. Available: https://developer.nvidia.com

[22] "NVIDIA Tesla P100," White Paper, NVIDIA Corp., 2016. [Online]. Available: https://images.nvidia.com/content/pdf/tesla/whitepaper/pascal-architecture-whitepaper.pdf

[23] T. G. Rogers, M. O'Connor, and T. M. Aamodt, "Cache-conscious wavefront scheduling," in *Proc. of the 45th Annu. IEEE/ACM Int. Symp. on Microarchitecture*, 2012, pp. 72–83.

[24] S. Song, M. Lee, J. Kim, W. Seo, Y. Cho, and S. Ryu, "Energy-efficient scheduling for memory-intensive GPGPU workloads," in *Proc. of Design, Automation and Test in Europe Conf. and Exhib. 2014*, 2014, pp. 1–6.

[25] S. Williams, A. Waterman, and D. Patterson, "Roofline: an insightful visual performance model for multicore architectures," *Commun. of the ACM*, vol. 52, no. 4, pp. 65–76, 2009.