International Journal of Computer Applications (0975 – 8887)
Volume 147 – No.12, August 2016

Variability Analysis in the Power Appetite of GPGPU Applications

Winnie Thomas
V. J. Technological Institute
Department of Electrical Engineering
Mumbai, India.

Rohin Daruwala
V. J. Technological Institute
Department of Electrical Engineering
Mumbai, India.

ABSTRACT
Due to the high-performance demands, GPGPUs are designed to be optimized for higher performance, even at the cost of large power consumption. This article presents the variation in the power appetite of GPGPU applications. It proposes a method to predict the characteristics of an application through the way power is consumed by different components of the GPGPU. It is observed that certain components which are over-used by one application may be under-used by another application. This presents a challenge for the GPU architects to design a favorable and balanced system for different types of GPGPU applications. An architecture that improves power efficiency is currently required but for time-constraint real time system, performance cannot be compromised. This work is an attempt to provide precious insights on designing a reconfigurable system that fulfills the demand of end users.

General Terms
General purpose graphic processing units, power consumption, CUDA, execution unit

Keywords
Variability analysis, power breakdown, cycle-accurate, frequency scaling.

1. INTRODUCTION
The General purpose Graphics processing Unit (GPGPU) based on Single Instruction Multiple Data (SIMD) architecture has proved to be an efficient and cost-effective platform for various high performance computing (HPC) applications. Originally SIMD is popular among the applications such as digital signal processing and 3D rendering that deal with large data sets and do not demand complex control flow. With the advent of programming models like CUDA [1] and OpenCL [2], programming a GPGPU has become easier. The HPC applications are now written for such platforms to exploit parallelism and to improve throughput. These applications however show variation in terms of amount of parallelism, and affinity towards specific resources [3], [4], [5]. Hence the amount of power consumed by the applications also differs. A detailed analysis of the power appetite of some representative GPGPU applications (also shown in Fig 1) is presented. Not all the resources within the system require the same share from the total power. Analyzing individual application, it is found that one or more of the total resources within the GPGPU can consume a major lump of the total power relative to other resources. Hence it is concluded that the power consumption across the components in a GPGPU system is uneven. Combating this irregularity presents a challenge for future architects as power efficiency is to be enhanced without trading the performance of the applications.

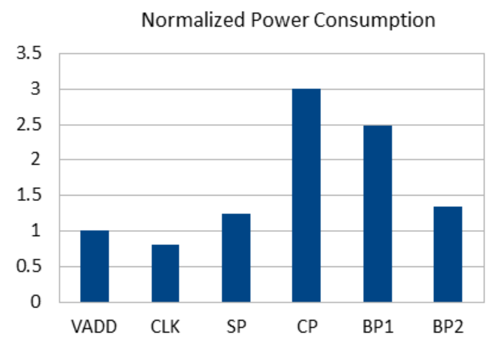

Fig 1: Power consumption of Applications normalized to VADD

2. PRIOR WORK
GPU power analysis can be found in several prior works. Huang et al. [6] evaluate the performance, energy consumption and energy efficiency of commercial GPUs. Hong and Kim [7] propose an integrated GPU power and performance analysis that predicts the power consumptions of GPU workloads based on the PTX instruction. Zhang et al. [8] analyze the performance and power consumption of a typical ATI GPU at the architectural level. They use micro-benchmarks to study power consumption which can be seen as a software management technique. Abe et al. in [9] show that energy can be saved by scaling cores and memory system of a GPU. Lee et al. [3] presented a method to apply DVFS algorithms to the GPU. They particularly aimed at maximizing performance under the given power constraint. Wang et al. [10] propose a technique for saving static energy in both L1 and L2 caches. They propose putting the L1 cache (which is private to each core) in state-preserving low-leakage mode when there are no threads that are ready to be scheduled. Further, the L2 cache is transitioned to low-leakage mode when there is no memory request. Lashgar et al. [11] propose the use of a filter cache to save energy in GPUs by reducing accesses to the instruction cache. A unified local memory design for GPUs is presented by Gebhart et al. [12]. Before the launch of each kernel, the system reconfigures the memory banks to change the partitioning of the memory. By effectively using the local storage, their design reduces the accesses to main memory.

We show that how the power appetite of different components of the GPGPU system varies across the applications. This variation in power appetite is the result of application characteristic and it becomes a challenge for the architects to build a system that runs the application with best performance with high power efficiency in compliance with the demands of users.

3. BACKGROUND

This section provides salient information of GPGPU architecture and GPGPU application.

3.1 GPGPU Application Structure

The applications for this work are written in CUDA. A CUDA application consists of many kernels. Kernels implement specific functions of an application [13]. The execution on GPUs starts with the launch of a kernel. Due to which, threads are generated, collectively called as a grid. When all the threads complete their execution, the grid formed by threads also ends for that kernel [13]. The remaining non-kernel part of the program is executed on host till the next kernel is called by host.

The threads on CUDA are organized into a hierarchy of threads, warps, CTAs and grid of CTAs. Once a kernel is launched, the grid corresponding to the threads is generated. To assign the threads to execution resources, they are grouped into CTAs. Execution of CTAs can be performed in any order. CTAs are scheduled in the pipeline of cores in the form of warps which are made up of 32 threads. For our target architecture the warps are scheduled with greedy-then-oldest technique into the pipeline by the warp scheduler in each core which prioritizes a warp until it stalls and then oldest warp is selected for execution.

For this work, one kernel is active at a time. Once the kernel is launched the CTA scheduler assigns CTAs to all the available cores [14]. The assignment of CTAs are carried in round-robin and load balanced fashion [15], [16].The first CTA is assigned to core 0, then next CTA to core 1 and so on. If a core is capable to run multiple CTAs then next round of assignment begins if there are enough CTAs. The maximum number of CTAs that a core can execute depends on the available on-chip resource requirement of each thread which includes register file size, number of ALU units, shared memory size. Hence this process of CTA allocation continues till all the cores get the maximum number of CTAs a core can run.

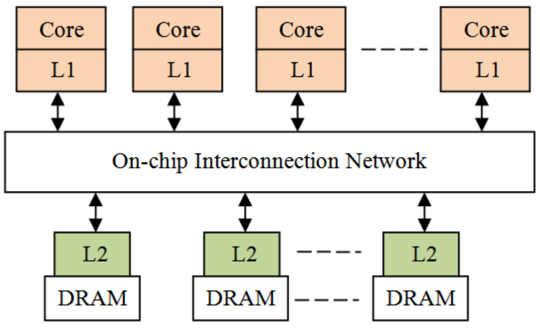

Fig 2: GPGPU Architecture

3.2 Baseline GPGPU Architecture

A GPGPU consists of many simple in-order cores. Each core has 8 to 32 Single Instruction Multiple Thread (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 on-chip 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 crossbar interconnection network. There are 8 memory partitions in our target system. The baseline configuration used in this work is described briefly in Table 1.

3.3 Application Suite

The applications for this work include Vector Addition (VADD), Clock (CLK), Scalar Product (SP) from CUDA NVIDIA SDK application suite [17]. Colombic Potential (CP) is from [14] and the two kernels of Back Propagation (BP1 and BP2) are from Rodinia [18] application suite. The evaluation of the techniques was carried on GPGPU-Sim v3.2.2 [14], a publicly-available cycle-accurate GPGPU simulator. The configuration described in Table 1 was modeled in GPGPU-Sim. Each application is run till completion or 1 billion instructions, whichever comes first. Power consumption is estimated using GPUWattch [19] assuming 65 nm technology.

Table 1: Baseline Architectural Configurations

Number of Shader cores	**30 , 650**MHz each
Warp size	32
Resource per core	Max.1024 threads,16kB shared memory, 16384 registers
Caches (per core)	2KB 4-way L1 inst. cache, 16KB 4-way L1data cache, 8KB 2-way constant cache, 12 KB, 128B cache line size except const cache of 64B line size.
L2 cache	128KB/memory sub-partition, 8-way, 128B cache line size
Scheduling policies	Greedy then oldest for warps, round robin and Load-balanced scheme for CTAs.
Interconnect	Crossbar, 32B channel byte, destination tag routing mechanism, 625MHz
DRAM model	Out-of-order FR-FCFS,8 Memory partitions,4B Bus width,4B Burst Length,
GDDR3 timing	800 MHz, t_{CL} = 10, t_{RP} = 10, t_{RC} = 40, t_{RAS} = 25, t_{RCD} = 12, t_{RRD} = 8, t_{CDLR} = 6, t_{WR} = 11

4. POWER APETITE ANALYSIS

Prior work [5], [15] and [16] have categorized the applications using various parameters as throughput, L1 cache miss rate, congestion in the network, amount of off- chip DRAM usage and even amount of branch divergence exhibited by the applications. The characteristic of an application is determined by leveraging the power consumption of different architectural components of the GPGPU. This work also investigates the potential of saving the power with different techniques without trading the performance of the application.

4.1 Vector Addition (VADD)

With the GPGPU configuration described in Table 1 the execution time of VADD is shortest among all the applications because of few instructions in its kernel. The throughput of the application is 26.6% of the peak throughput of the GPGPU [5]. High throughput represents the high computational power of an application. The low throughput of VADD signals underutilization of compute resources as floating point units (FPU), Special function units (SFUs) that collectively form execution unit. Fig 3 depicts the average power consumption of each micro architectural component of the application.

The major share of the total power is consumed by the memory with 54% of the total power. The memory system of GPGPU includes interconnection network between cores and L2 cache partitions also termed network-on-chip (NOC); the memory controller and the DRAMs. The percentage of power consumed by compute resource comprising execution units and register files are only 19%. Dynamic constant power is 14% which is represented by "Constant" in Fig 3.

Next, when the number of compute resources was doubled there is a slight increase in power consumption as shown in Fig 4. But there is degradation in the performance as shown in Fig 5. Clearly this application is memory intensive as adding more number of cores increases the number of memory requests saturating the memory system. As power dissipation of a component and frequency are directly proportional, third bar C-Freq-Down in Fig 4 corresponds to power consumption when the frequency of all cores in baseline is reduced by 50%. In this case the performance is enhanced by 67.8% and power saving relative to baseline is 15.3%. The last bars in Fig 4 and 5 show power and performance when the frequency of cores is reduced by 50% and that of DRAMs increased by 50% represented as C-Down-M-Up. In this case the performance improvement is 85.39% but power saving is 7.53% relative to baseline.

Hence the last two cases are favorable configurations for applications like VADD that requires the strength of the computational resources to be reduced. This can be accomplished by decreasing the population of processing cores or by reducing the operating frequency of the application. If speed is the priority as in real-time system then the last case should be preferred.

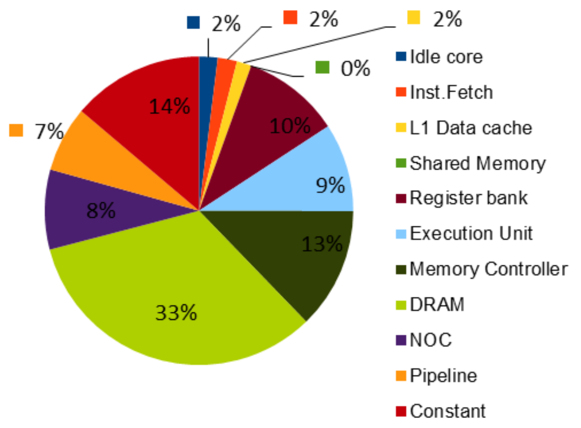

Fig 3: Average power breakdown of VADD

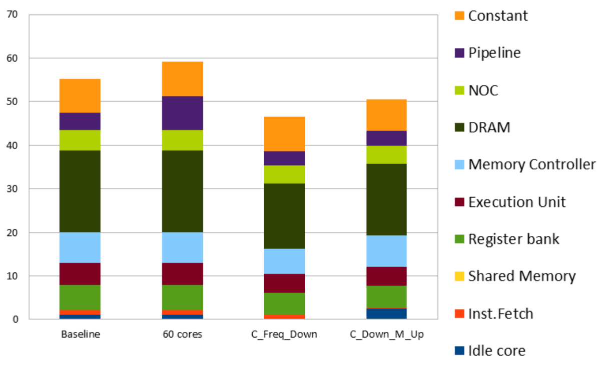

Fig 4: Average Power consumption of VADD with different configuration

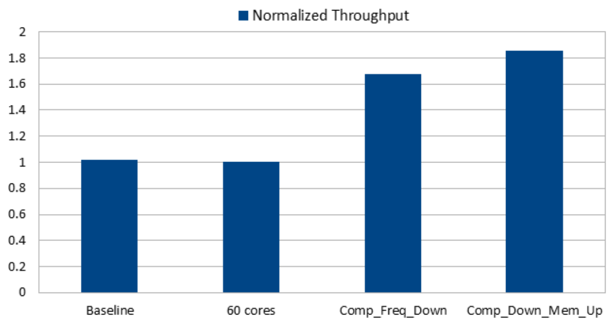

Fig 5: Performance of VADD normalized to baseline

4.2 Clock (CLK)

Clock (CLK) is an application with 64 CTAs and the performance of 52% relative to the peak performance of the baseline GPGPU. Fig 6 shows the power consumption when run on baseline configuration. The highest power consumer is the register bank which forms the part of the compute resources.

The performance of CLK in case of 60 cores is improved by 16.7% whereas power consumption also increases by 28.6% which is high. When the frequency of cores is reduced, 5.3% performance improvement is observed and power saving of 31.55% relative to baseline is seen. Hence for such applications the last two configurations are favorable as the performance improvement is limited due to few CTAs. Thus configuration that saves the power should be preferred for applications like CLK.

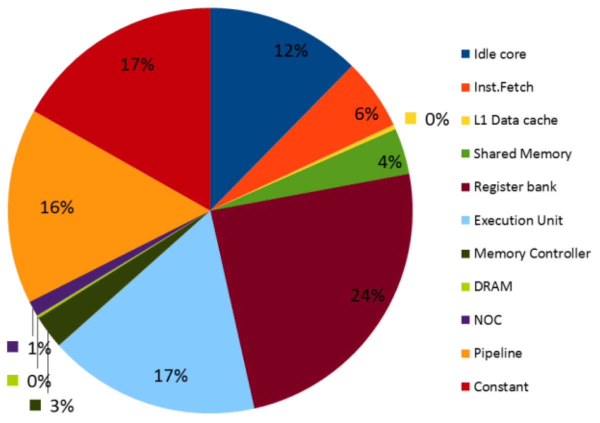

Fig 6: Average power breakdown of CLK

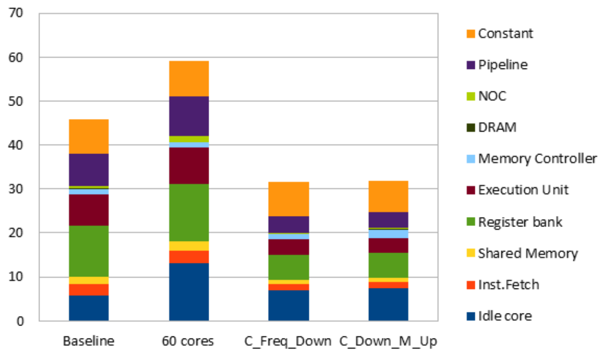

Fig 7: Average Power consumption of CLK

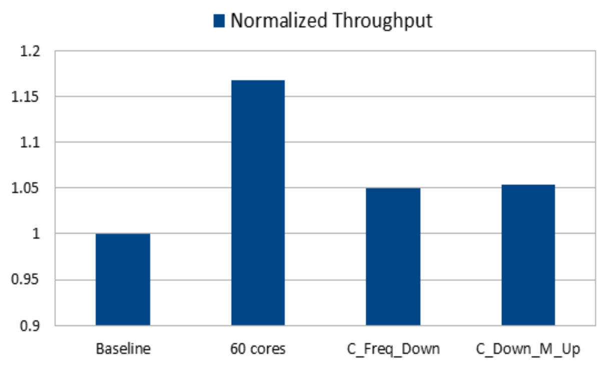

Fig 8: Performance of CLK normalized to baseline

4.3 Scalar Product (SP)

Number of CTAs that execute the kernel of SP is 196. Fig 9 shows that 39% of total power is consumed by memory system and execution unit consumes 10% and register file consumes 15% of the total power. Relative to VADD the application suffers higher idle periods. As each core at a time accommodates 3 CTAs in first round and remaining 38 CTAs wait for their turn. By the end of execution many cores would be idle as no more CTAs are to be scheduled. This creates significant amount of idle periods. Doubling the compute resources would not only degrade the performance by 26.11% as shown in Fig 10 but also increases the power consumption by 3.9% as shown in Fig 9.

As can be seen from the pie chart significant power is consumed by compute resources decreasing just the core frequency does not enhance the performance of the application. However certainly there is a significant power saving of 25% relative to baseline. The performance improvement of 23.498% is experienced as shown in Fig 11 depicted by C-DOWN-M-UP when memory frequency is increased. The power consumption is reduced by 20.3% relative to baseline. Clearly the GPGPU with the configuration corresponding to C-DOWN-M-UP is the suitable most for applications exhibiting the characteristics like that of SP.

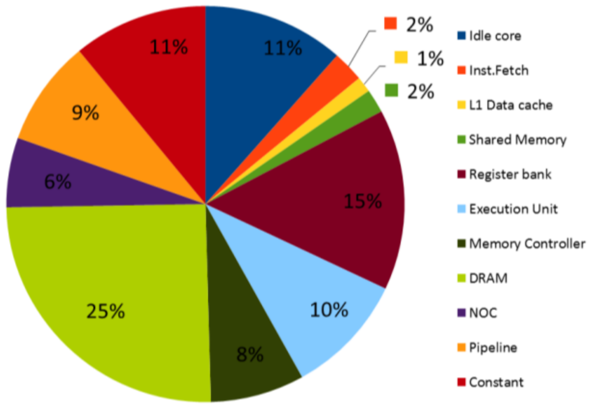

Fig 9: Average power breakdown of SP

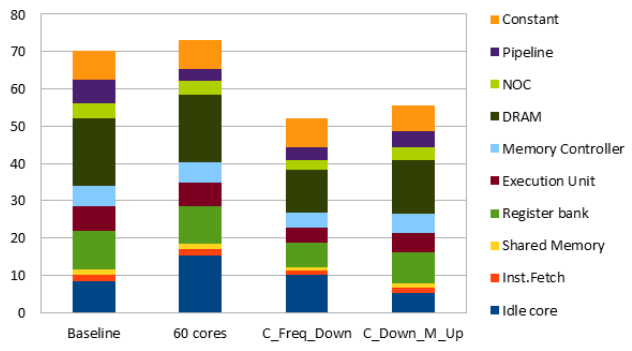

Fig 10: Average Power consumption of SP

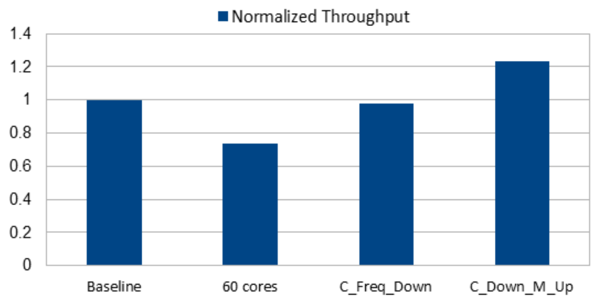

Fig 11: Performance of SP normalized to baseline

4.4 Colombic Potential (CP)

Previous work in [15] and [16] has established CP as a computationally intensive application and this is further substantiated here by the major power consuming components as shown in Fig 12, 71.92% of power is alone consumed by the execution unit. Doubling the cores definitely increases the power consumption the improvement in the performance is by 70.5%. In the next two cases where frequency of the cores are scaled down the performance is not affected but a power saving of 46% is observed. Increasing the frequency of memory depicted as C_Down_M_Up in Figures 13 and 14 saves no power and presents approximately similar performance relative to baseline configuration. Thus third case can be preferred due to the ability of the configuration to save maximum power without any loss of performance. The other way to get performance equivalent to second case would be to double the computation resource and reduce the frequency of each core to have a significant reduction in power consumption.

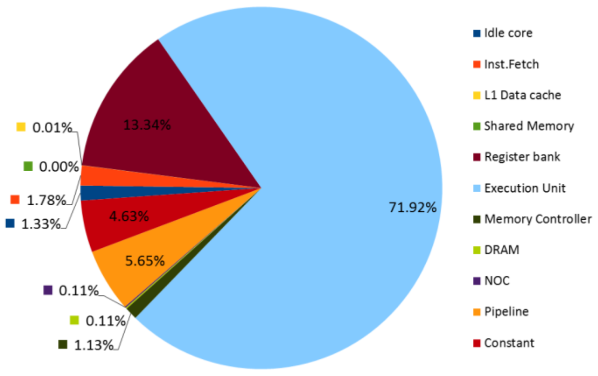

Fig 12: Average power breakdown of CP

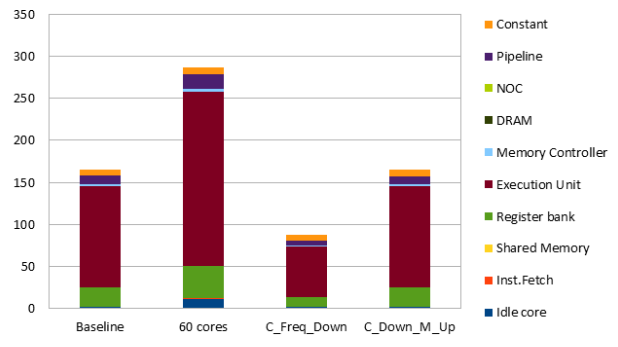

Fig 13: Average Power consumption of CP

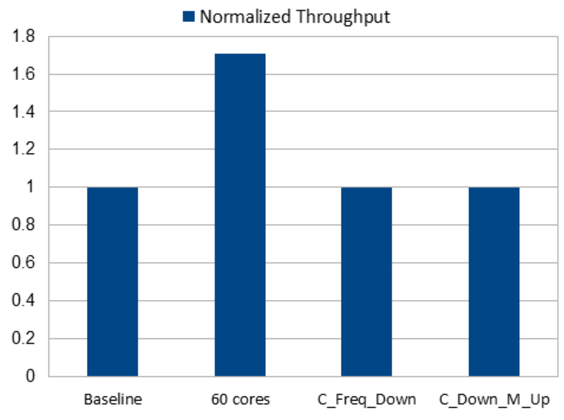

Fig 14: Performance of CP normalized to baseline

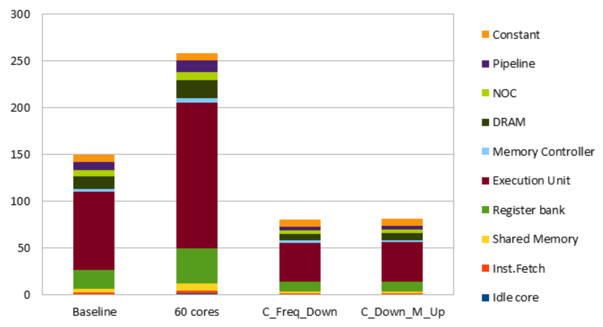

Fig 16: Average Power consumption of BP1

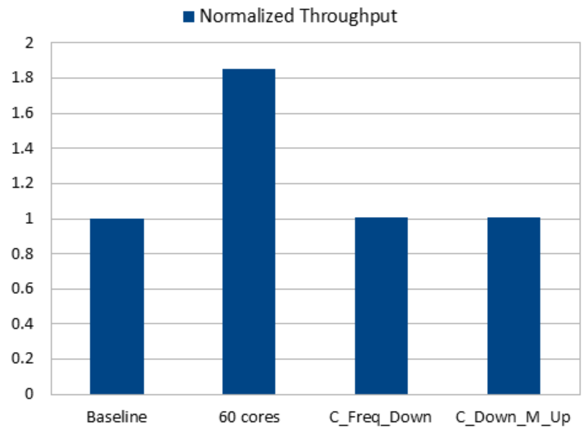

Fig 17: Performance of BP1 normalized to baseline

4.5 Back Propagation-1 (BP1)

The first kernel of Back Propagation (BP1) shows a very high demand of power from execution unit (60%) and 14% from register bank as shown in Fig 15. Clearly providing additional amount of compute resource will enhance its performance as shown in Fig 17 by second bar. An 85.3% of performance improvement is experience by application relative to baseline configuration. Accordingly the power consumption also increases by 81.9% as shown in Fig 16. In the next two cases an improvement of only 1.3% is observed but the amount of power saved is approximately 46% with frequency scaling. Hence second and third configurations (C_Freq_Down) can be preferred for the speed and power-efficiency respectively of applications like BP1.

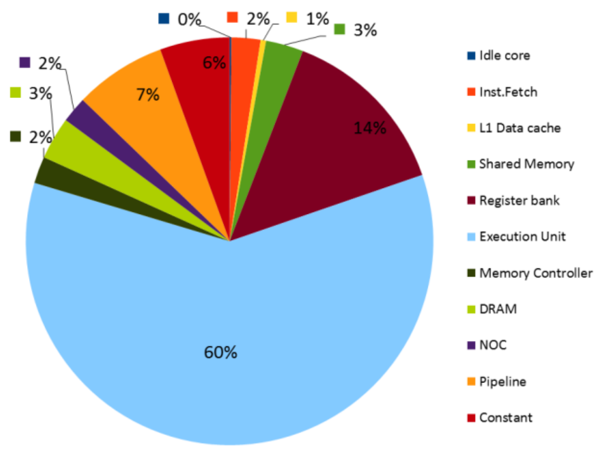

Fig 15: Average power breakdown of BP1

4.6 Back Propagation-2 (BP2)

From power breakdown pie chart of BP2 as shown in Fig. 18 both DRAM and execution unit consumes equal amount of power. The requirement of application is both compute resources and memory resource in equal proportion. On increasing population of compute resources the performance improves by 40.8% (Fig 20) and the power consumption increases by 36.68% as shown in Fig 19. When the frequency of baseline is reduced by 50% the 41.71% of power can be saved relative to baseline. If further the memory frequency is increased 41.1% power is saved and performance improves by 5.82% relative to baseline configuration. Thus the third case of reduced core frequency should be preferred due to improved performance and significant power saving for the applications like BP2 that have balanced power demand from execution and memory resources.

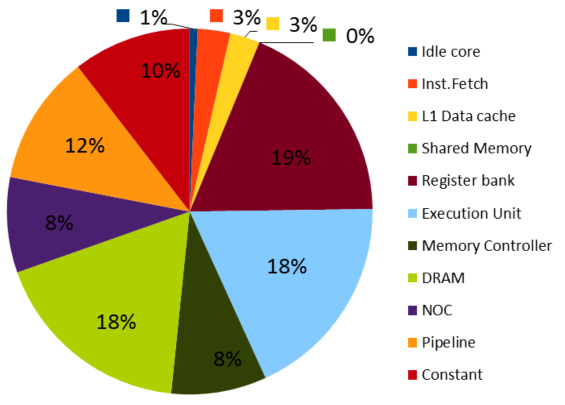

Fig 18: Average power breakdown of BP2

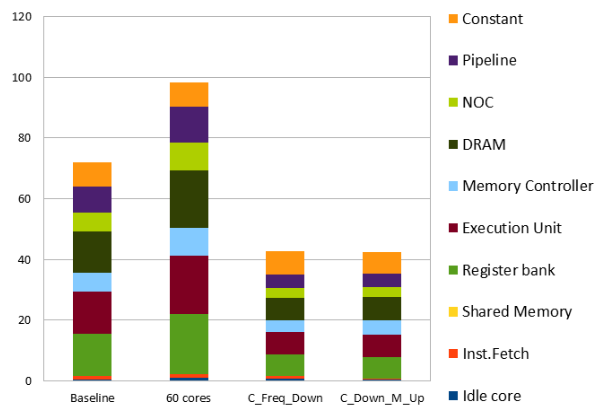

Fig 19: Average Power consumption of BP2

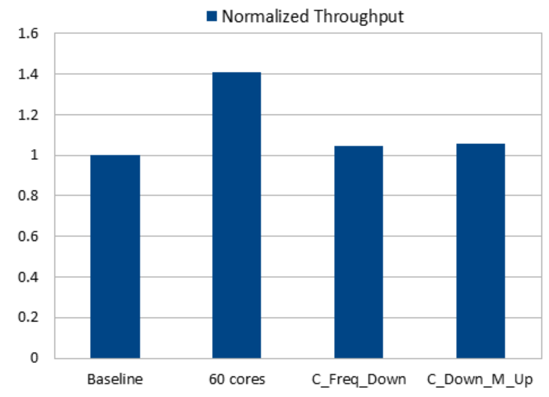

Fig 20: Performance of BP2 normalized to baseline

5. CONCLUSIONS

The article presents the variation in the power consumption of GPGPU applications. It makes an attempt to predict the characteristics of an application through the way power is consumed by different components of GPGPU. It is observed that how certain components which are over-used by one application may be under-used by another application. This presents a challenge for the GPU architects to design a favorable and a balanced system for all kinds of GPGPU application. An architecture that saves power is always good but for time-constraint real time system, performance cannot be compromised. Hence this work can provide some insights on designing a reconfigurable system that fulfills the demand of end users. In a multi-GPU environment with the different types of GPUs, predicting the behavior of an application through the power-appetite during run-time and porting the application to the appropriate GPU is the potential future scope of this work.

6. REFERENCES

[1] CUDA C Programming Guide. Retrieved February 3, 2016 from docs.nvidia.com/cuda/cuda-c-programming-guide

[2] Stone, J.E., Gohara, D. and Shi, G., 2010. OpenCL: A parallel programming standard for heterogeneous computing systems. *Computing in science & engineering*, *12*(1-3), pp.66-73.

[3] Lee, J., Sathisha, V., Schulte, M., Compton, K. and Kim, N.S., 2011, October. Improving throughput of power-constrained GPUs using dynamic voltage/frequency and core scaling. In *Parallel Architectures and Compilation Techniques (PACT), 2011 International Conference on* (pp. 111-120). IEEE.

[4] Sethia, Ankit, and Scott 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.

[5] Thomas, W. and Daruwala, R.D., 2015, December. Investigations into techniques to accelerate memory intensive GPGPU applications. In *2015 Annual IEEE India Conference (INDICON)* (pp. 1-6). IEEE.

[6] Huang, S., Xiao, S. and Feng, W.C., 2009, May. On the energy efficiency of graphics processing units for scientific computing. In *Parallel & Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on* (pp. 1-8). IEEE.

[7] Hong, S. and Kim, H., 2010, June. An integrated GPU power and performance model. In *ACM SIGARCH Computer Architecture News* (Vol. 38, No. 3, pp. 280-289). ACM.

[8] Zhang, Y., Hu, Y., Li, B. and Peng, L., 2011, July. Performance and power analysis of ati gpu: A statistical approach. In *Networking, Architecture and Storage (NAS), 2011 6th IEEE International Conference on* (pp. 149-158). IEEE.

[9] Abe, Y., Sasaki, H., Peres, M., Inoue, K., Murakami, K. and Kato, S., 2012. Power and performance analysis of GPU-accelerated systems. In *Presented as part of the 2012 Workshop on Power-Aware Computing and Systems*.

[10] Wang, W., Duan, B., Tang, W., Zhang, C., Tang, G., Zhang, P. and Sun, N., 2012, February. A coarse-grained stream architecture for cryo-electron microscopy images 3D reconstruction. In *Proceedings of the ACM/SIGDA international symposium on Field Programmable Gate Arrays* (pp. 143-152). ACM.

[11] Lashgar, A., Baniasadi, A. and Khonsari, A., 2013, February. Inter-warp instruction temporal locality in deep-multithreaded GPUs. In *International Conference on Architecture of Computing Systems* (pp. 134-146). Springer Berlin Heidelberg.

[12] Gebhart, M., Johnson, D.R., Tarjan, D., Keckler, S.W., Dally, W.J., Lindholm, E. and Skadron, K., 2011, June. Energy-efficient mechanisms for managing thread context in throughput processors. In *ACM SIGARCH Computer Architecture News* (Vol. 39, No. 3, pp. 235-246). ACM.

[13] Kirk, D.B. and Wen-mei, W.H., 2012. *Programming massively parallel processors: a hands-on approach*. Newnes.

[14] Bakhoda, A., Yuan, G.L., Fung, W.W., Wong, H. and Aamodt, T.M., 2009, April. Analyzing CUDA workloads using a detailed GPU simulator. In *Performance Analysis of Systems and Software, 2009. ISPASS 2009. IEEE International Symposium on* (pp. 163-174). IEEE.

[15] Kayıran, O., Jog, A., Kandemir, M.T. and Das, C.R., 2013, October. Neither more nor less: optimizing thread-level parallelism for GPGPUs. In *Proceedings of the 22nd international conference on Parallel architectures and compilation techniques* (pp. 157-166). IEEE.

[16] Jog, A., Kayiran, O., Chidambaram Nachiappan, N., Mishra, A.K., Kandemir, M.T., Mutlu, O., Iyer, R. and Das, C.R., 2013, March. OWL: cooperative thread array aware scheduling techniques for improving GPGPU performance. In *ACM SIGPLAN Notices* (Vol. 48, No. 4, pp. 395-406). ACM.

[17] NVIDIA CUDA Toolkit 4.1.-Archive. Retrieved July 3, 2016 from https://developer.nvidia.com/cuda-toolkit-31-downloads

[18] Che, S., Boyer, M., Meng, J., Tarjan, D., Sheaffer, J.W., Lee, S.H. and Skadron, K., 2009, October. Rodinia: A benchmark suite for heterogeneous computing. In *Workload Characterization, 2009. IISWC 2009. IEEE International Symposium on* (pp. 44-54). IEEE.

[19] Leng, J., Hetherington, T., ElTantawy, A., Gilani, S., Kim, N.S., Aamodt, T.M. and Reddi, V.J., 2013, June. GPU Wattch: enabling energy optimizations in GPGPUs. In *ACM SIGARCH Computer Architecture News* (Vol. 41, No. 3, pp. 487-498). ACM.