Journal of Systems Architecture 89 (2018) 73–83

Contents lists available at ScienceDirect

Journal of Systems Architecture

journal homepage: www.elsevier.com/locate/sysarc

# Application aware Scalable Architecture for GPGPU

Winnie Thomas*, Rohin D. Daruwala

*Department of Electrical Engineering, V. J. Technological Institute, Mumbai 400019, India*

ARTICLE INFO

*Keywords:*
GPGPU
CTA scheduler
CUDA
Bandwidth utilization
Instructions per cycle
Branch divergence

ABSTRACT

Modern General Purpose Graphic Processing Units (GPGPU) offer high throughput for parallel applications with their hundreds of integrated cores. However, there are applications that experience performance saturation and even degradation with increasing number of cores. At present the scheduler in the GPU hardware allocates all the available resources to maximize their utilization. We observed that applications have preference towards specific set of resources. The utilization of other redundant resources can reduce the throughput of the applications. To overcome this problem, in this paper we first classify the applications into two types; type-I that dominantly require processing cores and type-II that rely on the performance of the memory-system. We propose an Application aware Scalable Architecture (ApSA) for GPGPU based on classified applications which performs run-time tailoring of the GPU resources to present an optimal set of resources to the running application. The results are analyzed and compared in terms of instructions per cycle, bandwidth utilization and branch divergence. We found that if the application is identified to be of type-I with the proposed technique the average profiling overhead is 1.6%. Type-II applications experience average profiling overhead of 1.15%. The average power saved by clock-gating redundant resources in the case of type-II applications is 20.08%.

## 1. Introduction

The General Purpose Graphic Processing Units (GPGPUs) are emerging as a preferred high performance computing platform for various general purpose parallel applications over conventional multi-core CPUs [6,27]. This preference to GPGPUs is due to their architecture, designed for high-throughput data-parallel applications [32].The hardware threads in a GPGPU process single instruction on different data elements at a time which is the underlying principle of Single Instruction Multiple Threads (SIMT) architecture [2,3,19]. With the advent of programming models like CUDA [25,38] and OpenCL [34], programming a GPGPU is no more a cumbersome task.

However, it is seen that majority of the GPGPU applications are bottlenecked by the available compute resources and memory bandwidth [15,33]. The GPU hardware is primarily optimized for highly compute-intensive applications. Hence the GPGPUs allot resources to the applications to its maximum capacity to mitigate resource under-utilization. Today many general-purpose applications are fine-tuned or re-written to be executed on the GPUs. A general purpose application that is not so compute-intensive may not get benefit due to the maximum utilization of the GPGPU hardware. We observed as shown in Fig. 1 that with the increasing core counts, some applications exhibit linear improvement in the performance, whereas other applications don't. In this paper we attempt to address these issues and propose an Application aware Scalable Architecture.

We present an Application aware Scalable Architecture (ApSA) for GPGPU which identifies the nature of the application in terms of resource requirements and accordingly performs run-time tailoring. GPGPU applications are typically divided into several kernels, where each kernel is capable of launching many threads [24]. To assign the threads to execution resources, they are divided into thread blocks also known as Cooperative Thread Arrays (CTAs) [19]. The group of threads in a CTA cooperates with each other by synchronizing their execution. There are absolutely no dependencies among the threads of different CTAs which facilitate the execution of CTAs in random order. This makes today's GPU application scalable and leads to a reliable execution of the application even on scaling up or down the number of shader scores. The existing CTA schedulers always allocate maximum possible number of CTAs in a core depending on the available resources on core [14]. Large number of CTAs per core improves performance by hiding long latency operations [35].

A GPGPU exploits parallelism in two ways primarily. Each core of a GPGPU can simultaneously execute many, but fixed number of threads. The threads within a CTA are grouped and executed in the form of warps each of 32 threads (NVIDIA) or wavefronts each of 64 threads (AMD).The second way in which a GPU exploits parallelism is by

* Corresponding author.
*E-mail addresses:* winnie.vjti@gmail.com (W. Thomas), rddaruwala@el.vjti.ac.in (R.D. Daruwala).

https://doi.org/10.1016/j.sysarc.2018.07.003
Received 16 October 2017; Received in revised form 8 July 2018; Accepted 24 July 2018
Available online 26 July 2018

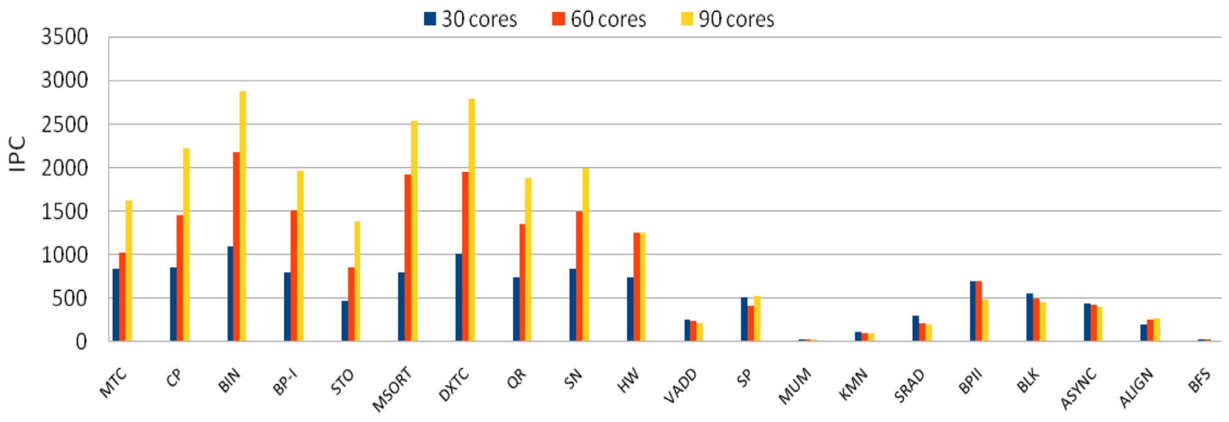


**Fig. 1.** Effect on performance with different number of cores in GPGPU with 8 memory partitions.

**Table 1**
Application description.

| Name | Abr. | #of Kernels | Type |
|---|---|---|---|
| Monte Carlo Option Pricing [28] | MTC | 2 | I |
| Coulombic Potential [1] | CP | 1 | I |
| Binomial Options [28] | BIN | 1 | I |
| Back propagation (2 kernels) [5] | BP-I | 1st kernel | I |
| | BPII | 2nd kernel | II |
| Store GPU [1] | STO | 1 | I |
| Merge Sort [28] | MSORT | 1 | I |
| HQ DXT Compression [28] | DXTC | 1 | I |
| Quasi Random Generator [28] | QR | 1 | I |
| Sorting Networks [28] | SN | 1 | I |
| Heart Wall [5] | HW | 2 | I |
| Vector Addition [28] | VADD | 1 | II |
| Scalar Product [28] | SP | 1 | II |
| MUMmerGPU [1] | MUM | 1 | II |
| *k*-means [5] | KMN | 2 | II |
| SRAD v2 [5] | SRAD | 2 | II |
| Blackscholes Option Pricing [28] | BLK | 3 | II |
| Async API [28] | ASYNC | 1 | II |
| Aligned Types [28] | ALIGN | 1 | II |
| Breadth First Search [1] | BFS | 8 | II |

**Table 2**
Architectural configurations.

| On-chip Components | |
|---|---|
| # of Shader cores | 60 (in ApSA), 650 MHz each |
| Warp size | 32 |
| Resource per core | Max.1024 threads,16 kB shared memory, 16,384 registers |
| Caches (per core) | 2 kB 4-way L1 inst. cache, 16 kB 4-way L1data cache, 8 kB 2-way constant cache, 12 kB, 128 B cache line size except constant cache of 64 B line size. |
| Scheduling policies | |
| Warp | Greedy then oldest [31] |
| CTA | Round-robin and Load-balanced scheme |
| Interconnect details | |
| Interconnect | Crossbar, 32 B channel, destination tag routing mechanism, 625 MHz |
| Memory partition specifications | |
| L2 cache | 128 kB/memory sub-partition, 8-way, 128 B cache line size |
| DRAM model | Out-of-order FR-FCFS,8 Memory partitions,4 B Bus width,4 B Burst Length |
| GDDR3 timing | 800 MHz, tCL = 10, tRP = 10, tRC = 40, tRAS = 25, tRCD = 12, tRRD = 8, tCDLR = 6, tW R = 11 |

having replicas of such cores, which in turn increases number of concurrently executing warps or wavefronts. The existing CTA scheduler [26] utilizes all the cores available in the system by dispatching CTAs one by one in a round-robin order to all the cores until each core has the maximum number of CTAs it can run. The amount of on-chip resources like register file size, shared memory size, SIMT units required by a CTA, limits the total number of CTAs that can be scheduled simultaneously in each core. Fig. 1 shows the performance of applications (Table 1) from CUDA SDK 4.1 [28], Rodinia [5] and from Bakhoda et al. [1], on the GPGPU-Sim 3.2.2 which simulates the architectural configuration described in Table 2 with 30, 60 and 90 cores.

With increasing core counts, some applications exhibit linear improvement in the performance. Such applications require resources within the core (or the core itself) more frequent than the other resources in GPGPU. However, some applications show the opposite trend in the performance with the increasing number of cores as shown in Fig. 1. A GPGPU is known for hiding stalls in the execution due to long latency memory operations by hiding the latency of a thread while executing another one. But some applications may not have sufficient ready warps or threads to hide latency as all warps are waiting for their data from the off-chip memory. Increasing number of cores in this case may increase number of memory requests sent by multiple cores causing an avalanche of stalls in the application. Therefore, such massive-multithreading may cause contention in the on-chip network and in the off-chip memory system. Memory requests from threads can cause DRAM bank conflicts, row buffer conflicts, data and address bus conflicts with other threads' memory requests if the memory system is shared among all the threads [4,22]. Such applications exert enormous pressure on DRAM system, which is evident from increased DRAM bandwidth utilization. However, it is observed that there are parallel applications whose off-chip bandwidth requirement is low. They are dependent on very high speed memories namely L1 cache, shared memories, registers within the core, rather than off-chip DRAMs. It is observed that these applications prefer to have additional number of cores.

We anticipate that with the advancement in silicon technology, the future GPGPU architecture will accommodate large number of cores. However, considering large core counts with limited memory bandwidth, we demonstrate that using all the available cores in the system may not necessarily be a good choice to ameliorate the performance of some applications. The applications of this type show performance improvement when the capacity of the memory system in GPGPU is enhanced. The memory intensive applications experience performance improvement when the number of memory partitions is increased or the frequency of memory subsystem is scaled up. Whereas, the same memory intensive applications do not show performance improvement when frequency of core is increased or when the number of cores is scaled up [33].

In this article, we propose an (***Ap***)plication aware (***S***)calable (***A***)rchitecture called ApSA for GPGPU. This architecture includes Acquire and Regulate CTA (AR-CTA) scheduler that acquires the metrics to be used to identify the resource requirement of an application (i.e. type) and decides the number of cores which will be operational throughout the execution of the application with the non-operational cores clock-gated. ApSA further includes a Frequency Calibrator (FC) module that is responsible to scale the frequency of memory sub-system and cores using the statistics procured from AR-CTA scheduler.

In summary this article makes the following major contributions:

— We categorize a suite of applications in two types as type-I and type-II leveraging the metrics like the average bandwidth utilization across memory partitions, the instructions per cycle (IPC) with respect to maximum IPC of GPGPU and amount of branch divergence. The application type indicates the set of resources required by an application that favours its performance.
— We propose an Application aware Scalable Architecture (ApSA) which incorporates a) AR-CTA scheduler that identifies the application type and carries out the optimal resource allocation. b) Frequency calibrator which is controlled by AR-CTA scheduler, that scales core and memory frequency if required by an application.
— We present the performance of the proposed architecture with the profiling overhead experienced by various applications. The resulting hardware overhead and power consumption are evaluated.

The article is organized as follows, after a brief introduction in Section 1; Section 2 describes an overview of GPGPU architecture and causes of memory bandwidth saturation. Section 3 presents a classification scheme. Section 3 also describes the implementation of our proposed Application aware Scalable Architecture (ApSA) for GPGPU in detail. Experimental methodology is described in Section 4 followed by results that are discussed in Section 5. In Section 6 we survey prior work in this area, followed by conclusions in Section 7.

## 2. GPGPU architecture overview

A GPGPU consists of many simple in-order cores. Each core has 8 to 32 SIMT lanes. Our target system shown in Fig. 2 consists of computing cores with 32 SIMT lanes each. Each core has a private L1 data cache and a high bandwidth shared memory that can be shared by threads from the same CTA. If the requested data is not present in shared memory or in L1 data cache, the memory request is forwarded to one of the memory partitions through the scalable crossbar interconnection network. Each memory-partition consists of an L2 shared cache bank, memory controller and a DRAM. There are 8 such memory partitions in our target system. The architectural configuration used in this work is described briefly in Table 2 which is simulated using GPGPU-Sim [1].

### 2.1. Memory system in a GPGPU

Memory system in a GPGPU comprises of the scalable interconnection network that connects cores to the memory partitions that include the L2 cache banks, memory controllers and the DRAMs. As mentioned before, each core has a private L1 cache. A fixed number of outstanding requests for L1 cache can reside in Miss Status Handling Register (MSHR). The requested address on miss from L1 cache is forwarded to an L2 cache bank which is connected to a DRAM channel. L2 is also supported by MSHR. As in the case of memory intensive applications the number of memory requests is large the MSHR of L2 is now filled completely. This prevents L2 from accepting new requests and the interconnection between L1/core and L2 is now saturated. With this condition no more requests are accepted from L1 to the interconnect. The MSHR of L1 may get filled up if a core sends more requests. Thus continued rejection of memory requests reaches the core when all MSHRs of L1 are filled up. No new memory requests can be issued from any new warp now. This causes frequent stalls due to serialization of new memory requests. The overall effect of this is the degradation in the performance in the form of low IPC of the applications.

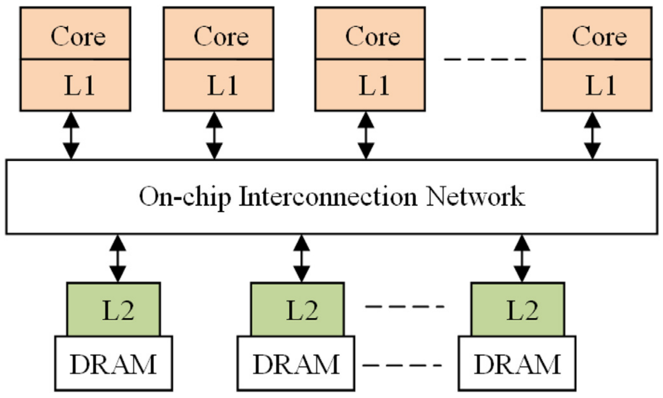


**Fig. 2.** GPGPU Architecture.

While compute intensive benchmarks have shown better performance with GPGPUs than conventional multicores [13,27], accelerating memory intensive workloads on GPGPUs presents a challenge [21]. Scaling up number of compute cores that consists of ALU, floating point units (FPUs) often benefits type-I applications. Type-I applications are those designed by the programmer in such a way that most of their requirements are fulfilled from on-chip resources such as high speed memories like shared memory,L1 cache, register banks or ALUs, FPUs, etc. or both. Whereas in the case of type-II applications, too many off-chip requirements (i.e. memory requests) may lead to congestion in the network, which causes cores to stall their operation. To relieve the load on the network and the memory subsystem, increasing number of memory partitions or by incorporating sub-channels within a partition [40] benefits their performance. Previous works [10–12,16,36] have proposed thread scheduling algorithms to improve the performance of memory intensive applications. These works present a dynamic or runtime solutions by periodically decreasing and increasing number of threads within each core for type-I and type-II applications respectively. Impact of static scaling of memory partitions in GPU is presented by Marino and Li [20].The other technique to reduce stalls in a memory dependent application is to slow down the operating speed of cores or scale up the frequency of the memory system or both [15,33].

### 2.2. GPGPU applications and its classification

This section presents a preliminary technique we used for classification of the representative applications, listed in Table 1, as type-I and type-II. The threshold values of the metrics to classify the applications are defined by statically running the application on a 30-core GPGPU with architectural configurations presented in Table 2. The metrics adopted for classification presents the important information of IPC, traffic in the interconnection network and the amount of branch divergence in the application.

#### 2.2.1. Instructions per cycle (IPC)

It is defined as the average number of instructions executed for each clock cycle. It is the multiplicative inverse of cycles per instruction [9]. IPC of a GPGPU is dependent on many factors. The clock speed of a GPGPU being lower than that of today's CPUs, the IPC of a GPU is highly dependent on number of operating cores. The availability of the input operands is another factor. Farther the operands are from the cores architecturally, more cycles would be consumed for the execution. A memory intensive application frequently accesses L2 caches and off-chip memories due to which number of clock cycles to execute an instruction also grows. Hence we consider IPC for the GPGPU as the principal metric to determine the compute intensiveness or memory intensiveness of an application. For an NVIDIA core also known as a Streaming Multiprocessor (SM) if Single Instruction Multiple Data (SIMD) width is 32 (considered in this work), maximum theoretical IPC is 32, hence for a 30-core configuration maximum IPC is 32 (warp size)

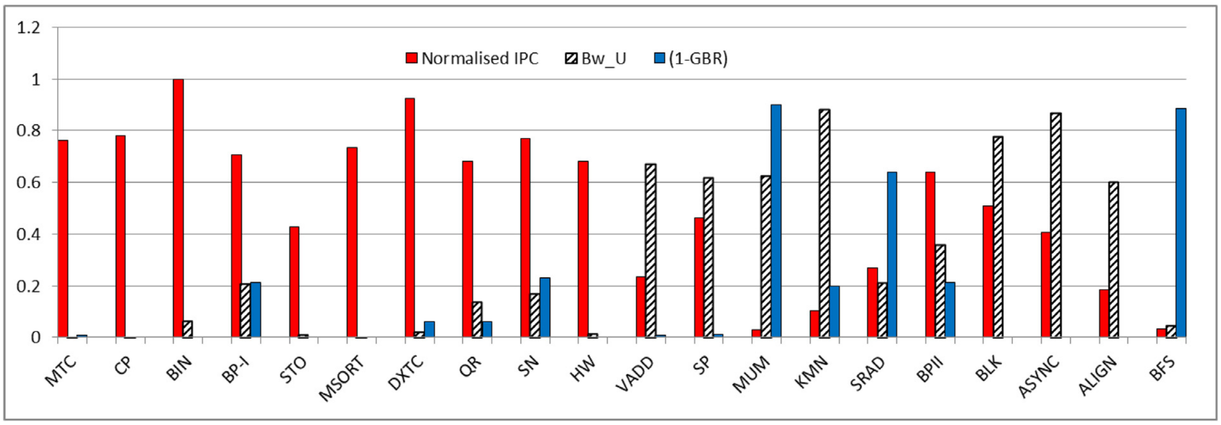


**Fig. 3.** IPC and bandwidth utilization of application on a 30-core GPU.

times 30 (the number of cores). Ideally a single precision instruction in a GPGPU can be executed in one cycle. If the IPC of an application is close to the maximum IPC of a GPGPU system then the application often exhibits high compute intensiveness. In Fig. 3,the plot of IPC of Binomial options pricing (BIN) is close to the 100% performance of the system putting it in type-I category without further inspection in terms of other metrics.

### *2.2.2. Bandwidth utilization (Bw_U)*

Bandwidth utilization (Bw_U) is defined by Yuan and Aamodt [37], as the percentage or fraction of time that a DRAM chip is transferring data across its pins over a given period of time. In our work, bandwidth utilization is averaged across all 8 memory partitions to classify the application. For highly compute intensive applications the bandwidth utilization is extremely low which is evident from BIN kernel application whereas for type-II applications bandwidth utilization is greater than atleast 25% and performance in terms of IPC is usually lower than that of type –I applications.

### *2.2.3. Good to bad cycles ratio (GBR)*

Good to bad cycles ratio (GBR) represents amount of branch divergence in an application. It is the ratio of number of cycles in which atleast 78% of threads in a warp (i.e. 25 and above threads out of 32) are launched to the execution pipelines to the total number of stalls in the execution. The stalls can add significant delay in the execution. The delay arises when

a) There are serialized executions of threads due to branching
b) No warps are available to the cores
c) When threads are waiting for data from memory. A GBR of 100% implies absence of branching as in the case of MTC, CP, HW. A very low GBR indicates the presence of significant amount of branching as in the case of BFS. In other words amount of parallelism in application reduces with increased branch divergence.

From the analysis depicted in Fig. 3 the performance of BIN and DXTC is close to the maximum attainable performance, making them a type-I application. Remaining applications like MTC, CP, STO, MSORT, DXTC, QR, SN and HW exhibit average bandwidth consumption not higher than 24%. With such a low pressure on the memory system these applications can easily afford to have additional number of cores as substantiated in Fig. 1 where IPC improvement of these applications can be seen with increasing core counts with a fixed memory capacity. The estimate of all the threshold values of all the metrics are obtained by observing the behaviour of applications when run on a standard 30-core GPU similar to QuadroFX5800.

Applications like VADD, MUM, KMN, SRAD, ALIGN and BFS are not able to achieve the performance higher than 40% with a 30 core GPU. Even though ALIGN shows slight performance improvement when the core counts are increased (Fig. 1.), it is still categorized under type-II as the core utilization efficiency is merely 21%, 13% and 9.4% on 30-core, 60-core and 90-core GPGPU respectively. This efficiency is the ratio of actual IPC to the maximum IPC, GPU system is capable to offer. A high amount of branch divergence of 90% in MUM and 88% in BFS is observed. Therefore, adding cores will make the overall performance of such bandwidth-limited and parallelism-limited applications ineffective. The bandwidth consumption of SRAD is comparable to that of type-I applications but due the low IPC (26%) and branch divergence of 64%, makes it a type-II.

Some applications as SP, BLK and ASYNC exhibit IPC of 46%, 51% and 41% respectively. Their high bandwidth consumption does not allow these applications to have additional number of cores.

BP-II is the only application here, that has the performance of 63.8% and it shows slight improvement when number of cores is doubled as depicted in Fig. 1. The application also presents the bandwidth consumption of 37% which is higher than those of type-I applications. Hence BP-II is categorized as type-II application.

## 3. ApSA: Application aware Scalable Architecture

The present-day Ge-Force 10 series GPUs such as GeForce GTX 1080 features 28 cores and Volta Series' NVIDIA Titan V features 80 cores. The proposed target architecture ApSA consists of 60 cores. The same architecture reorients to 30-core architecture for some applications (type-II). The features in the target architecture are proposed assuming that more number of cores will be incorporated than those in present architectures.

The non-uniformity across applications due to their specific requirements and the analysis of the classification scheme motivated us to propose and implement a Scalable Architecture that encompasses an AR-CTA scheduler which identifies the application type and the frequency calibrator which scales the frequency of the operational cores and the memory system.

Fig. 4 shows ApSA offering two different platforms for type-I and type-II applications. ApSA handles one application at a time and the timeline of the whole operation consists of three steps (i) profiling (ii) decision making and (iii) action. The first two steps are carried out by the proposed AR-CTA scheduler and the action step is jointly performed by AR-CTA and frequency calibrator module based on the decision taken by AR-CTA. AR-CTA identifies the type of the application and then decides if all or half of the cores are to be operational for the rest of the execution. As shown in Fig. 1 using large number of cores is not favourable in all the cases, the proposed AR-CTA scheduler regulates number of cores once the application type is identified.

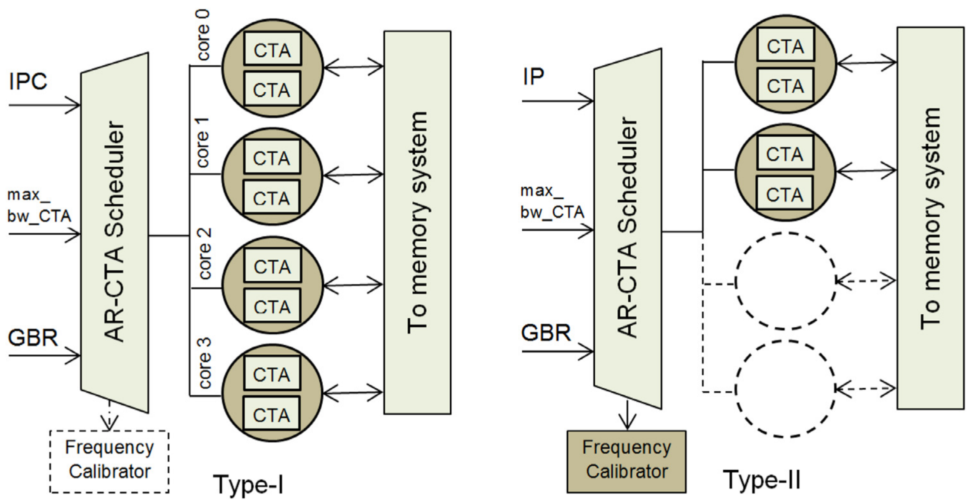


**Fig. 4.** ApSA overview.

### 3.1. Acquire and regulate CTA (AR-CTA) scheduler

The existing CTA scheduler in GPGPU determines maximum number of CTAs that can be run in a core depending on the requirements of each thread in terms of register file size, shared memory and number of SIMT units [7,12]. All the cores are allotted CTAs one after the other in a round-robin manner by the CTA scheduler as shown in Fig. 5. On the other hand the operations performed by proposed AR-CTA scheduler consists of a PROFILE phase in which metrics are acquired and a DECISION point in which decision is made whether the application is of type-I or type-II.

#### 3.1.1. PROFILE phase

Profiling some part of the application has been a common way to determine the characteristics of the whole application in computing. In the proposed architecture ApSA, when the kernel is launched the AR-CTA scheduler after determining maximum number of CTAs per core, launches one CTA in one of the cores, unlike the existing CTA scheduler that launches all CTAs one by one to all the available cores. In the existing architecture, CTAs are assigned in round-robin manner to all the available cores. CTA-0 to core-0, CTA-1 to core-1, CTA-2 to core-2 and so on. When the last core is reached the next CTA is issued to again core-0 then core-1 and this cycle is continued.

The proposed architecture profiles one of the CTAs first, say CTA-0. Hence initially the CTA scheduler launches only one CTA i.e. CTA-0 to core-0 and waits till all the threads in the CTA finishes with the execution. Once CTA-0 completes with the execution, decision is made on the basis of metrics collected from the execution.

If application is identified to be of type -I then the scheduler launches CTA-1to core-1, CTA-2 to core-2 till CTA-60 to core-60; CTA-61 will be issued to core-0, CTA-62 to core-62 and so on till the maximum number of CTAs per core is accommodated.

Whereas if the application is of type-II, CTA-1 to CTA-30 will be issued to core-1 to core-30 respectively then CTA-31 will be issued to core-0, CTA-32 to core-1 and so on.

All the threads in a CTA execute all the instructions in the kernel. It was noted by Zhang et al. [39] that if CTAs come from same application across different cores, the program instruction stream and execution flows on different cores will be quite similar or sometimes identical. The AR-CTA scheduler waits till the launched CTA is finished with the execution. The metrics acquired after profiling includes Instructions per cycle (*ipc_CTA*), maximum bandwidth utilization across all memory partitions (*max_bw_CTA*) and good to bad cycles ratio (*GBR*) which represents the branch divergence. Single CTA may not access some memory banks during the execution hence maximum bandwidth utilization is chosen instead of average utilization across all memory partitions. Algorithm 1 shows the steps followed in the PROFILE phase of ApSA and subsequently the acquired metrics are used to take the decisions and actions by the AR-CTA scheduler and the frequency calibrator.

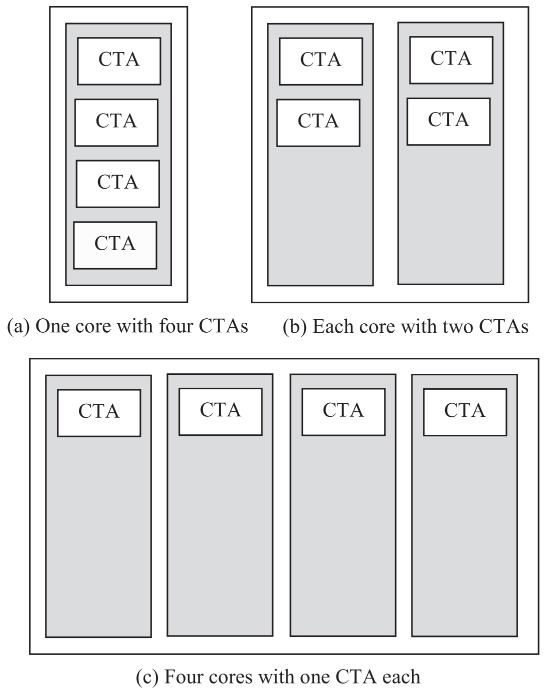

CTA
(a) One core with four CTAs
(b) Each core with two CTAs
(c) Four cores with one CTA each

**Fig. 5.** Three different GPGPUs with different number of cores with total 4 CTAs launched.

#### 3.1.2. DECISION point

The metrics collected through profiling the CTA are used by AR-CTA scheduler to determine if the CTA belongs to a type-I or type-II application.

The algorithm to determine the type of application is described in Algorithm 2.

At the end of the execution of first CTA, IPC (*ipc_CTA)* is checked. If it is greater than upper threshold *t_u_ipc* then application is certainly of type-I. The frequency calibrator is not enabled to calibrate the frequency so that rest of the execution could be carried with the default frequency with which profiling was carried out. The maximum

**Algorithm 1**
PROFILE phase.

```
N_cores: Total number of cores in the system
N: the total number of CTAs that have finished with the execution
ipc_CTA: instructions per cycle collected after profiling single CTA or 7000 cycles
max_bw_CTA: the maximum bandwidth consumption of all the memory partitions
    collected after profiling single CTA
IPC: current IPC,
n_cycles: number of cycles elapsed,
bwX: bandwidth utilization of Xth memory-partition,
GBR: Good to bad cycle ratio,
GBR_CTA: GBR after profiling.
procedure PROFILE
N_cores = 1;
IfN < 1 or n_cycles = = 7000 then // profiling window is 1 CTA or 7000 cycles
    whichever comes 1st
ipc_CTA = IPC,
max_bw_CTA = max (bw1, bw2,…, bw8)
GBR_CTA = GBR
end if
end procedure
```

**Algorithm 2**
Algorithm for DECISION point in ApSA.

```
type_I: application is of type-I
type_II: application is of type-II
procedure DECISION
Ifipc_CTA > t_u_ipcthen
type_I = true
elseifipc_CTA < t_l_ipcthen
type_II = true;
        else if max_bw_CTA > t_max_bwthen
        type_II = true
                else if GBR > t_GBR
type_I = true
elsetype_II = true
        end if
end if
end if
end if
end procedure
```

obtainable IPC from single core is 32 (100%), *t_u_ipc* is set to 85% which was found to be plausible choice by trial and error method. However, if *ipc_CTA* is less than the lower threshold *t_l_ipc* then the application is of type-II which is set to be 15%. For the values of thresholds, various experiments are conducted and a set of plausible choices is made. Execution of single CTA on single core and of entire application on 30 cores were analysed to arrive at the suitable threshold values. These values are in percentage, so that it is generic across different configurations. For example, these threshold values were used in a configuration similar to that of GTX480. On executing single CTA the algorithm was able to differentiate the applications. The categorization of application with CTAs that take few cycles to execute was similar to the configuration used in this work. Every architecture avails certain amount of compute and memory resource. Therefore as per the dynamic classification, a type-I application may be classified as type-II or vice-versa, so that an application gets optimal set of resources. For example BP-I is identified as type-II in architecture when memory specification was configured similar to that of GTX480. In this case after the identification of BP-I as type –II the frequency scaling operations were carried out.

When the *ipc_CTA* is between *t_l_ipc* and *t_u_ipc* then the maximum bandwidth utilization (*max_bw_CTA)* among all the memory partitions is checked. The threshold for *max_bw_CTA* is *t_max_bw,* which is 1.025% below which the application is considered to be of type-I. It is also observed that application with higher bandwidth utilization can be compute intensive. Therefore the next parameter the scheduler looks for is the amount of branch divergence. So if *max_bw_CTA* is greater than *t_max_bw* then value of *GBR* is checked. If *GBR* is lower than threshold *t_GBR* which is 65%, then the application is identified to be of type-II. Large number of parallel cores is not suitable for applications with limited amount of parallelism. In general, the applications that exhibit rich amount of branch divergence undergoes serialized execution, hence we keep them under type-II category. Once the decision is made the ACTION phase is carried jointly by AR-CTA scheduler and frequency calibrator. Application from the commencement of ACTION phase is executed in usual manner till the completion.

#### 3.1.3. ACTION phase

Action phase is carried out jointly by AR-CTA scheduler and frequency calibrator once the decision is made. At DECISION point the application is identified either to be of type-I or of type-II.

For type-I, the AR-CTA scheduler launches the rest of the awaiting CTAs on all the available cores in round-robin fashion. All the cores (60 in ApSA) would be operational carrying execution of rest of the CTAs. Frequency calibrator maintains the default frequency for the rest of the execution with which profiling was carried. Whereas if the application is identified to be of type-II after PROFILE and DECISION phase, AR-CTA scheduler launches CTAs to half of the total cores i.e. 30 in this study. AR-CTA scheduler at the same time flags the frequency calibrator to scale down the frequency of all the operating cores and scale up the frequency of memory system.

### 3.2. Frequency calibrator

Frequency calibrator is enabled when application is identified as type-II by the AR-CTA scheduler. Frequency calibrator is included especially for type-II applications to improve memory performance by scaling up the frequency of memory system.

Scaling frequency of memory system and of cores was proposed by Lee et al. [15] and Sethia and Mahlke [33]. Since the favourable resources for type-II applications are memory system, we attempt to reduce the strength of cores by a) keeping less number of cores operational and b) by scaling down the frequency of operational cores to keep up with the power constraints.

The steps that are carried out in ACTION phase are described in Algorithm 3.

## 4. Experimental methodology

We evaluate ApSA with applications summarized in Table 1. We use GPGPU-Sim 3.2.2 [1], a cycle-level GPU simulator for modelling the configuration in Table 2. ApSA employs total 60 cores with limited memory of 1GB of 8 partitions. Unlike the configuration used in Paul et al. [30], ApSA retains twice as many number of cores as in Paul et al. [30] making ApSA appear a skewed architecture with large core counts with limited memory. We demonstrate that this skewness due to several cores is beneficial to GPGPU applications that fall under type-I. ApSA reorients to a balanced architecture for type–II applications by keeping operational half of the total number of cores. Since the algorithm takes

**Algorithm 3**
Algorithm for ACTION phase in ApSA.

```
N_cores: Total number of cores in the system
New_N_cores: Number of cores to be operational after the decision.
FC: Variable that flags (enable or disable) frequency calibrator
procedure ACTION
Iftype_Ithen
New_N_cores = N_cores
  FC = 0;
elseiftype_IIthen
New_N_cores = N_cores/2;
        FC = 1;
    end if
end procedure
```

decision for every kernel we refer kernel as an application. These GPGPU applications are chosen from CUDA SDK [25], Rodinia [5] and from Bakhoda et al. [1] for comparative analysis. The applications are run till completion or one billion instructions, whichever comes first. Power consumption is estimated using GPUWatch [17] with 65 nm technology.

ApSA runs one kernel application at a time. During the profile phase, single CTA can take longer time to finish the execution. This can cause a huge performance penalty. Hence the profiling is carried till 7000 cycles or till complete execution of the CTA (whichever comes first). From the experience gained by running all the applications, we found that execution of CTA till 7000 cycles presents sufficient and necessary information about the characteristic of the application. Unlike threshold values, the number of cycles for which CTA is executed is micro-architecture dependent. For different configurations the optimal number of cycles for CTA execution needs to be recalculated.

Memory system comprises of interconnection network, L2 caches, memory controllers and DRAMs. Frequency calibrator module changes the frequency of entire memory system and all the operational cores. In [33], the frequency of cores were increased to 15% and that of memory system decreased to 15% if the application is found to be compute-intensive and vice-versa if application is memory-intensive. Therefore, we restrict the scaling amount to 15% employed in [33].The frequency calibrator in ApSA is enabled for type-II applications which have usually limited parallelism or heavy memory system requirements or both. It decreases the frequency of cores by 15% and increases the frequency of the memory system by 15% when enabled.

## 5. Results and discussion

### 5.1. Performance of ApSA

Fig. 6 shows the performance of type-I and type-II applications on 30-core, 60-core GPGPU and on our proposed architecture ApSA of 60 cores. The performance in terms of IPC is normalized with respect to that of MTC on a 30-core GPGPU in type-I, and VADD on a 30-core GPGPU in type-II applications. ApSA incorporates the ability of AR-CTA scheduler which scales the number of cores after identifying the resource requirement of the launched application and the ability of frequency calibrator which scales the frequency of resources namely cores and the memory system. Binomial option pricing (BIN) being very compute intensive exhibits a linear speed up with increasing number of cores and it presents the performance almost equal to that of a 60 core GPGPU with no profiling. Therefore, most of the type-I applications would like to have computing resources rather than off-chip memory resources. Accordingly ApSA is able to choose all cores available in the system without giving any boost to memory system for type-I applications. This results in an average improvement of 76% with respect to a 30-core GPGPU. However, when ApSA is compared with a 60 core GPGPU (only in the case of type-I) there is a slight performance degradation due to overhead caused by run-time profiling. The profiling overhead is just 1.61% averaged across all the type-I applications whereas it is 1.15% across all type-II applications.

Type-II applications exhibit the performance improvement in ApSA due to combined effect of AR-CTA and frequency calibrator. Relative to a 30 core GPGPU, ApSA with type-II applications presents average performance improvement of 11.04% with MUM and BLK experiencing a highest improvement of 31% and 33% respectively.

Since ApSA employs 60 cores for type-I applications, the ideal GPGPU architecture to compare the performance would be a 60-core GPGPU without profiling. For type-II applications as ApSA uses only half of the available cores i.e. 30 in this case, the ideal GPGPU architecture for comparing performance of ApSA would be a 30-core GPGPU without profiling.

### 5.2. Hardware overheads

GPUs are equipped with low-level hardware performance counters. Leng et al. [18] used 28 performance counters on a GTX 680 GPU that includes IPC and utilization level of different functional units of GPU. Due to the limited publicly available information we assume that counters for collecting IPC and bandwidth utilization level of each memory partition are available. The additional hardware external to the CTA scheduler are required to obtain GBR and maximum bandwidth utilization among all memory partitions. The hardware that identifies the application type is implemented within the CTA scheduler. The metrics IPC, max_bw_CTA and GBR form a feedback loop to the AR-CTA scheduler and 32-bit registers are used to store the threshold values of variables presented in decision point algorithm. Frequency calibrator selects the appropriate frequencies of operational cores and memory system upon receiving the signal AR-CTA scheduler. All the mentioned hardware circuitry is implemented in RTL using VHDL and synthesized with Synopsys Design Compiler using Faraday 65 nm standard cell library. Overall the design requires an additional area of 0.057 mm$^2$ and total leakage and dynamic power of 12 mW in the 60-core GPGPU. Considering 65 nm design, this additional hardware amounts to the overhead of 0.01% of the area of NVIDIA GP100, a 60 core system, implemented in a much smaller 16 nm FinFET Plus technology [29].

### 5.3. Average power consumption

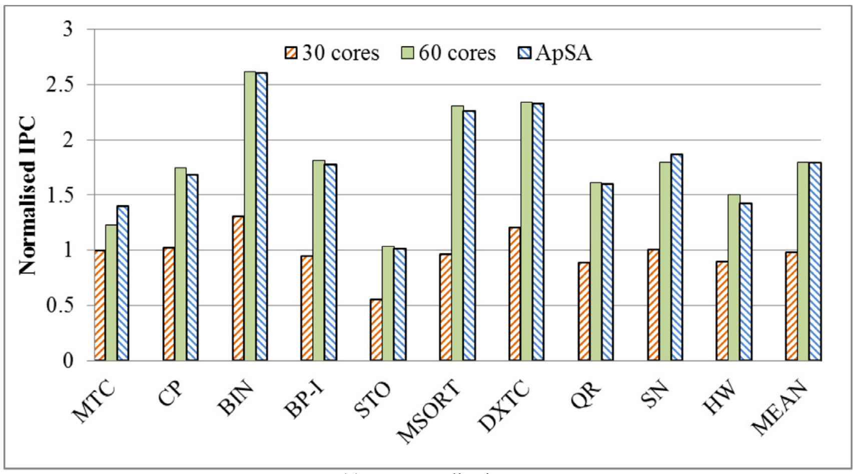
The objectives of prior work by Hong and Kim [8] and recent work by Paul et al. [30] were to improve the power efficiency of 30-core and 32-core GPGPU respectively without compromising the performance. The primary aim of ApSA is to provide a common platform that re-orients itself to a commonly used 30-core but frequency-scaled GPGPU for type-II applications or to a relatively skewed 60-core GPGPU for type-I applications to enhance the performance. Fig. 7(a) presents average power consumption across type-I applications normalized to that of MTC application with30-core GPGPU. Fig. 7(b) presents average power consumption across type-II applications normalized to that of VADD application with 30-core GPGPU. We found that major share of the total power is consumed by the register files, special function units and floating point units in type-I applications. As shown in Fig. 7, average power consumption for type-I application is 5.3% more with ApSA-II relative to default 60-core architecture. For memory–bandwidth limited applications the highest power is consumed by memory-controller and DRAM. Whereas execution pipelines, idle cores are the major consumer of power in the type-II applications such as BFS and SRAD, which exhibit significant branch divergence. As ApSA increases the frequency of memory system further, power consumption in many type-II applications are higher than the standard 30-core GPGPU. On average ApSA consumes 10.09% more power than a 30 core GPGPU as memory frequency is boosted for type-II applications.

Relative to a 60-core GPGPU, for type-II applications, an average of 20.08% of reduction in power consumption is observed for ApSA. We believe power gating non-operational cores will further improve the power efficiency of the GPGPU.

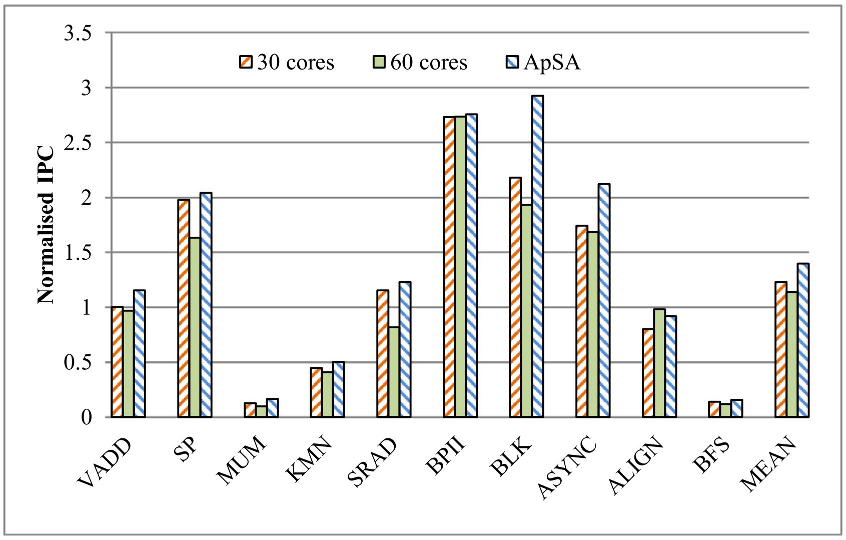
## 6. Related work and discussion

In prior work of Hong and Kim [8] proposes a few cores to be operational for memory-bandwidth limited applications in order to save power. Their analytical model analyses the source code of application statically and predicts the power, performance per watt and optimal number of cores. Unlike ApSA, the model limits number of cores by limiting total CTAs in the kernel application. In our proposal, number of cores is directly controlled by the CTA scheduler, eliminating the need of modifying the application by the programmers or compilers.

An in-depth study of the effects of scaling voltage/frequency of



(a) Type-I applications



(b) Type-II applications

**Fig. 6.** Performance of ApSA.

cores and number of cores on the throughput of a power-constrained GPU is presented by Lee et al. [15]. Throughput of application execution is analyzed in small intervals. Taking an oracle approach the applications are run with the best resource frequencies and optimal number of cores that maximizes the throughout for a single interval period and then combined. Whereas ApSA encompasses a complete runtime system which analyzes memory bandwidth utilization and amount of branch divergence along with IPC of a CTA to decide the appropriate resource for the application.

Sethia and Mahlke [33] proposed an Equalizer, a runtime system that identifies the requirements of the applications and tunes number of CTAs per core, core and memory frequency for efficient execution. Unlike ApSA number of cores is not tunable in Equalizer as CTAs are launched to all the available cores in their work.

Paul et al. [30] proposed Harmonia that adjusts the hardware tunables of GPGPU to balance the power of cores and memory system. They use a metric called as operations per byte (ops/byte) to find the characteristic of applications. Unlike ApSA, ops/byte is determined within phases of an application and each time the resources are configured for optimal performance and maximum power saving. A mismatch between ops/byte and the available hardware configuration may cause longer execution time which is a drawback. With the improved profiling technique the average performance loss of 1.3% is observed in ApSA which is less than that of Harmonia (3%) relative to the oracle scheme.

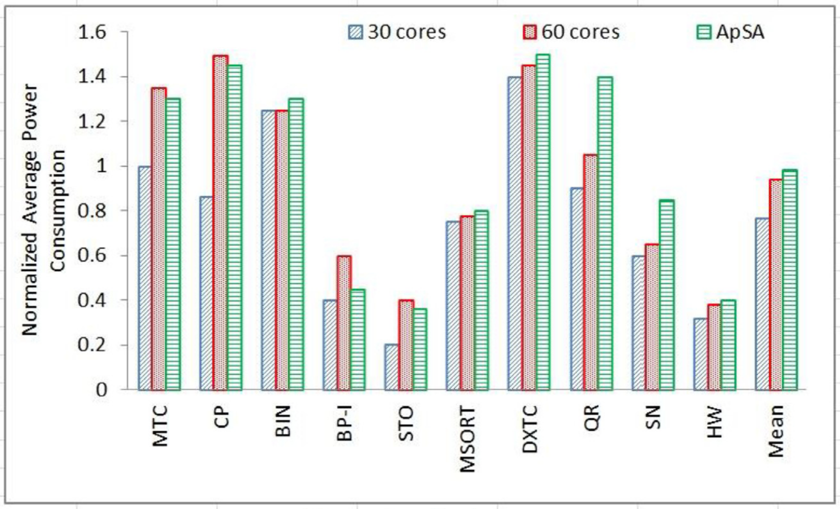
Kayiran et al. [12] proposed a dynamic CTA scheduling mechanism, DYNCTA, which allocates optimal number of CTAs per core determined by the requirements of an application. They demonstrate the benefits gained by memory intensive applications by decreasing the number of concurrently running CTAs. For ApSA the idea is to analyse the behaviour of single CTA first and then take the necessary steps. In previous work metrics are collected or execution is monitored periodically.

An alternative thread block scheduling technique proposed by Lee et al. [16], reduces number of CTAs per core to maximize the performance of applications that are bottlenecked by limited parallelism and available memory bandwidth.

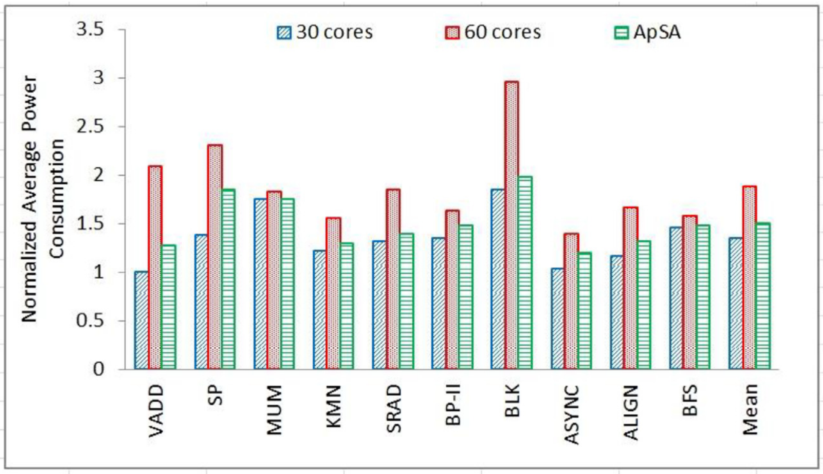
All the prior mechanisms involve continuous monitoring of the behavior of the application, in our work the decision is taken after the execution of one of the CTAs.

To improve the performance of GPGPU architecture, apart from CTA scheduler various warp scheduling techniques have been proposed. Cache-conscious wavefront scheduling proposed by Roger et al. [31], aims to reduce L1 miss within the core by throttling number of scheduled wavefronts. This technique is beneficial for highly cache sensitive workloads.

Narasiman et al. [23] proposed large warp micro-architecture that creates fewer but larger warps and divides them into SIMD width sized subwarps. These subwarps are made up of active threads chosen from large warp. This technique addresses resource underutilization due to branch divergence.



(a) Type-I applications



(b) Type-II applications

**Fig. 7.** Average power consumption.

Jog et al. [10], proposed another warp scheduler which enables efficient prefetching policies to improve memory latency tolerance in GPUs. OWL [11], a combination of CTA aware warp scheduling policy is proposed which aims to improve cache and DRAM utilization for a given amount of thread-level parallelism.

The proposed method ApSA deals with analysis of behavioural approach of CTA followed by necessary steps. This was not adopted in the previous work to best of our knowledge. In previous work metrics are collected in regular time intervals [10–12,33]. The idea is to analyse the behaviour of single CTA and then take the necessary steps. In previous periodic monitoring of execution and metrics also lead to switching the amount of resources accordingly.

Other warp scheduling techniques such as Dynamic warp formation by Fung et al. [7] addresses the underutilization of the execution pipelines within core due to branch divergence. All the diverged threads with same PC are regrouped so that all the pipelines are active. Since this technique does not focus on memory divergence among threads within a warp, Meng et al. [21] proposed dynamic warp subdivision for branch and memory divergence tolerance. This technique creates warp-splits when core does not have sufficient warps to hide latency and allows threads to interleave their execution in an asynchronous manner. Another warp scheduling technique PATS proposed by Xu and Annavaram [36] prioritizes warps with similar divergence pattern and power gates the unused execution lanes. Energy saving is the primary purpose of this technique, while it slightly reduces the performance due to overhead.

All the prior CTA and warp scheduling techniques do not scale number of cores. The prior scheduling techniques assume few (less than32) but all the cores available in the system. Whereas if the core count is high then Thread Level Parallelism (TLP) per core might be reduced, but number of CTAs running concurrently will still be high. This again leads to cache and DRAM contention in most of the proposed techniques which are targeted for memory- intensive applications. Compute intensive applications are bottlenecked if available cores are not sufficient. We believe that prior and proposed techniques would be more efficient if the architecture provides an optimal number of cores dynamically for the execution.

If we consider the latest 60-core GPU with Pascal architecture, it is similar to the 60-core that we have used in our work. The Ge-Force Series' NVIDIA Titan Xp GPU has 30-cores.The scaling is carried at the granularity of number of cores even though the number of SIMD pipelines may vary across the architectures. We believe that the proposed solution is scalable across architecture as the architecture is capable to take decision based on the amount of resources within the core. Depending on the resources in a GPU the nature of application identified might be different across GPUs. The final allocated resources would be the most suitable one for the application that favors its performance.

## 7. Conclusions

Massive multithreading offered by GPGPU leads to contention for resources like computational cores and available memory bandwidth. If increasing number of cores elevates the performance of some application, then available memory bandwidth or limited parallelism due to branching can become a bottleneck for some other applications.

In this article, we first classify the applications as type-I and type-II using a GPGPU with 30 cores and 8 memory partitions leveraging IPC, bandwidth consumption and amount of branch divergence as the metrics. Type-I applications have an average IPC of 85.31% and bandwidth utilization is less than 24% whereas for type-II applications average IPC is less than 40% and bandwidth consumption is higher than 24% and amount of branch divergence is upto 90%. When number of cores is increased, the performance of type-I applications improves without causing contention in the memory system as the memory requirement is not high. Type-II applications on the other hand experience performance saturation and even degradation when number of cores is high due to heavy contention for available memory capacity. Next, we proposed an Application aware Scalable Architecture (ApSA) for GPGPU which identifies the nature of the application and accordingly performs run-time tailoring. If application identified by ApSA is of type-I after profiling, then all the cores available in the system will be operational throughout the execution of application. If the application is identified as type-II, then half of the cores are activated and the frequency calibrator in ApSA scales up the frequency of memory system and scales down the frequency of all operational cores for rest of the execution. The average profiling overhead is 1.6% for type-I and to 1.15% for type-II applications on average. The average power saving by keeping half of the cores non-operational while running type-II applications is 20.08% with ApSA. This research work can be further extended with other mathematical algorithms which involve higher level of computational parallelism especially examples from artificial intelligence and data analytics domain.

## Supplementary materials

Supplementary material associated with this article can be found, in the online version, at doi:10.1016/j.sysarc.2018.07.003.

## References

[1] A. Bakhoda, G.L. Yuan, W.W.L. Fung, H. Wong, T.M. Aamodt, Analyzing CUDA workloads using a detailed GPU simulator, Performance Analysis of Systems and Software, 2009. ISPASS 2009. IEEE International Symposium on, IEEE, 2009, pp. 163–174.
[2] M. Baskaran, S. Ramanujam, P. Sadayappan, Automatic C-to-CUDA code generation for affine programs, Compiler Construction, Springer, Berlin Heidelberg, 2010.
[3] M. Bauer, H. Cook, B. Khailany, Cuda DMA: optimizing GPU memory bandwidth via warp specialization, Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis, ACM, 2011.
[4] G. Chadha, S. Mahlke, S. Narayanasamy, When less is more (LIMO): controlled parallelism for improved efficiency, Proceedings of the 2012 International Conference on Compilers, Architectures and Synthesis for Embedded Systems, ACM, 2012, pp. 141–150.
[5] S. Che, M. Boyer, J. Meng, D. Tarjan, J.W. Sheaffer, S.-H. Lee, K. Skadron, Rodinia: a benchmark suite for heterogeneous computing, Workload Characterization, 2009. IISWC 2009. IEEE International Symposium on, IEEE, 2009, pp. 44–54.
[6] Z. Fan, F. Qiu, A. Kaufman, S. Yoakum-Stover, GPU cluster for high performance computing, Proceedings of the 2004 ACM/IEEE Conference on Supercomputing, IEEE Computer Society, 2004, p. 47.
[7] W.W.L. Fung, I. Sham, G. Yuan, T.M. Aamodt, Dynamic warp formation and scheduling for efficient GPU control flow, Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture, IEEE Computer Society, 2007, pp. 407–420.
[8] S. Hong, H. Kim, An integrated GPU power and performance model, ACM SIGARCH Computer Architecture News, 38 ACM, 2010, pp. 280–289.
[9] J.L. Hennessy, D.A. Patterson, Computer Architecture: A Quantitative Approach, Elsevier, 2011.
[10] A. Jog, O. Kayiran, N.C. Nachiappan, A.K Mishra, M.T. Kandemir, O. Mutlu, R. Iyer, C.R. Das, OWL: cooperative thread array aware scheduling techniques for improving GPGPU performance, ACM SIGARCH Comput. Archit. News 41 (1) (2013) 395–406.
[11] A. Jog, O. Kayiran, A.K. Mishra, M.T. Kandemir, O. Mutlu, R. Iyer, C.R. Das, Orchestrated scheduling and prefetching for GPGPUs, ACM SIGARCH Comput. Archit. News 41 (3) (2013) 332–343.
[12] O. Kayıran, A. Jog, M.T. Kandemir, C.R. Das, Neither more nor less: optimizing thread-level parallelism for GPGPUs, Proceedings of the 22nd International Conference on Parallel Architectures and Compilation Techniques, IEEE Press, 2013, pp. 157–166.
[13] S.W. Keckler, W.J. Dally, B. Khailany, M. Garland, D. Glasco, GPUs and the future of parallel computing, IEEE Micro 5 (2011) 7–17.
[14] D.B. Kirk, W.H. Wen-mei, Programming massively parallel processors: a hands-on approach, Newnes (2012).
[15] J. Lee, V. Sathisha, M. Schulte, K. Compton, N.S. Kim, Improving throughput of power-constrained GPUs using dynamic voltage/frequency and core scaling, Parallel Architectures and Compilation Techniques (PACT), 2011 International Conference on, IEEE, 2011, pp. 111–120.
[16] M. Lee, S. Song, J. Moon, J.-H. Kim, W. Seo, Y. Cho, S. Ryu, Improving GPGPU resource utilization through alternative thread block scheduling, High Performance Computer Architecture (HPCA), 2014 IEEE 20th International Symposium on, IEEE, 2014, pp. 260–271.
[17] J. Leng, T. Hetherington, A. ElTantawy, S. Gilani, N.S. Kim, T.M. Aamodt, V. JanapaReddi, GPUWattch: enabling energy optimizations in GPGPUs, ACM SIGARCH Comput. Archit. News 41 (3) (2013) 487–498.
[18] J. Leng, A. Buyuktosunoglu, R. Bertran, P. Bose, V.J. Reddi, Safe limits on voltage reduction efficiency in gpus: a direct measurement approach, Proceedings of the 48th International Symposium on Microarchitecture, ACM, 2015, pp. 294–307.
[19] E. Lindholm, J. Nickolls, S. Oberman, J. Montrym, NVIDIA Tesla: a unified graphics and computing architecture, IEEE Micro (2) (2008) 39–55.
[20] M.D. Marino, K.-C. Li, Insights on memory controller scaling in multi-core embedded systems, Int. J. Embedded Syst. 6 (4) (2014) 351–361.
[21] J. Meng, D. Tarjan, K. Skadron, Dynamic warp subdivision for integrated branch and memory divergence tolerance, ACM SIGARCH Comput. Archit. News 38 (3) (2010) 235–246. ACM.
[22] O. Mutlu, T. Moscibroda, Parallelism-aware batch scheduling: enhancing both performance and fairness of shared DRAM systems, ACM SIGARCH Comput. Archit. News 36 (3) (2008) IEEE Computer Society.
[23] V. Narasiman, M. Shebanow, C.J. Lee, R. Miftakhutdinov, O. Mutlu, Y.N. Patt, Improving GPU performance via large warps and two-level warp scheduling, Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture, ACM, 2011, pp. 308–317.
[24] J. Nickolls, W.J. Dally, The GPU computing era, IEEE Micro (2) (2010) 56–69.
[25] NVIDIA Corporation 2011. CUDA C programming guide. Retrieved February 3, 2016 from docs.nvidia.com/cuda/cuda-c-programming-guide.
[26] NVIDIA Corporation, NVIDIA's Next generation CUDA compute architecture: FERMI, Retrieved February 3, 2016, 2009. http://www.nvidia.com/content/pdf/fermi_white_papers/nvidia_fermi_compute_architecture_whitepaper.
[27] NVIDIA Corporation, NVIDIAs next generation CUDA compute architecture: Kepler GK110, Retrieved February,3, 2016 from, 2012. https://www.nvidia.com/content/PDF/kepler/NVIDIA-Kepler-GK110-Architecture-Whitepaper.pdf.
[28] NVIDIA Corporation, NVIDIA CUDA toolkit 4.1.-archive, Retrieved February 3, 2016 from, 2012. https://developer.nvidia.com/cuda-toolkit-31-downloads.
[29] NVIDIA Corporation, NVIDIA Tesla P100, Retrieved September7, 2016, 2016. https://images.nvidia.com/content/pdf/tesla/whitepaper/pascal-architecture-whitepaper.pdf.
[30] I. Paul, W. Huang, M. Arora, S. Yalamanchili, Harmonia: balancing compute and memory power in high-performance GPUs, Computer Architecture (ISCA), 2015 ACM/IEEE 42nd Annual International Symposium on, IEEE, 2015, pp. 54–65.
[31] T.G. Rogers, M. O'Connor, T.M. Aamodt, Cache-conscious wavefront scheduling, Proceedings of the 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture, IEEE Computer Society, 2012, pp. 72–83.
[32] S. Rostrup, S. Srivastava, K. Singhal, Fast and memory-efficient minimum spanning tree on the GPU, Int. J. Comput. Sci. Eng. 8 (1) (2013) 21–33.
[33] A. Sethia, S. Mahlke, Equalizer: dynamic tuning of GPU resources for efficient execution, Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture, IEEE Computer Society, 2014, pp. 647–658.
[34] J.E. Stone, D. Gohara, G. Shi, OpenCL: A parallel programming standard for heterogeneous computing systems, Comput. Sci. Eng. 12 (1-3) (2010) 66–73.
[35] V. Volkov, J.W Demmel, Benchmarking GPUs to tune dense linear algebra, High Performance Computing, Networking, Storage and Analysis, 2008. SC 2008. International Conference for, IEEE, 2008.
[36] Q. Xu, M. Annavaram, PATS: pattern aware scheduling and power gating for GPGPUs, Proceedings of the 23rd International Conference on Parallel Architectures and Compilation, ACM, 2014, pp. 225–236.
[37] G.L. Yuan, T.M. Aamodt, A hybrid analytical DRAM performance model, Proceedings of the 5th Annual Workshop on Modeling, Benchmarking and Simulation, Austin, TX, 2009.
[38] L. Yu, H. Yao, X. Liao, A novel GPU resources management and scheduling system based on virtual machines, Int. J. High Perform. Comput. Netw. 9 (5–6) (2016) 423–430.
[39] T. Zhang, N. Jing, K. Jiang, W. Shu, M.-Y. Wu, X. Liang, Buddy SM: sharing pipeline front-end for improved energy efficiency in GPGPUs, ACM Trans. Archit. Code Optim. 12 (2) (2015) 16.
[40] N. Chatterjee, M. O'Connor, D. Lee, D.R. Johnson, S.W. Keckler, M. Rhu, W.J. Dally, Architecting an energy-efficient dram system for GPUs, High Performance Computer Architecture (HPCA), 2017 IEEE International Symposium on, February, IEEE, 2017, pp. 73–84.

**Winnie Thomas** was born in Mumbai. She has Bachelor of Engineering degree in Electronics and Telecommunication from University of Mumbai in 2007 and Master of Technology and PhD degree in Electronics from V. J. Technological Institute, Mumbai, in 2010 and 2017 respectively. She is presently associated with V. J Technological Institute as research fellow. Her research interest includes computer architecture, parallelism, GPU, CUDA programming and high performance computing. She is also student member of IEEE. Ms. Winnie Thomas is a recipient of the University Grant Commission Maulana Azad scholarship. She has received best paper award at the prestigious IEEE International Conference INDICON 2015 which is a flagship conference of IEEE India Council.

**Rohin D. Daruwala** is a Professor of Electronics Engineering at Veermata Jijabai Technological Institute, Mumbai, India. He received his Bachelor of Engineering (B.E.) from National Institute of Technology (NIT), Surat, India and Master of Engineering and PhD from Mumbai University, India. He has more than 25 years of experience in teaching undergraduate & post graduate students. His research interests include Digital Design, Computer architecture, Microprocessor Systems Design and Wireless Networks. He serves as a member of All India Board of Post-Graduate Education and Research in Engineering & Technology which is a constituent body of All India Council for Technical Education (AICTE). He, along with his students, is the recipient of prestigious IEEE Bendix award for two times. Dr. Daruwala is an active member of IEEE and IEEE Computer Society.