Energy-Aware High Performance Computing with Graphic Processing Units

Mahsan Rofouei, Thanos Stathopoulos, Sebi Ryffel, William Kaiser, Majid Sarrafzadeh
{mahsan,Thanos,majid}@cs.ucla.edu, {sebi,Kaiser}@ee.ucla.edu
University of California, Los Angeles

***Abstract* – The use of Graphics Processing Units (GPUs) in *general purpose computing* has been shown to incur significant performance benefits, for applications ranging from scientific computing to database sorting and search. The emergence of high-level APIs facilitates GPU programming to the point that general purpose computing with GPUs is now considered a *viable system design* and programming option. Nevertheless, the inclusion of a GPU in general purpose computing results in an associated increase in the system's *power budget*. This paper presents an experimental investigation into the power and energy cost of GPU operations and a cost/performance comparison versus a CPU-only system. Through real-time energy measurements obtained using a novel platform called LEAP-Server, we show that using a GPU results in *energy savings* if the performance gain is above a certain bound. We show this bound for an example experiment tested by LEAP-Server.**

I. Introduction

Graphic Processing Units (GPUs) are special-purpose programmable parallel architectures, primarily designed for real-time rasterization of geometric primitives. Due to their highly parallel design and dedicated computational nature, GPUs have recently been used for scientific, bioinformatics and database applications, including sorting and searching [5,6], increasing performance by at least an order of magnitude compared to conventional CPUs. This vast performance increase, combined with the wide availability of GPUs and the existence of high-level APIs such as the NVDIA CUDA [4] present system designers with a very appealing performance improvement solution.

From the perspective of *energy consumption* however, the choice of shifting computational load to a dedicated co-processor becomes more complex. As a sophisticated hardware component with multiple parallel elements, the GPU requires significant power to operate. In addition to requiring expensive cooling solutions so as to control heat dissipation, modern GPUs also require a dedicated direct connection to the power supply. From a system design perspective, the performance increase offered by the inclusion of an additional hardware component must be balanced by the associated energy cost induced by the new component.

This paper presents an experimental investigation into the performance and energy efficiency of a combined CPU-GPU system. Our goal is to characterize the conditions under which the inclusion of a GPU component is beneficial, from *both* a performance *and* an energy efficiency perspective. Our investigation is based on *LEAP-Server*, a novel architecture that incorporates standard server functionality with high-fidelity, real-time energy monitoring of individual system components, such as the CPU, GPU, motherboard and RAM. Through real-time energy measurements obtained by LEAP-Server, we show that, despite an increase in total system power, using a GPU is more *energy efficient* when the performance improvement is above a certain bound which depends on application specific factors, compared to a CPU-only solution. We also use an analytical model to derive maximum throughput when having both CPU and GPU executing tasks. Through our experiments we also demonstrate the value of a real-time measurement system such as LEAP-Server in selecting the best performance-energy operating point in real time.

The paper is organized as follows. Section II describes the importance of general purpose computing on GPUs and gives a brief summery on GPU architecture. Section III describes the experimental setup used for our experiments, while section IV shows several experiments and their results. Finally, conclusions are drawn in section V.

II. Importance of GPGPU and GPU Architecture

General purpose computing with graphic processing units (GPGPU) has enabled orders of magnitude speedups over the conventional CPUs for various applications in science and engineering. With the progress on the level of programmability, support for IEEE floating-point standards and arbitrary memory addressing, GPUs now offer new capabilities beyond the graphic applications which they were initially designed for. Recently the challenges in GPGPU

community have revolved around the constraints of the programming environment and on optimal mapping of applications so to best leverage the highly parallel GPU architecture.

CUDA [4] (Compute Unified Device Architecture) is a new API that is designed to facilitate GPU programming for general purpose tasks. CUDA allows programmers to implement algorithms in a data-parallel programming model. CUDA treats the GPU as a coprocessor that executes data-parallel functions, also known as *kernel functions*. The source program is divided into host (CPU) and kernel (GPU) code, which are then compiled by the host compiler and NVIDIA's compiler (nvcc). The common execution path for an application on a combined CPU-GPU system is as follows:

1) Allocate memory on GPU-exclusive memory
2) Transfer data from CPU to GPU
3) Execute kernel on the GPU
4) Transfer results back from GPU to CPU.

The rest of this Section presents a brief summery of NVIDA's G80 GPU architecture, which supports the single-program, multiple-data (SPMD) programming model. G80 graphics processing unit architecture was first introduced in NVIDIA'S GeForce 8800 GTS and GTX graphics cards.

The G80 GPU consists of 16 streaming multiprocessors (SMs), each containing eight streaming processors (SPs). Each SM has 8,192 registers and 16KB of on-chip memory that are shared among all threads assigned to the SM. The threads on a given SM's cores execute in SIMD fashion, with the instruction unit broadcasting the current instruction to the eight cores. Each core has a single arithmetic unit that performs single-precision floating-point arithmetic and 32-bit integer operations.

In order to reduce the application's demand for off-chip memory bandwidth, there are several on-chip memories that can be employed to exploit the data locality and data sharing. Each SM has shared memory for data that is either written and reused or shared among threads. The constant memory space is cached. Finally, for read-only data that is shared by many threads but not necessarily accessed simultaneously by all threads, the off-chip texture memory and the on-chip texture caches exploit 2D data locality to substantially reduce memory latency.

Applications that can benefit from the above described SPMD model can result in very high speedups. There are numerous speedup reports in variety of application domains. Some examples are image registration for medical imaging, numerical algorithms, fluid simulation and molecular simulation [8,9,10]. Many types of CT reconstruction algorithms are successfully accelerated on commodity graphical graphics hardware. RapidTC [7] can greatly benefit from the SIMD parallelism that GPU provides. It was demonstrated that both iterative and non-iterative algorithms suite the GPU architecture well.

The aforementioned prior work indicates that GPUs are good platforms for executing parallelizable applications. However, as an extra piece of hardware they incur a cost, mostly in terms of power. In this paper we aim to find conditions where it is beneficial to add GPUs to an existing system both in terms of performance and energy consumption.

III. Experimental Setup

In this Section, we will describe the LEAP-Server platform that was used in our experiments and also provide information on the particular applications used as workloads for our experiments.

A. Real-time Energy Measurements with LEAP-Server

LEAP-Server is the adaption of the embedded low power energy-aware processing (LEAP) project [1] to desktop and server-class systems. LEAP-Server differs from previous approaches such as PowerScope [11] in that it provides both real-time power consumption information and a standard application execution environment *on the same platform*. As a result, LEAP-Server eliminates the need for synchronization between the device under test and an external power measurement unit. Moreover, LEAP-Server provides power information of *individual subsystems*, such as CPU, GPU and RAM, through *direct measurement*, thereby enabling accurate assessments of software and hardware effects on the power behavior of individual components.

The LEAP-Server platform used in our experiments is equipped with an Intel® Core™ 2 Duo CPU E7200 with 3MB of shared L2 cache, 2GB 800MHz DDR2 SDRAM and a NVIDIA® CUDA™ enabled graphics processor. Power measurements are performed by an NI PCI-6024E DAQ card capable of sampling 200kS/s with a resolution of 12bit. In order to measure the energy consumption of individual subsystems, we inserted 0.1 Ohm sensing resistors in all the DC outputs of the power supply---3.3, 5 and 12V rails. Components such that are powered through the motherboard such as SDRAM DIMS are placed on riser cards in order to gain access to the voltage pins. Power measurements are obtained by first deriving the *current* flowing over the sensing resistors through voltage measurements across the resistors and then multiplying with the measured voltage on the DC power connector. The DAQ card autonomously samples the voltages at the specified frequency and stores them in its buffer. A Linux driver periodically initiates a DMA transfer of the

buffer's content to main (kernel) memory. The module then exports the values to user space, where the power is calculated and integrated over time. Figure 1 depicts the architectural diagram of the LEAP-Server.

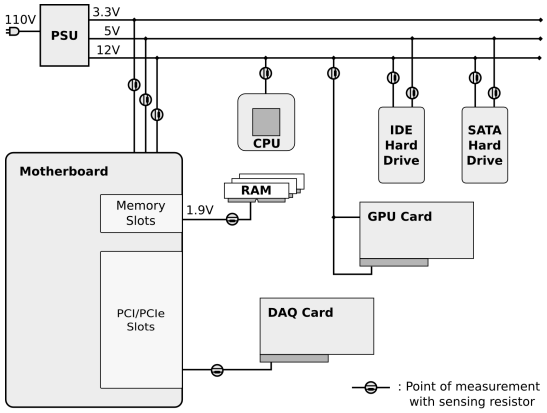

Fig.1 LEAP-Server Architectural Diagram

It must be noted that LEAP-Server utilizes the main CPU to process the power information, unlike the LEAP2 platform which contains a dedicated ASIC for this task. As a result, care must be taken so that the task of energy measuring does not create a negative performance---or energy---impact in the rest of the system. The performance overhead is directly related to the sampling rate as more samples result in higher amounts of data that need to be transferred to the CPU and processed. Experiments showed that sampling above 500Hz per channel does not result in any significantly higher accuracy. At 500Hz, the CPU performance penalty was under 3%.

B. GPU Applications

Making the correct decision in choosing the best platform in order to meet both performance and energy goals depends on the execution times on each platform. In situations where the GPU can finish a task in a very small period compared to its CPU counterpart, the performance gain results in energy savings as well, making the GPU a preferred choice. However, when the GPU speedup is not as pronounced and as rich the execution times on CPU and GPU are comparable, choosing the right approach is more complex. Based on this, for our experiments we categorized applications in two major groups: first, applications that benefit from high speedups when using the GPU implementation compared to their CPU implementation and second, applications resulting in lower speedups. For the purpose of our experiments, we consider speedups of 5x and higher as high speedup applications. Section IV will give more accurate criteria for distinguishing between these two categories. All the applications chosen are from the CUDA developer SDK examples [2,3]. We do note that that the CPU and GPU implementations in these examples are not necessarily fully optimized; however, their wide availability makes them good candidates for experimentation.

High Speedup Applications: We have chosen *separable convolution* to represent this category. Convolutions are used by a wide range of systems in engineering and mathematics. Many algorithms in edge detection use convolutional filtering. Separable filters are a special case of general convolution in which the filter can be expressed in terms of two filters, one on rows and the other on the columns of the image. In image processing, computing the scalar product of input signals with filter weights in a window surrounding output pixels is a highly parallelizable operation and results in good speedup using GPUs. The GPU speedup over its CPU counterpart as implemented in CUDA SDK is 30-36x [2].

Low Speedup Applications: The Prefix-sum (scan) algorithm is one of the most important building blocks for data-parallel computation. Its applications include parallel implementations of deleting marked elements from an array (stream-compaction), sort algorithms (radix and quick sort), solving recurrence equations and solving tri-diagonal linear systems. In addition to being a useful building block, the prefix-sum algorithm is a good example of a computation that seems inherently sequential, but for which there are efficient data-parallel algorithms [3]. In our experiments we use the version implemented to use for large arrays of arbitrary size. The result of an array scan is another array where each element is the partial sum of all elements up to and including j (inclusive scan). If the j^{th} element is not included the scan is exclusive. The speedup of the SDK example over a CPU implementation is around 2-6x [3].

IV. Experimental Results

In this Section, we present our experimental results, based on the application categories described in the previous Section. Figure 2 shows a sample result of a LEAP-Server experiment on the separable convolution example. In all our experiments we account for the memory transfers to the GPU when computing the energy. The data in all cases fits in the GPU memory and a single transfer at the beginning is sufficient to copy data to the GPU. We copy the results back in the end.

A. Idle power and event frequency analysis

A well-engineered and energy optimized system would place an unused hardware asset to its lowest possible power state, while still retaining a quick reaction time, to account for an unanticipated increase in the workload. During the course of our experiments, our LEAP-Server energy managements indicated the

GPU was *not* placed in a low-power state when not used; rather it was placed in its peak power state, thereby dissipating a higher amount of energy without any actual benefit.

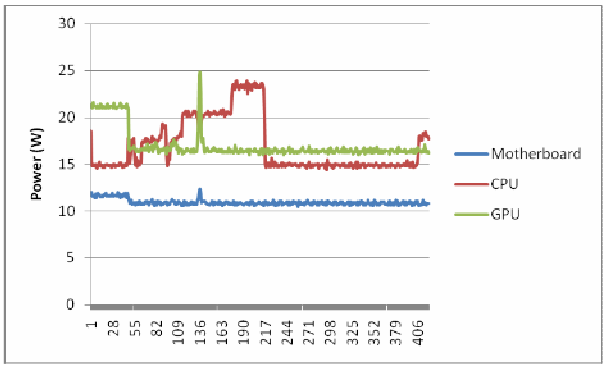

Fig. 2 LEAP-Server Output for Separable Convolution.

Inclusion of an additional component in a system imposes at least idle power consumption when not used. Therefore assuming the component uses its idle power when not used, there should be a minimum bound on the number of events executed on the GPU in order to balance between the performance gain and energy cost.
The amount of energy consumed for performing separable convolution on a 3072 × 3072 data with kernel radius of 8 on CPU and GPU are as follows:

E_{CPU}= 17.74 J, E_{GPU} = 2.13J

In a timeframe of 10s, the addition of a GPU to the system will add an extra 16500 J to the total energy consumed. In order to benefit from adding an extra component from energy prospective we must therefore have at least 11 GPU kernel calls. We take this idle power into account in all our experiments.

B. High-speedup applications

In applications having significant differences in their execution times on CPU and GPU where the energy consumed for data transfer is negligible, the performance benefits will result in significant energy saving benefits as well. Changing system or application parameters that increase the performance of an application over its CPU counterpart result in lower execution times and so decrease the energy consumption of the overall system. Our experiments on separable convolution confirm this. Table 1, shows the effect of different memory usage of the separable convolution SDK example on both performance and energy consumption.

Table. 1 Performance-Energy Comparison in Separable Convolution

	Performance (MPix/sec)	Energy on GPU (J)	Energy on CPU (J)
Convolution (shared)	346	2.44	12.91
Convolution (Texture)	215	2.76	12.91

As Table 1 shows, due to high performance on the GPU both implementations (shared and texture) result in less energy consumption. Optimizing performance using shared memory therefore also uses less energy.
We present a first-order analysis to determine how to distinguish applications that fit into this category.

$$E_{CPU} = t_{CPU} \times P_{avg-CPU} \quad (1)$$

$$E_{GPU} = t_{GPU} \times (P_{avg-GPU} + P_{idle-CPU}) + E_{transfer} \quad (2)$$

From Equations 1 and 2, if the energy consumed for transfer can be neglected we have:

$$\text{Speedup} = \frac{t_{CPU}}{t_{GPU}} < \frac{P_{avg-GPU} + P_{idle-CPU}}{P_{avg-CPU}} \quad (3)$$

As Equation 3 indicates there is no constant bound on the limit of the application speedup since $P_{avg\text{-}GPU}$ and $P_{avg\text{-}CPU}$ can change based on the specific characteristics of an application. Therefore the presence of a real time measurement system such as LEAP-Server can identify the best performance-energy choice real-time.

C. Break-even point for power-performance graph

In applications having low speedups where the CPU execution time is comparable with GPU execution time, there is a break-even point for the power-performance graph.
Our experiments on the *scan* SDK example confirm this. The total energy consumed for performing scan on 1000000 elements is 0.13J on CPU and 0.16J on the GPU. The speedup in this case is 2.23. This example demonstrates a case where it is beneficial to run the task on CPU. The results can be seen in Fig. 3. In practice, a well-engineered system can have the CPU executing other tasks while waiting for the results from the GPU task. Thus, adding the CPU idle power to the total power can bias the results in favor of a CPU-only approach. On the other hand, we cannot eliminate the CPU power completely by turning the CPU off, since a GPU acts as a coprocessor to the CPU and as such

requires the CPU to transfer data to/from it and execute kernels on it. In the case that both a CPU and GPU execute tasks, we consider the following scenario.

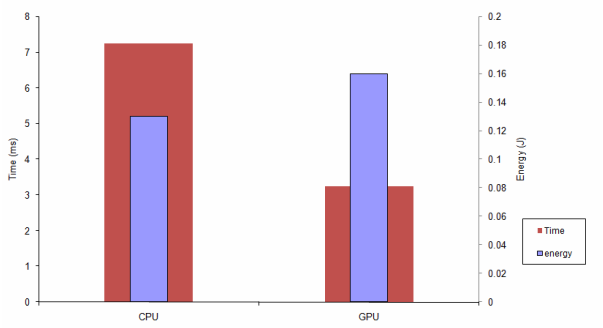

Fig. 3 Energy and performance comparison of *Scan*

We have two tasks T_1 and T_2 that take t_{1C} and t_{2C} to execute on a single core CPU and t_{1G} and t_{2G} to execute on the GPU. In the first case that we execute both tasks on the CPU, the overall energy consumed is computed as below:

$$E_1 = (t_{1C} + t_{2C}) \times P_C \qquad (4)$$

Now if we have T_1 executing on the CPU and T_2 on GPU, we will have two cases: If $t_{2G} < t_{1C}$:

$$E_2 = t_{2G} \times (P_G + P_C) + (t_{1C} - t_{2G}) \times (P_{G-Idle} + P_C) \quad (5)$$

And if $t_{2G} > t_{1C}$ we will have:

$$E_2 = t_{1C} \times (P_G + P_C) + (t_{2G} - t_{1C}) \times (P_{C-Idle} + P_G) \quad (6)$$

Here the problem becomes similar to the well-studied category of task scheduling. This problem can also be extended to multiple core CPU and GPU systems.

D. Discussion

There is a huge potential of research in the field of energy-aware high performance computing with GPUs. For parallel applications that suit the GPU's architecture, GPUs are good choices both from performance and energy consumption prospective. Nevertheless in order to optimize energy consumption based on the specific application there are several considerations to make.

As described in Section II, there are constant, texture and global memories available to use by the streaming multiprocessors. Based on the application and the locality and frequency of data usage pattern the programmer can choose from the cached memories (constant and texture) or global memory or choose to copy data to shared memory to increase performance. Similar categorization as done in Sections IV.B and C can be done in terms of memory access patterns. Here again the memory choice can affect total system energy consumption. Finally, the order of execution of tasks on the CPU-GPU system, the length of CPU idle time and task scheduling are factors to consider in a heterogeneous system composed of CPUs and GPUs. Based on our experiments, a real-time measurement system such as LEAP-Server can be very helpful in making correct decisions.

V. CONCLUSION

Graphic processing units have been introduced as high computational units offering high throughputs for a broad range of applications in science and engineering. In this paper we investigated the use of GPUs from the perspective of energy consumption as well as performance. Our experiments are based on a real-time measurement system called LEAP-Server and we show that being able to monitor energy real-time can have beneficial results in choosing appropriate platforms for different applications both from performance and energy consumption prospective.

REFERENCES

[1] Thanos Stathopoulos, Dustin McIntire, and William J. Kaiser. The energy endoscope: Real-time detailed energy accounting for wireless sensor nodes. In IPSN, 2008.

[2] V.Podlozhnyuk. Image Convolution with CUDA. NVIDIA whitepaper

[3] M. Harris. Parallel Prefix Sum (Scan) with CUDA. NVIDIA whitepaper.

[4] NVIDIA Corporation. NVIDIA CUDA Programming Guide. 2007.

[5] W. Liu, B. Schmidt, G. Voss, A.Schroder, and W. Muller-Wittig. Bio-Sequence Database Scanning on a GPU. 20th IEEE IPDPS, 2006

[6] N. Govindaraju, J. Gray, Ritesh Kumar, and Dinesh Manocha. GPUTeraSort: High Performance Graphics CoprocessorSorting for Large Database Management. In ACM SIGMOD, June 2006.

[7] K. Mueller and F. Xu. Practical considerations for GPU-accelerated CT. IEEE Symp. Biomedical Imaging (ISBI'06), pp. 1184-1187, 2006.

[8] Sanjiv S. Samant, Junyi Xia, Pinar Muyan-Özçelik, John D. Owens. High performance computing for deformable image registration: Towards a new paradigm in adaptive radiotherapy. Medical Phisics, 2008.

[9] Naga K. Govindaraju , Dinesh Manocha, Cache-efficient numerical algorithms using graphics hardware, Parallel Computing, v.33 n.10-11, p.663-684, November, 2007

[10] C. I. Rodrigues, D. J. Hardy, J. E. Stone, K. Schulten, W. W. Hwu., GPU acceleration of cutoff pair potentials for molecular modeling applications. Proceedings of the 2008 Conference On Computing Frontiers, pp.273-282, 2008.

[11] J. Flinn and M. Satyanarayanan. Powerscope: a tool for profiling the energy usage of mobile applications. In Second IEEE Workshop on Mobile Computing Systems and Applications, Feb. 1999.