Energy Efficiency Analysis of GPUs

Juan M. Cebrián, Ginés D. Guerrero and José M. García
Computer Engineering Deptartment
University of Murcia
Murcia, Spain
Email: {jcebrian,gines.guerrero,jmgarcia}@ditec.um.es

***Abstract*—In the last few years, Graphics Processing Units (GPUs) have become a great tool for massively parallel computing. GPUs are specifically designed for throughput and face several design challenges, specially what is known as the Power and Memory Walls. In these devices, available resources should be used to enhance performance and throughput, as the performance per watt is really high. For massively parallel applications or kernels, using the available silicon resources for power management was unproductive, as the main objective of the unit was to execute the kernel as fast as possible.**

However, not all the applications that are being currently ported to GPUs can make use of all the available resources, either due to data dependencies, bandwidth requirements, legacy software on new hardware, etc, reducing the performance per watt. This new scenario requires new designs and optimizations to make these GPGPU's[1] more energy efficient. But first comes first, we should begin by analyzing the applications we are running on these processors looking for bottlenecks and opportunities to optimize for energy efficiency.

In this paper we analyze some kernels taken from the CUDA SDK[2] in order to discover resource underutilization. Results show that this underutilization is present, and resource optimization can increase the energy efficiency of GPU-based computation. We then discuss different strategies and proposals to increase energy efficiency in future GPU designs.

***Keywords*-Power Dissipation, Energy Efficiency, GPU, GPGPU.**

I. Introduction

GPGPU computing and GPUs in general have become one of the hottest trending topics in computer architecture research, but that's not the only thing that is getting hot. As with future exascale microprocessors, the increasing amount of transistors in the same area due to technology scaling is causing power dissipation problems. In the near future it is uncertain if all the transistors of the chip are going to be available in a given cycle due to temperature constraints [1].

In terms of energy efficiency, GPUS are considered to be one of the most power-efficient architectures up to date [2]. However, as the number of available resources increases their optimal assignment and utilization becomes harder, lowering the overall performance per watt. There are several key factors that reduce the performance per watt of massively parallel microprocessors:

- Memory bandwidth (on-board and off-board).
- Power dissipation and power supply.
- Scalability and data dependencies.
- Legacy software.
- Legacy graphic applications and engines.

NVIDIA Fermi GPU family has around 340-512 available computation units while ATI HD7xxx cards offer 2048+ computation units, depending on the analyzed model. Even though the number of available cores will increase in the next generations of GPUs (nVidia Kepler promises to double the cores) their total number will eventually saturate. This is mainly because of memory bandwidth problems, power dissipation problems and inter-core data dependencies.

If we consider the internal memory bandwidth (GPU to on-board memory) of current GPUs[3], like, for example:

- nVidia GeForce GTX 465: 102.592 GB/sec (256 bit bus). Per core bandwidth, 102.592/352 = 0.2914 GB/sec.
- nVidia GeForce GTX 480: 177.408 GB/sec (384 bit bus). Per core bandwidth, 177.408/480 = 0.3696 GB/sec.
- nVidia GeForce GTX 580: 192.384 GB/sec (384 bit bus). Per core bandwidth, 192.384/512 = 0,37575 GB/sec.

we can see that the per/core bandwidth is very limited, and increasing the amount of available cores without increasing the bandwidth at the same rate may cause starvation for certain applications. Moreover, technology scaling reduces die size, allowing manufactures to reduce production costs by obtaining more GPU dies per wafer. However, smaller dies mean less surface area for interconnection buses, thus further reducing the total memory bandwidth. This not only affects on-board memory bandwidth, if we take a look to the available bandwidth between the GPU and main memory, numbers are far worse. Nevertheless, these periods of low core usage may be slightly hidden by using background memory transfers or by running several kernels simultaneously, but still offer many chances to increase energy efficiency.

From the power point of view, core scaling is also limited. As stated in [1], it is uncertain that all the cores of massively parallel microprocessors can be used simultaneously due to temperature constraints and leakage power, even if we use low voltages and frequencies. Moreover, even nVidia is having problems with the additional power requirements

[1]General Purpose Graphics Processing Units
[2]Software Development Kit
[3]In our work we have only analyzed nVidia GPUs.

of multi-kernel execution and background memory transfers on their cards [3]. This is not only due to thermal problems, but also because of the need to ensure that enough power is provided to the board during background transfers. To solve these problems a power budget matching mechanism is required to ensure certain levels of power consumption for stability (available in the latest 5xx Fermi boards).

We must also notice that, for the first years of the big "boom" of the GPU computing, the vast majority of applications ported to GPUs were massively parallel, with almost no data dependencies or synchronization between execution threads. However, programmers are running out of "simple" problems and regular parallel applications are currently being ported to GPUs. For these applications scalability is limited to a certain number of processors, depending on the input size. This leads to high amounts of power being wasted in useless work, idle cores or just spinning.

On the other hand we have legacy software. In order to squeeze all the GPU performance, programmers usually rely on fine-grain code optimizations for the specific hardware/compiler they are using. However, when there is a generational change in GPUs, these fine-grain optimizations are unable to make use of new resources. Recompiling is usually not enough, and re-optimizing the code for the new set of available cores/memory is required. This is a serious problem that not all companies can handle. Optimizing the GPU resources to match the program requirements in order to improve energy efficiency is an open field that may solve these problems.

Another source of wasted power are legacy graphic applications and engines. These applications usually produce huge amounts of frames per second (FPS) in modern hardware that the monitor cannot handle, and it has to randomly drop the frames that it cannot display. This sometimes causes a feeling of bumpy movement on the player. Wang already proposed to apply power gating in addition to DVFS to address this issue [4].

With the analysis performed in this paper we want to discover if this underutilization of resources is already happening and discuss different strategies that can be used do to improve energy efficiency and, if possible, performance. These strategies include, but are not limited, to voltage and frequency scaling and SM power gating. Results not only show that this underutilization is present, but also that resource optimization can increase the energy efficiency of GPU-based computation.

The rest of the paper is organized as follows. Section II provides some background on power and power-saving techniques and summarizes the related work. Section III reports our evaluation methodology and the main experimental results. Finally, Section IV shows our concluding remarks and future work.

II. Background and Related Work

In this section we will provide some background about power and power control mechanisms as well as discuss some related work in order to improve the energy efficiency of GPUs.

A. Power Dissipation

Like any other transistor-based electronic device, GPUs have two sources of power dissipation, dynamic and static power dissipation (or leakage). Dynamic power dissipation is proportional to usage (every time we access a structure), due to the constant charge and discharge of transistors. On the other hand, static power dissipation is derived from gate leakage and subthreshold leakage currents that flow even when the transistor is not in use.

As process technology advances toward deep submicron, the static power component becomes a serious problem, especially for large on-chip array structures such as caches or prediction tables. The leakage component is something that many studies do not take into consideration when dealing with temperature, but it cannot be ignored. For current technologies (under 32nm), even with gate leakage under control by using high-k dielectrics, subthreshold leakage has a great impact in the total power dissipated [5].

More specifically, for the GPU scenario, the total power dissipated by the GPU (without the DDRx[4] memory power) can be estimated as:

$$GPUPower = IdlePower + (SMsPower + MemPower)$$

where SMsPower is the power dissipated by the SM[5] units of the GPU and MemPower is the power dissipated by the on-die memory of the GPU (Instruction Cache, Shared Memory, L1 and L2).

SMsPower can be also decomposed into:

$$SMsPower = \sum_{i=0}^{NumSMs} (A * SMPower)$$

where A is the activity factor (or usage, that varies between 0 and 1), and SMPower is the combined power dissipated by each SM hardware (Fetch, Decode, Schedule stages plus the power disspated its Registers, FP, Integer, Load-Store and Special units).

B. Power Control Mechanisms

1) Dynamic Voltage and Frequency Scaling: Dynamic Voltage and Frequency Scaling (DVFS) has been widely used since the early 90's [6] offering a great promise to reduce energy consumption in microprocessors. DVFS relies on the fact that dynamic power dissipation depends on both

[4]Double Data Rate

[5]Streaming Multiprocessor

voltage and frequency ($P_D \approx V_{DD}^2 \cdot f$), and it dynamically scales these terms to save dynamic power [7][8].

However, as the building process goes into deep submicron, the margin between V_{DD} (supply voltage) and V_T (threshold voltage) is reduced, and as this margin decreases, the processor's reliability is reduced (among other undesirable effects). Moreover, the transistor's delay (switching speed) depends on: $\delta \approx 1/(V_{DD} - V_T)^\alpha$, with $\alpha > 1$. This means that we can lower V_{DD} for DVFS as long as we keep the margin between V_{DD} and V_T constant, so we can obtain the desired speed. However, the counterpart of reducing V_T is twofold: a) leakage power increases as it exponentially depends on V_T, which makes leakage an important source of power dissipation as the process technology scales below 65nm [9][10][11]; and b) processor reliability is further reduced.

Multicore architectures exhibit some peculiarities when running parallel workloads, especially in terms of power and performance. In such workloads threads must periodically synchronize (e.g., for communication purposes) and any delay introduced in one of the threads may end up delaying the whole application. Nevertheless, it is uncertain if future GPUs will implement per-core DVFS, as current generation of GPUs (i.e., Fermi from nVidia) even has limitations on manually setting frequencies for all the cores due to the complexity of the clock trees [12].

2) Clock Gating and Power Gating: Clock gating [13] adds an *AND* gate to the clock signal that powers an specific unit or structure with a control signal. If the control signal is on, the unit will be clocked as expected. If the unit is unneeded for a cycle or more, the control signal can be set to 0, in which case the unit will not be clocked.

On the other hand, power gating [14] is a hardware mechanism that turns off the supply voltage of a circuit block to lower leakage power dissipation by applying a "sleep" signal to the gate of the header or footer transistor of the circuit. This signal should be removed in order to restore the voltage of the circuit back to normal. Power gating is able to lower the leakage power dissipation of the circuit close to zero [15]. However, power gating is not free, as it implies area and energy overheads. The main source of area overhead comes from the header or footer transistor, that depends on the overall switching current of the target circuit (x3 according to [16]). We need that the circuit block stays in sleep mode long enough in order to compensate for the energy used in the header-footer transistor switching. This time is determined by the circuit design limits [17].

C. The Problem of Energy Efficiency on GPUs

As mentioned in the previous sections, energy efficiency in GPUs is high as long as the application uses the available resources, but there are almost no hardware mechanisms to optimize resources to application needs. Hong *et al.* [18] proposes a power and performance model that is used to select the number of optimal cores, based on the available memory bandwidth, however, this is not the only limitation as we mentioned previously, as input size, legacy software or data dependent software, may need different optimizations. In [19], Sheaffer *et al.* studied a thermal management for GPUs. Fu *et al.* [20] also presented some experimental data evaluation of GPUs. More recently Wang [21] *et al.* proposes an instruction-level energy estimation methodology for the GPU.

Table I: System Specs.

Core System	
Proc.:	Core2 duo 6850@3Ghz
GPU 0:	nVidia 7300GT
Memory:	4GB DDR2 @ 333Mhz
GPU 1: GTX 465	
GPU:	GTX 400
Process:	40 nm
Core Clock:	607 MHz
Memory Size:	1024 MB
Memory Clock:	1603 MHz
Memory Bus Type:	64x4 (256 bit)
Memory Bandwidth:	102.592 GB/sec
SMs:	11 (32 SPs[a] each)
Max Power Draw:	200 W
GPU 2: GTX 480	
GPU:	GTX 400
Process:	40 nm
Core Clock:	700 MHz
Memory Size:	1536 MB
Memory Clock:	1848 MHz
Memory Bus Type:	64x6 (384 bit)
Memory Bandwidth:	177.408 GB/sec
SMs:	15 (32 SPs each)
Max Power Draw:	250 W

[a]Streaming Processor

III. Experimental Results

A. Evaluation Environment

In this section we will describe our evaluation environment. We decided to use real hardware in order to perform the evaluation analysis of performance, temperature, power and energy of GPUs. Table I shows our main system specifications. GPU 0 is always present in the system during all tests, and is used only for graphics, while GPUs 1 and 2 are used to run CUDA kernels, but only one of them is connected to the motherboard during the tests. The reason we decided to use these cards is that both of them are equipped with same chip, codenamed GTX400, with 3200 million transistors in $529mm^2$ but with different bandwidth, memory size and, more importantly, number of active SMs. The GTX465 has 11 SMs enabled from the total 16 of the GTX400 chip, while the GTX480 has 15 active SMs. This will allow us to estimate the impact of power gating on the analyzed benchmarks.

Table II: Evaluated kernels and working sets.

	Benchmark	*Size*	*Benchmark*	*Size*
SPLASH-2	CUDA C Monte Carlo: Single Asian Option	default	Eigenvalues	default
	Simple Multi Copy and Compute	default	Vector Addition	default
	CUDA C 3D FDTD	default	Scalar Product	default
	Discrete Cosine Transform (DCT) (8x8)	default	Line of Sight	default
	1D Discrete Haar Wavelet Decomposition	default	CUDA Histogram	default
	Fast Walsh Transform	default		

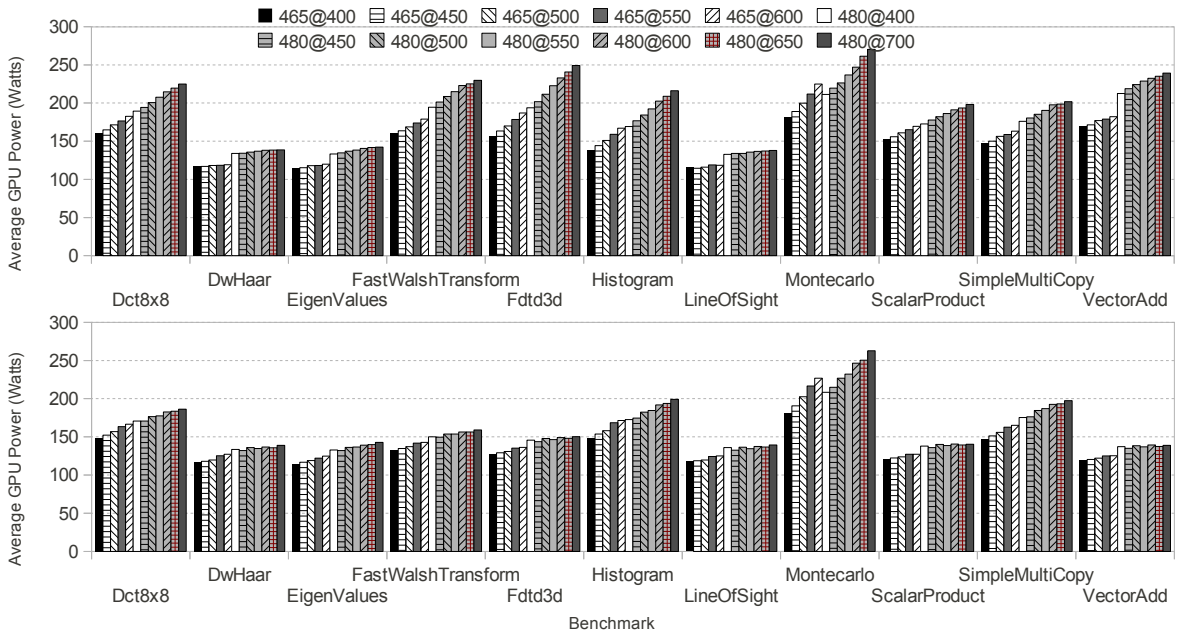

Figure 1: Avg. GPU power with (bottom) and without (top) memory transfers for different pairs of GPU/Frequency.

Power dissipation numbers will be provided by *Watts up? .Net* power meter [22]. This device is connected between the power source and the power supply of the system, and provides power dissipation information every second (at-the-wall power). Power information is logged by a different machine on the same room. Room temperature is controlled and set to 26ºC during the measurements to minimize temperature impact on static power. We decided not to isolate power dissipated by the GPU from the rest of the system (CPU, Motherboard, Memory, etc), as the hardware used in our experiments is always the same, and is required to execute the GPU kernels, so it's fair to account this hardware when optimizing for energy efficiency (Energy and Energy Delay Square Product -ED^2P- results). Some benchmarks may look more energy efficient with certain configurations if GPU power is isolated, but that's not true (Amdahl's Law). However, we will also provide isolated GPU power dissipation numbers to compare with the full system energy numbers.

We will use MSI Afterburner [23] tool to modify the working frequency of the SMs, as nVidia decided to remove the official support for frequency settings at a driver level. MSI Afterburner is only available for Microsoft Windows, so we will perform our analysis under Windows 7, using nVidia's driver v285.62.

Due to this software limitation we couldn't use GPU benchmarks such as Rodinia or Parboil as there is no Windows version, however, we would like to test these benchmarks in the future. We will be testing 11 kernels (see Table II) from the CUDA SDK v4.0.19 [24]. Kernels were randomly selected from a subset of 27 "potentially" useful kernels in real applications [25]. The SDK kernels are meant for performance measurements (execution takes miliseconds) and our logging hardware is only able to capture power at a coarse grain (every second). We decided to tweak the kernels increasing the inputs and iterating on the kernel calls so that the base execution lasted around 5 minutes. Due to time restrictions we had to leave 16 kernels out of this analysis, but we pretend to tweak them later on for our future work. Looping will be analyzed both with and

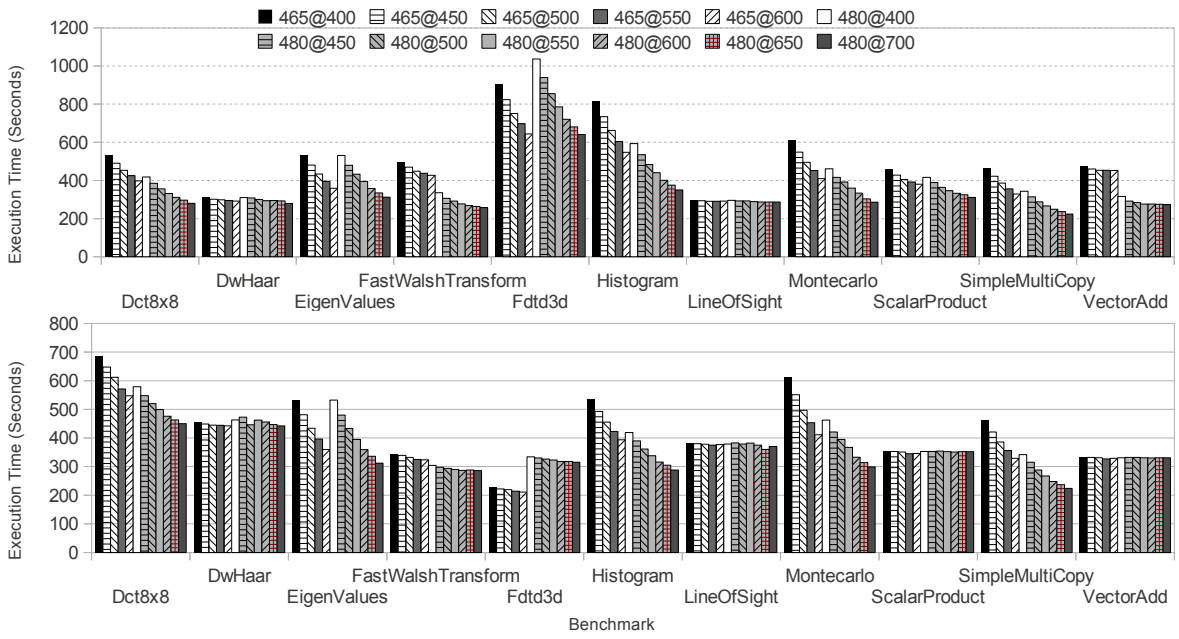

Figure 2: Total execution time with (bottom) and without (top) memory transfers for different pairs of GPU/Frequency.

without memory transfers.

B. Kernel Analysis

In this section we will analyze how the different kernels behave in terms of power, time, energy, energy delay square product, and temperature, both with and without accounting for memory transfers in the code. In addition, we will also compare the advantages and disadvantages of increasing frequency when compared to increasing the number of computation nodes (SMs) for the studied kernels.

Figure 1-top shows the isolated GPU power dissipation for different frequencies on both the GTX 465 and 480 graphic cards when we don't consider the memory transfers in the iterative kernel calls. We can clearly see that there are three benchmarks that don't make high usage of the available resources of the GPU, DwHaar, EigenValues and LineOfSight. The increment on power dissipation ($\sim 18W$) for these three benchmarks when changing from the GTX 465 to the 480 is mainly because of the additional leakage power from the 4 extra SMs and the additional 512MB of memory. We can also see a huge increment on power dissipation when changing cards in VectorAdd, however, for this benchmark, is mainly because of the additional bandwidth that allows more computation and also increments the power dissipation from the memory. Figure 1-bottom shows the isolated GPU power but considering memory transfers in the iterative kernel calls, so data is moved from main memory to device memory on every execution. With this configuration the real bottleneck is the PCI-e bandwidth, and that reduces the computation times and average power dissipation of the two cards. Only four benchmarks are left over 150W, Dct8x8, Histogram, Montecarlo and SimpleMultiCopy. This is important because, for the rest of the analyzed kernels, the card could be executing a different GPU-intensive kernel while the main one is moving data from-to memory or, as it happens with SimpleMultiCopy, overlapping kernel execution with memory transfers, doing small pieces of the problem and performing a "pipelined" execution of the problem.

Figure 2-top shows a performance analysis for the analyzed cards with different frequencies when not considering memory transfers. Please note that the number of iterations in the memory transfer and non-memory transfer charts vary in order to match the five minute execution to measure power, so do not try to compare execution time between them. As it happened with power, DwHaar and LineOfSight show almost no performance gains from neither frequency, bandwidth or core scaling. FastWalshTransform and VectorAdd obtain great benefits from the additional cores, memory and bandwidth, but less benefit from frequency scaling (because they are memory bound and memory frequency is not scaled). For Dct8x8, Histogram, Montecarlo, ScalarProduct and SimpleMultiCopy, the performance benefits of frequency scaling and additional cores are consistent. Numbers also show that we can trade 4 SMs for 150Mhz of frequency, and obtain similar performance numbers. More importantly we can see how EigenValues behaves exactly the same despite the number of available SMs. This is a clear example of a kernel that is running on less SM cores that it was designed for (legacy code),

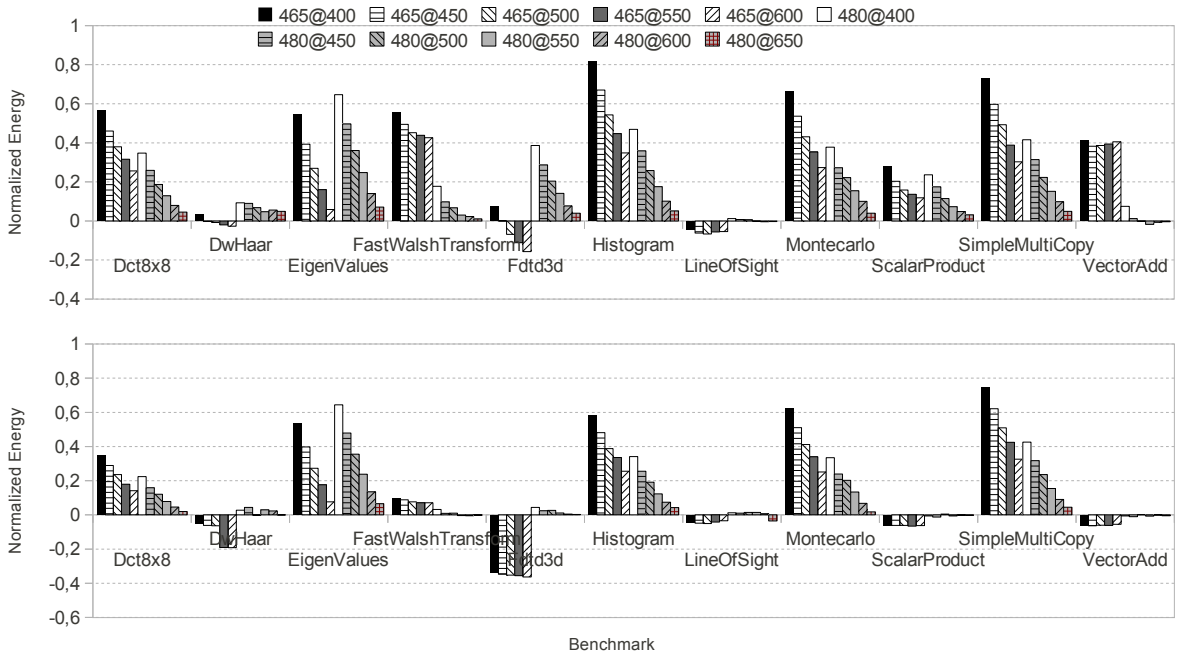

Figure 3: Normalized energy with (bottom) and without (top) memory transfers for different pairs of GPU/Frequency.

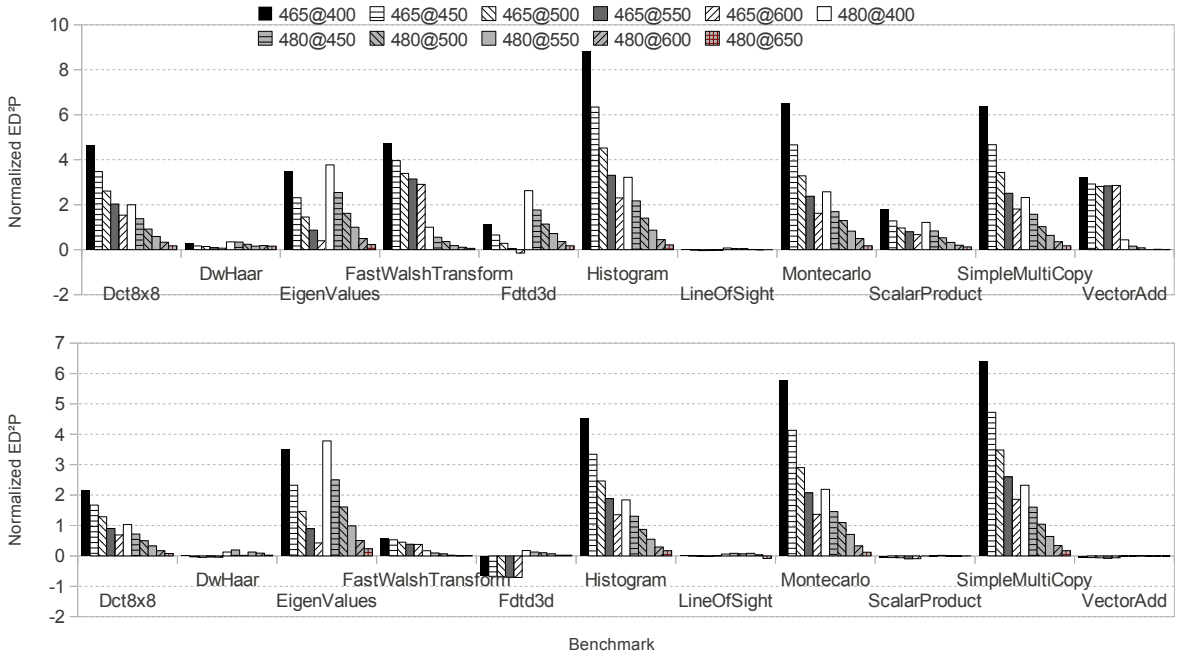

Figure 4: Normalized ED^2P with (bottom) and without (top) memory transfers for different pairs of GPU/Frequency.

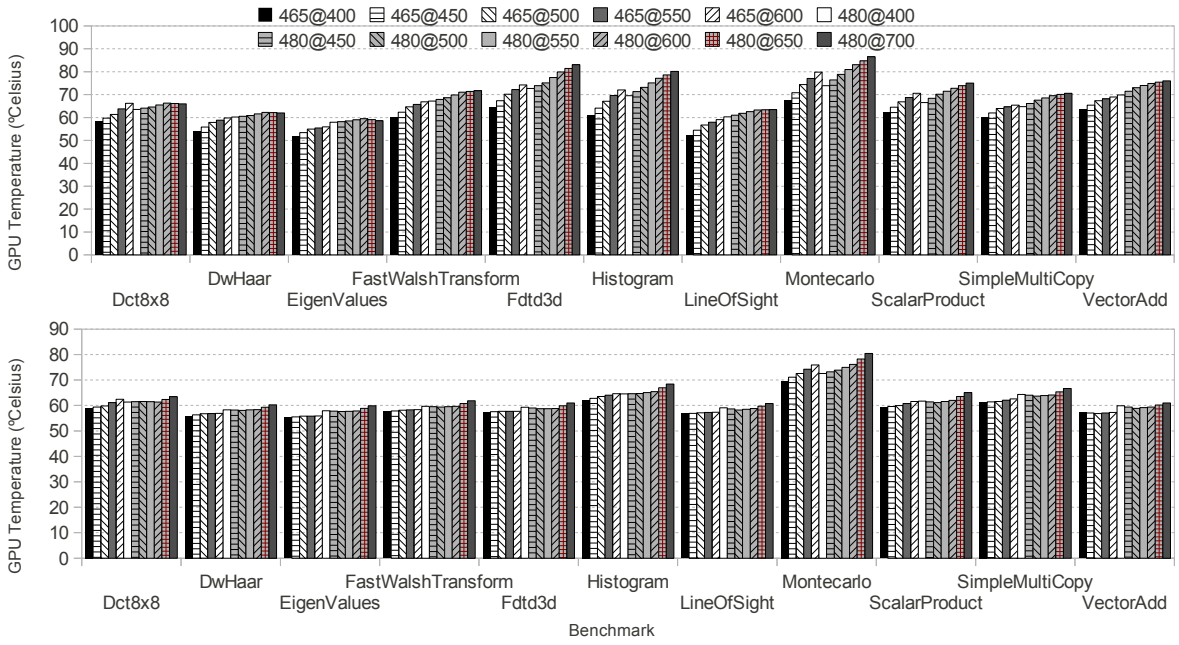

Figure 5: Average temperature with (bottom) and without (top) memory transfers for different pairs of GPU/Frequency.

trading cores for frequency could improve it's performance and energy efficiency. For Fdtd3d it is even worst, as having more core results on less performance, mainly because of data sharing. This is an example of applications that don't scale any further than a certain number of cores, at least for the analyzed inputs. We will discuss more in detail how can we optimize cores/frequency later on this section, with the performance/costs analysis. Results when considering transfers are similar (Figure 2-bottom), except for FastWalshTransform and VectorAdd, where the performance improvement from the internal memory bandwidth and cores is hidden by the slow external memory transfers.

Figure 3 shows normalized energy numbers for the full system (i.e., motherboard, CPU, GPU, etc) both without (top) and with (bottom) including memory transfers on the iterative kernel calls. Results are normalized to the GTX 480 card running at default frequency (700Mhz). In terms of energy, maximizing both the number of cores and frequency provides the best results for the majority of the kernels, only kernels like Fdtd3d, DwHaar and LineofSight obtain net energy benefits from a core reduction if we don't consider memory transfers. However, when transfers are considered, ScalarProduct and VectorAdd that had energy loses of 30-40% experience net benefits of 5% in terms of energy. We obtain similar results when using the metric energy delay square (Figure 4), with even lower net benefits from core reduction when we don't consider transfers. These results give us an idea on how much we need to optimize power on the GPU without incurring on any performance degradation if we want to obtain net energy benefits. The leakage power of the system every additional second ($\sim 138W$) accounts for much more than what we can save from removing one SM or reducing frequency by 50Mhz.

Finally, Figure 5 shows the temperatures of the two analyzed cards when running the studied kernels, with and without considering memory transfers. In order to measure temperature we forced the GPU fan to 70% of its capacity. Temperatures are obtained from the internal sensors of the GPU. Looking at the temperature numbers we can see that for most of the kernels the GPU resources are underutilized. Please note that the benchmarks have been running at least 5 minutes on the GPU so the temperature has been rising during the first 2-3 minutes and after 3 minutes is stable. In fact, only three benchmarks go over 70C, with only one benchmark reaching a peak temperature of 85C (Monte-Carlo), potentially giving us the chance of further increasing the frequency in most of the benchmarks. However, we must to be careful because these are coarse-grain sensors, and further increasing frequency may cause damage to the chip due to hot-spots. A more detailed modeling of the die (layout and power maps) is required to ensure proper "overclocking" without damaging the GPU.

C. Resource Optimizations for Energy Efficiency

This section analyzes the power costs and performance improvements of two different strategies to improve energy efficiency, increasing/decreasing frequency and/or available SMs. Figure 6 shows the speedup provided by an increment

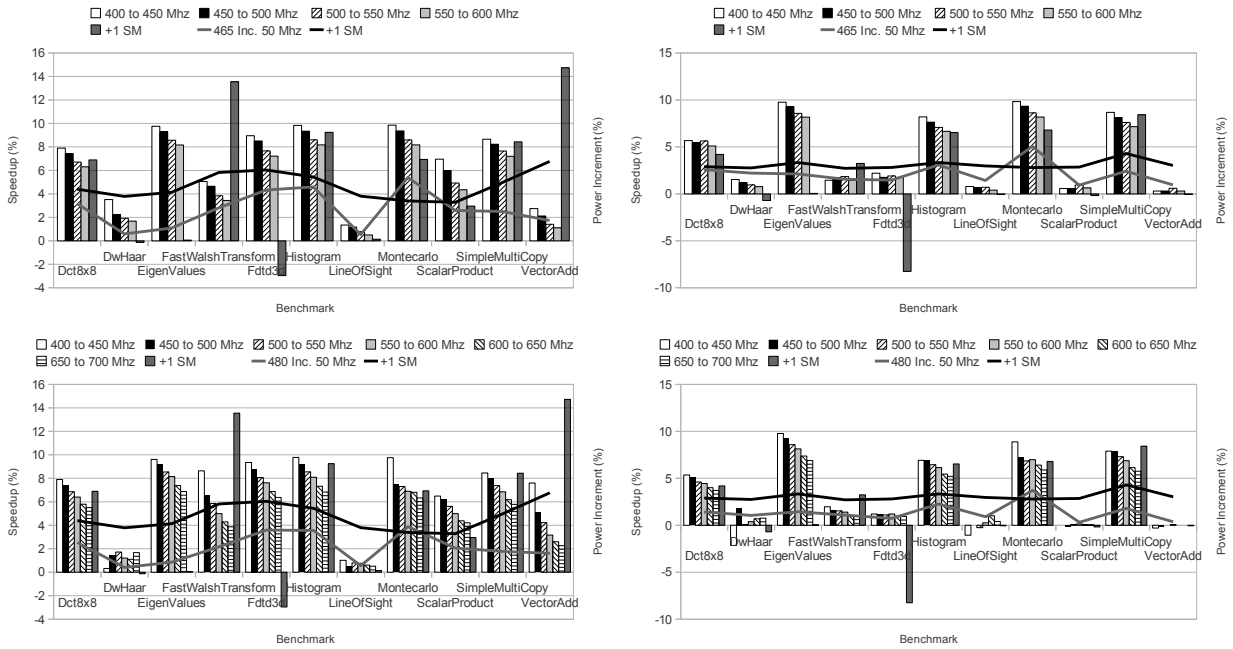

Figure 6: Speedup obtained by increasing frequency by 50Mhz or increasing the number of SMs by one vs Power costs for the GTX 465 (top) and the GTX 480 (bottom) with (right) and without (left) memory transfers.

of 50Mhz on the analyzed kernels (bars). Improvements of SMs frequency are not linear because memory frequency is not scaled at the same rate as cores in our analysis. On top of the speedup we draw two lines that represent the GPU power increment of increasing frequency by 50Mhz (black), or adding an extra SM (gray). As we mentioned in the previous section (time analysis), the high increment in performance from FastWalshTransform and VectorAdd is not only due to the additional SM, but because of the total available memory and the increased memory bandwidth, so let us ignore these benchmarks in the analysis.

For the both the GTX 465 and 480 cards when not considering memory transfers it is clear that we could trade cores for frequency on benchmarks like DwHaar, EigenValues, Fdtd3d and ScalarProduct. For Montecarlo, the cost of frequency is higher than adding an additional SM, but also the performance improvement, so in terms of energy it is balanced. SimpleMultiCopy can get benefits if it trades cores for frequency at low frequencies, as the performance increase is similar but the power costs of the additional SM is higher. Please note that the SM performance estimations also consider an increase in memory bandwidth. For a fairer comparative we should be also increasing memory bandwidth by increasing memory frequency, or just consider the bar that corresponds to the frequency increase from 400 to 450Mhz, as it is the one less affected by bandwidth limitations. Similar results are obtained when considering memory transfers on both cards. In order to optimize energy efficiency of the GPU we require an scheduler that is able to detect the kernel requirements and modify frequency (SMs & memory) and available SMs.

IV. Conclusions and Future Work

GPUs are massively parallel processors that have proved to be a useful tool in scientific research. Up until now GPU design has focused on performance and throughput, as the applications executed on the GPU were able to use all the available resources. However, due to memory bandwidth limitations, problem sizes, synchronization, legacy software, etc, GPUs can be underutilized, and have almost no power-saving hardware to optimize energy efficiency (besides coarse grain DFS and clock gating). In this paper we analyze performance, power, energy and temperature in order to confirm this hypothesis.

Results show that different kernels have different resource requirements. Off-card bandwidth is a key limiting factor when memory transfers are taken into account for half of the analyzed kernels. Moreover, dynamically modifying the working frequencies and available SMs GPU computation can be more energy efficient than it is right now. This optimization not only affects power, but sometimes also performance if the application is not optimized for new hardware. The next step would be to design a dynamic scheduler that analyzes kernel requirements at runtime and selects the optimal amount of cores/frequency.

ACKNOWLEDGMENT

This work was partially supported by the Spanish MICINN and European Commission FEDER funds under projects Consolider Ingenio-2010 CSD2006-00046 and TIN2009-14475-C04, and also by the Fundación Séneca (Agencia Regional de Ciencia y Tecnología, Región de Murcia) under project 15290/PI/2010.

REFERENCES

[1] H. Esmaeilzadeh, E. Blem, R. S. Amant, K. Sankaralingam, and D. Burger, "Dark silicon and the end of multicore scaling," in *Proceedings of the 38th International Symposium on Computer Architecture*, June 2011.

[2] S. Huang, S. Xiao, and W. Feng, "On the energy efficiency of graphics processing units for scientific computing," in *Proceedings of the 2009 IEEE International Symposium on Parallel&Distributed Processing*, ser. IPDPS '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 1–8. [Online]. Available: http://dx.doi.org/10.1109/IPDPS.2009.5160980

[3] W. B. Dally, "Power, programmability, and granularity: The challenges of exascale computing," in *IEEE International Parallel & Distributed Processing Symposium Keynote*, 2011.

[4] P.-H. Wang, C.-L. Yang, Y.-M. Chen, and Y.-J. Cheng, "Power gating strategies on gpus," *ACM Trans. Archit. Code Optim.*, vol. 8, pp. 13:1–13:25, October 2011. [Online]. Available: http://doi.acm.org/10.1145/2019608.2019612

[5] S. Kaxiras and M. Martonosi, *Computer Architecture Techniques for Power-Efficiency*. Morgan and Claypool Publ., 2008.

[6] P. Macken, M. Degrauwe, M. Van Paemel, and H. Oguey, "A voltage reduction technique for digital systems," in *Proceedings of the 37th IEEE International Solid-State Circuits Conference. Digest of Technical Papers*, 1990, pp. 238–239.

[7] T. Simunic, L. Benini, A. Acquaviva, P. Glynn, and G. de Micheli, "Dynamic voltage scaling and power management for portable systems," in *Proceedings on Design Automation Conference*, 2001, pp. 524–529.

[8] Q. Wu, P. Juang, M. Martonosi, and D. W. Clark, "Voltage and frequency control with adaptive reaction time in multiple-clock-domain processors," in *Proceedings of the 11th International Symposium on High-Performance Computer Architecture*, 2005, pp. 178–189.

[9] M. J. Flynn and P. Hung, "Microprocessor design issues: Thoughts on the road ahead," vol. 25, no. 3, pp. 16–31, 2005.

[10] A. et al. Keshavarzi, "Intrinsic iddq: Origins, reduction, and applications in deep sub- low-power cmos ic's," in *Proceedings of the IEEE International Test Conference*, 1997.

[11] N. S. Kim, T. Austin, D. Baauw, T. Mudge, K. Flautner, J. S. Hu, M. J. Irwin, M. Kandemir, and V. Narayanan, "Leakage current: Moore's law meets static power," *Computer*, vol. 36, no. 12, pp. 68–75, 2003.

[12] C. Zander, "Clock manipulation on fermi and newer gpus. website: http://www.phoronix.com/scan.php?page=news_item&px=otgxnq."

[13] A. CHANDRAKASAN and R. RODERSEN, *Low Power CMOS Design*. IEEE Press, 1998.

[14] A. P. Chandrakasan, W. J. Bowhill, and F. Fox, *Design of High-Performance Microprocessor Circuits*, 1st ed. Wiley-IEEE Press, 2000.

[15] M. Powell, S.-H. Yang, B. Falsafi, K. Roy, and T. N. Vijaykumar, "Gated-vdd: A circuit technique to reduce leakage in deep-submicron cache memories," in *Proceedings of the International Symposium on Low Power Electronics and Design*, 2000.

[16] A. Iyer, "Demystify power gating and stop leakage cold. website: http://www.powermanagementdesignline.com/howto/181500691."

[17] Z. Hu, A. Buyuktosunoglu, V. Srinivasan, V. Zyuban, H. Jacobson, and P. Bose, "Microarchitectural techniques for power gating of execution units," in *Proceedings of the 2004 international symposium on Low power electronics and design*, ser. ISLPED '04. New York, NY, USA: ACM, 2004, pp. 32–37. [Online]. Available: http://doi.acm.org/10.1145/1013235.1013249

[18] S. Hong and H. Kim, "An integrated gpu power and performance model," in *Proceedings of the 37th annual international symposium on Computer architecture*, ser. ISCA '10. New York, NY, USA: ACM, 2010, pp. 280–289. [Online]. Available: http://doi.acm.org/10.1145/1815961.1815998

[19] J. W. Sheaffer, D. Luebke, and K. Skadron, "A flexible simulation framework for graphics architectures," in *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, ser. HWWS '04, 2004.

[20] R. Fu, A. Zhai, P. chung Yew, and W. chung Hsu, "Reducing queuing stalls caused by data prefetching," 2007.

[21] Y. Wang and N. Ranganathan, "An instruction-level energy estimation and optimization methodology for gpu," in *Proceedings of the 2011 IEEE 11th International Conference on Computer and Information Technology*, ser. CIT '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 621–628. [Online]. Available: http://dx.doi.org/10.1109/CIT.2011.69

[22] "Wattup .net power meter. website: https://www.wattsupmeters.com/."

[23] "Msi afterburner overclocking tool. website: http://event.msi.com/vga/afterburner/index.htm."

[24] "Cuda sdk. website: http://developer.nvidia.com/gpu-computing-sdk."

[25] A. Kerr, G. Diamos, and S. Yalamanchili, "A characterization and analysis of ptx kernels," in *Proceedings of the 2009 IEEE International Symposium on Workload Characterization (IISWC)*, ser. IISWC '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 3–12. [Online]. Available: http://dx.doi.org/10.1109/IISWC.2009.5306801