Energy Analysis of Parallel Scientific Kernels on Multiple GPUs

Sayan Ghosh
Dept. of Computer Science
University of Houston
Houston, Texas
sghosh@mail.uh.edu

Sunita Chandrasekaran
Dept. of Computer Science
University of Houston
Houston, Texas
sunita@cs.uh.edu

Barbara Chapman
Dept. of Computer Science
University of Houston
Houston, Texas
chapman@cs.uh.edu

***Abstract*—A dramatic improvement in energy efficiency is mandatory for sustainable supercomputing and has been identified as a major challenge. Affordable energy solution continues to be of great concern in the development of the next generation of supercomputers. Low power processors, dynamic control of processor frequency and heterogeneous systems are being proposed to mitigate energy costs. However, the entire software stack must be re-examined with respect to its ability to improve efficiency in terms of energy as well as performance. In order to address this need, a better understanding of the energy behavior of applications is essential. In this paper we explore the energy efficiency of some common kernels used in high performance computing on a multi-GPU platform, and compare our results with multicore CPUs. We implement these kernels using optimized libraries like FFTW, CUBLAS and MKL. Our experiments demonstrate a relationship between energy consumption and computation-communication factors of certain application kernels. In general, we observe that the correlation of energy consumption to GPU global memory accesses is 0.73 and power consumption to operations per unit time is 0.84, signifying a strong positive relationship between them. We believe that our results will assist the HPC community in understanding the power/energy behavior of scientific kernels on multi-GPU platforms.**

***Index Terms*—Power, Energy, Energy Efficiency, Multi-GPU, Scientific Kernels, High Performance Computing.**

I. Introduction

Energy consumption is a growing concern for HPC systems, despite substantial improvements in performance in the recent years. For example, Tianhe-1A supercomputer (formerly #1 in the Top500 list [8]) consumes around 4MW of electricity with a performance per watt of 0.785 GFlops/Watt [38], which is quite far from the ideal target. To economize next generation of supercomputers (an exascale computer is supposed to yield 10^{18} floating point operations per second), an energy efficiency metric of approximately 18 Gflops/Watt would be necessary to cap the power consumption at 60MW [4] (this could normally power up 20,000 households). To mitigate energy consumption, presently, off-chip hardware accelerators like GPUs are explored since they are capable of achieving substantial throughput along thousands of parallel threads. As such, computer vendors have begun to manufacture multi-GPU nodes, i.e. 4-8 GPUs in a single cluster node. Consequently, increasing the computing potential of a node has also increased the power consumption significantly.

High Performance GPUs consume more than twice the power as compared to multicore CPUs. Hence, there is an onus on software optimization to bridge the gap in power by improving energy efficiency. We believe that the energy analysis of application kernels on heterogeneous high performance systems could lead to a better understanding of how the execution environment can be exploited for the overall energy consumption factor. Both coarse (system level, per node, rack) and fine grained (individual component level, e.g, CPU core, memory, bus) power measurements are indispensable if we need to gain deeper understanding of the total energy consumption of a system.

This paper summarizes energy characteristics of four common HPC application kernels based on the `seven dwarfs` [1] classification: Matrix-Matrix Multiplication (DGEMM), Fast Fourier Transform (FFT), Pseudo-Random Number Generator (PRNG) and 3-D Finite Difference (3D FD) Stencils. These kernels are implemented on multicore CPUs and multiple GPUs using optimized packages like FFTW [9], Intel MKL and CUDA BLAS (CUBLAS).

We observed that regardless of different types of communication and computation patterns found in each of these kernels, some common parameters such as the number of global memory accesses plays a significant role in determining the overall energy consumption, as well as the parameter operations per unit time that determines the power consumption factor.

This paper is organized as follows. In Sections II we discuss relevant trends in hardware and their energy and power consumption. In Section III, we explain the evaluation platform in detail. In Section IV, we introduce the application kernels that we have used and analyze the results obtained. Section V draws some inferences about this study. In Section VI we discuss some of the previous works in this area and distinguish our approach from the rest. Finally, we present our conclusions and future work in Section VII.

II. Background

Heterogeneous platforms are currently receiving great attention in the HPC community since they are known to provide high performance and parallel processing capabilities for certain types of applications with a potentially favorable performance to power ratio. Performance per watt or Energy

CFP1225P-ART/12 $26.00 © 2012 IEEE
DOI 10.1109/SAAHPC.2012.17

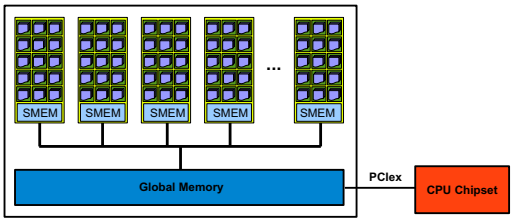

Fig. 1. *High level structure of a GPU* showing Streaming Multiprocessors (SM) and Streaming Processors (SP) within them.

Efficiency is emerging as an essential parameter to calculate the overall cost of an application. Since different applications exhibit different execution behavior, a classification of applications into different groups could highlight power/energy trends for a particular application class.

A. Power and Energy

Power is defined as the rate at which (Electrical) Energy is transferred by a circuit and is measured in Watts. According to *Joule's Law*:

$$P = \frac{W}{t} = \frac{qV}{t} = \frac{q}{t} * V = IV \tag{1}$$

Where W, t, q, V and I are work done, unit time, electrical charge, potential and current respectively. Energy could be defined as the flow of power along a conductor to do work, and it is measured in Joules.

$$\text{Energy } = \text{Power } X \text{ Elapsed time} \tag{2}$$

Energy efficiency is expressed by performance of an application per watt drawn.

$$\text{Energy Efficiency} = \frac{\text{Throughput (Operations Per Unit Time)}}{\text{Power (watts)}} \tag{3}$$

B. Energy Characteristics of GPUs

Complex applications are often composed of different sub-problems that might benefit from a hardware specifically suited to the execution behavior of such a sub-problem. This is the reason homogeneous cores may not suffice to meet certain application requirements. In the near future, heterogeneous parallel machines with nodes composed of various accelerators or specialized processors such as DSPs, GPUs, FPGAs, or on-chip vector processors, are expected to play a massive role in designing some of the largest systems in the world. As such, general purpose GPUs (GPGPUs) are slowly becoming ubiquitous devices in high performance computing.

Fig.1 shows a simplified GPU structure comprising of a number of streaming multi-processors, each of which has a number of simple vector units with private and shared memories. The main bottleneck with GPU is memory access latency, because the global memory that could be accessed by all processing elements has an overhead of more than 400 cycles [34]. Due to this reason, the applications that are ported to GPUs need to ensure that the best practices have been followed in order to reduce the latency by scheduling a large number of threads utilizing the shared memory and minimizing the global memory traffic.

TABLE I
ENERGY COST OF OPERATIONS AND INSTRUCTIONS W.R.T DOUBLE PRECISION FLOATING POINT OPERATIONS [36]

Operation	Energy (pJ)	Multiples of DP FLOPs	Instructions
I$ Fetch	33	0.67	2
Register Access (3W)	10.5	0.2	0.6
Access 3 D$	100	2	6
Access 3 L2 D$	460	9	27
Access 3 off chip	762	15	45
Access 3 from DRAM	6000	120	360

Table I refers to the microscopic energy usage and instruction counts in a device. It is interesting to notice that the energy cost for accessing 3 variables from global memory is 120 times as compared to a double-precision FLOP, suggesting that data movement governs the overall energy consumption for GPUs [36]. To get optimal performance from multiple GPUs, careful consideration about the data size is important, because the choice of moving computation to N devices could be justified only when the substantial speedup in computation could hide the associated latencies of data transfers to N devices.

C. Power Measurement Techniques

Broadly, there are two ways to measure power of a computing system, and they could be used together to enable deeper analysis - (1) Using sensors or hardware performance counters to read power values of specific areas (for e.g. memory, processor cores). Power of certain system architectures can be measured using hardware counters [12] that are capable of returning power consumed in watts by a single core, or any power plane (a mixture of registers, memory busses and processors) [28]. (2) An external probe (like a power analyzer) that monitors power consumed by the entire node. For a thorough power analysis of nodes, both these approaches are equally important.

III. Evaluation Platform

In this section we discuss the hardware configuration and software tools used for performing the experiments.

A. Hardware Configuration

The testbed node consists of an AMD Magny-Cours Processor - 6174 having a total of 24 physical cores. The node has 4 Nvidia Tesla M2050 GPUs, each with 448 compute cores. The CPU processor cores are clocked at 2.2 Ghz, and the graphics unit have persistent mode on, that enables the device driver to be resident on the device all the time (to minimize GPU initiation time). Fig.2 shows the layout of the node, with the Yokogawa WT500 power analyzer.

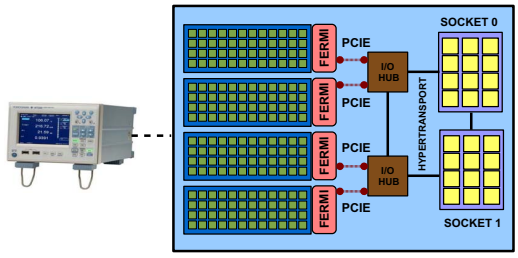

Fig. 2. *Testbed CPU-GPU node layout* with 4 GPUs (Nvidia Tesla M2050) and 24 CPU cores (2-way AMD Opteron Magny-Cours having 24 physical cores) with Yokogawa WT500 Power Analyzer

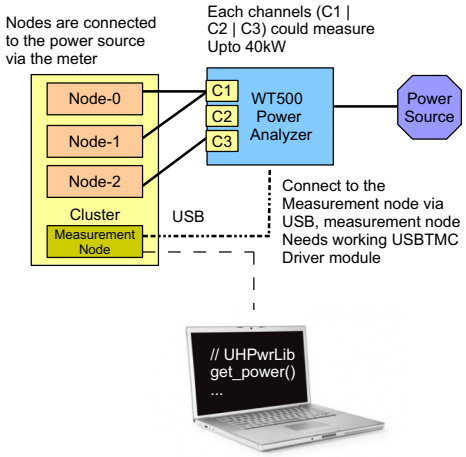

Fig. 3. *Layout of the power meter* with respect to the test node

B. Power Analysis

We have used a Yokogawa WT500 power meter, for reading the dynamic power consumed by the evaluation node. The Yokogawa WT500 is also endorsed as an accepted device by SPEC [10] for evaluating the power and performance characteristics of different classes of computers. The meter has been calibrated following the NIST regulations [29], conforming to highest standards. It is capable of producing samples every 100 ms, which makes it possible for performing fine-grained application analysis. The power range of the meter is 0-40 KW in a single channel. Fig.3 shows the layout of the meter with respect to the test nodes. Multiple nodes could connect to the meter via the same channel (as long as total power consumption is less than 40kW), or they may use separate channels (a total of 3 are available) in the device. The node under test is found to have an idle power of 320 watts. A thread-safe API was developed to remotely interface with the power analyzer. The API was found to have a negligible execution overhead. For a thorough analysis, it is required to sample power at frequent operating intervals of an application.

TABLE II
SOFTWARE TOOLS USED FOR THIS PAPER

Software Tools	CPU	GPU
Numerical Libraries	FFTW, Intel MKL	CUFFT, CUBLAS
Programming Models	OpenMP 3.1	
Compilers	Intel ICC	Nvidia CUDA Compiler 4.1 (LLVM based)
Profilers		CUDA Visual Profiler
Compiler Switches	-O3 -xHOST -ipo -ansi-alias -openmp	-fno-strict-aliasing -use_fast_math
Other Packages	Pthreads API	R Statistical Programming Language

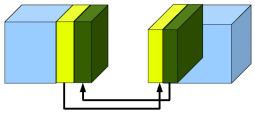

Fig. 4. *Ghost cell* transfers between two adjacent devices

C. Software Tools

Table II lists the software tools used for the experiments in this paper. In all of the GPU related experiments, memory transfer times were also included along with kernel execution time in performance evaluation calculations. Since, instantaneous power reading might be imprecise to judge the average power consumption of an application kernel, all the test cases are executed for many iterations to ensure that it reaches steady state power level. Intel ICC compiler is used for all the CPU case studies. The R statistical programming language is used to conduct correlation tests on GPU performance profiles. CUDA Visual Profiler is used to collect performance profiles of the GPU kernels.

IV. CASE STUDIES

In this section the energy profiles of a number of kernels used in high performance computing have been explored and analyzed. Each of these kernels demonstrate a unique computation-communication pattern. This analysis relies heavily on the actual experiments performed on multiple CPU cores and multiple GPUs. Four kernels - DGEMM, FFT, PRNG and 3D FD have been chosen from the *seven dwarf* categories of Dense Linear Algebra, Spectral Methods, Structured Grids and Monte Carlo Methods respectively for the experimental purposes in this study.

A. Dense Linear Algebra: Double Precision Matrix-Matrix Multiply

The BLAS specification [3] defines level-3 DGEMM as one of the following operation to perform matrix-matrix multiplication.

$$C \leftarrow \alpha AB + \beta C, C \leftarrow \alpha A^T B + \beta C,$$
$$C \leftarrow \alpha AB^T + \beta C, C \leftarrow \alpha A^T B^T + \beta C \quad (4)$$

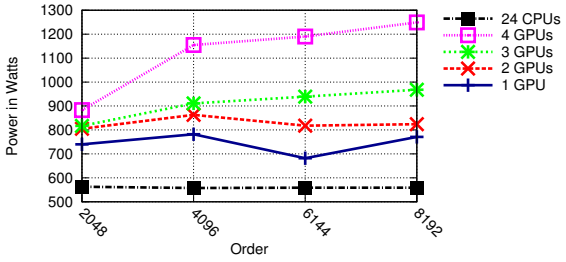

Fig. 5. *DGEMM* Power Consumption

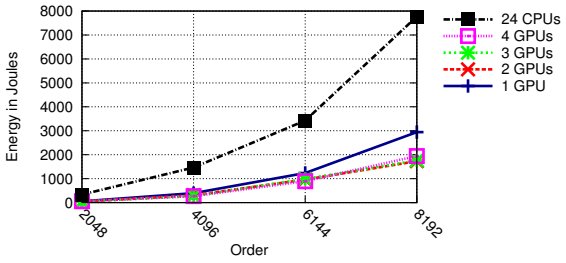

Fig. 6. *DGEMM* Energy Consumption

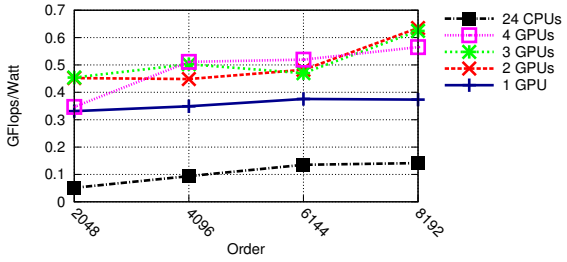

Fig. 7. *DGEMM* Energy Efficiency

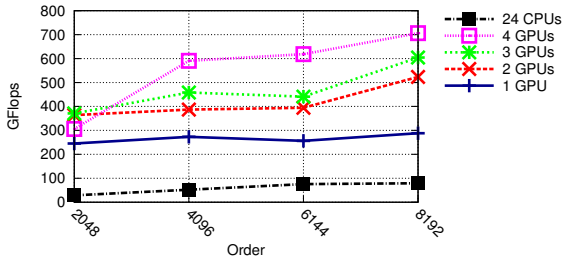

Fig. 8. *DGEMM* Performance

Matrix Multiplication is a costly operation in terms of complexity (O(n^3)) if performed naively. We have used Intel MKL with Pthreads API for multicore CPUs and CUBLAS v2.0 for GPUs. CUBLAS is the Nvidia runtime implementation of BLAS (Basic Linear Algebra Subprograms).

Fig.5 shows the power and energy consumption of DGEMM under varying data sizes. The figure suggests that the power consumption of 4 GPUs is approximately twice to that of 24 CPU cores (according to the vendor manuals average CPU power is around 80W, and for the GPU it is 225W). The experimental analysis for the input data considered, suggests that it is economical to use only 3 GPUs, instead of using all the available GPUs. Fig.6 shows that the energy consumed by the multiple GPUs is lesser by 4X as compared to the multicore CPUs.

Fig.7 reflects that the energy efficiency of 2 to 3 GPUs is slightly more than using all the 4 GPUs together. Although, the power consumption of 4 GPUs was found to be approximately twice to that of 24 CPU cores, the energy efficiency of DGEMM on multi-GPUs is more than 6 times to that of multicore CPUs. Fig.8 shows that the performance increases with the data sizes using multiple GPUs at a much faster rate than CPUs. For the CPUs, energy efficiency degrades as the data size is increased. The results suggest that DGEMM benefits from multiple GPUs.

In some applications (for e.g. image processing), we might need to perform many independent matrix-matrix multiplications concurrently on small matrices. independent of each other. The current version of CUBLAS supports batched execution. For DGEMM, a large number of batches (of the order of thousands) should be scheduled with smaller matrices in order to minimize the communication overhead to obtain a substantial speedup.

Fig.9 and Fig.10 shows the power and energy consumption of batched DGEMM using small to medium sized matrices on GPUs. In the batched case, each GPU is executing DGEMM on the matrices concurrently in 1000 batches (using GPU streams) and there is no domain decomposition to N devices.

The GPU hardware can only execute 16 streams at a given time, hence even if 1000 batches are scheduled, only 16 will be executed concurrently at the given time. The peak power of DGEMM was found to be approximately 1300 watts (Fig.5), slightly more than the maximum power capacity of the compute node, according to the vendor specifications. DGEMM is compute bound, and with large amount of data access instructions it is also the most power consuming application on multiple GPUs amongst our test cases. Using batches of small matrices could be an energy efficient solution.

B. Spectral Methods: Fast Fourier Transforms

Fourier transform is a well known and widely used algorithm in many scientific and engineering fields. Fourier analysis involves conversion of one function from spatial (or time) to frequency domain, since it is faster to perform some operations in the frequency domain.

$$F(x,y) = \sum_{m=0}^{M-1}\sum_{n=0}^{N-1} f(m,n)e^{-j2\pi(x\frac{m}{M}+y\frac{n}{N})}$$
$$f(m,n) = \frac{1}{MN}\sum_{m=0}^{M-1}\sum_{n=0}^{N-1} F(x,y)e^{j2\pi(x\frac{m}{M}+y\frac{n}{N})} \quad (5)$$

As evident from equation 5, fourier transform is separable, and hence computation could be performed independently in

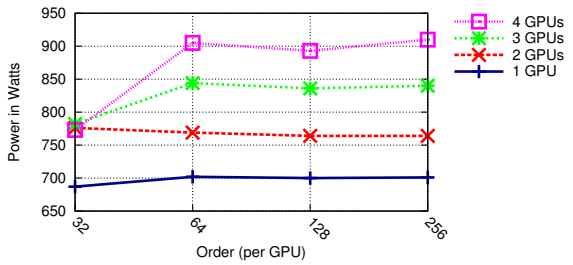

Fig. 9. *Batched DGEMM* Power Consumption on GPUs for 1000 concurrent batches

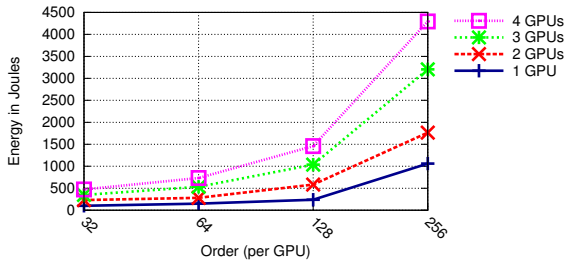

Fig. 10. *Batched DGEMM* Energy Consumption on GPUs for 1000 concurrent batches

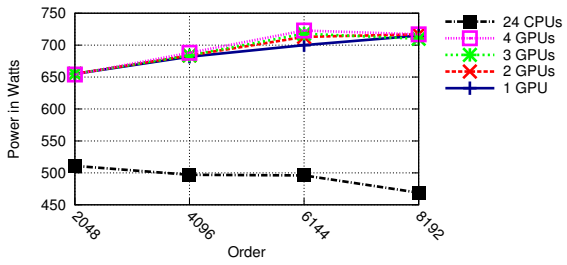

Fig. 11. *FFT* Power Consumption

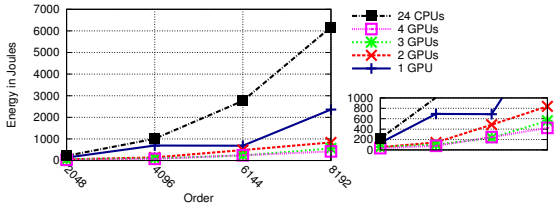

Fig. 12. *FFT* Energy Consumption

D-dimensions. We have experimented 2-D FFTs, using CUDA FFT (CUFFT) libraries in GPUs and FFTW library [9] using OpenMP on multicore CPUs running OpenMP threads [6]. The operation of 2-D FFT involves 1-D FFTs on each row and column of the input array, with a transpose operation to ensure column elements are in contiguous locations. We used multiple GPUs to perform the transpose operation and inferred that it requires frequent data transfers between GPUs, thereby degrading the performance. In order to address this issue, we perform an in-place transpose on the host side, this will help in considerably reducing the execution time on the GPUs. We do not include the timings for the in-place transpose in the total GPU FFT timing measurements.

A single FFT could be expressed as a combination of many small FFTs i.e. we use CUFFT for the GPU implementation and this performs FFTs in concurrent batches (rather than one large FFT execution). For FFT, the GPU computation is significantly less compared to the overall computation (the transpose is performed at the host side), and hence the power consumption of all the GPUs appears to be the same and approximately 1.2 times to that of 24 CPU cores as shown in Fig.11.

In Fig.12, it is shown that the energy consumed by the GPUs for the different data sizes is considerably low.

As a result, it is evident that the energy efficiency for 4 GPUs as shown in Fig.13 was found to be approximately 30 times higher to that of multicore CPUs that is using FFTW with OpenMP. Fig.13 also shows that for larger data size (greater than order of 6144), the latency of global memory affects the execution time. The energy efficiency between multicore CPUs and GPUs are observed to diverge upon increasing the number of devices. While using single GPU, the performance per watt of GPU is approximately 6 times to that of CPUs, but it increases to approximately 40 times when 4 GPUs were used in the computation.

Fig.14 shows that for a single GPU, the performance of FFT was around 10 times to that of multicore CPUs, and once all the 4 GPUs are being used, the speedup was around 50 times to that of the CPU implementation. We see that although CUFFT can handle non-power of 2 data sizes, there is still performance deviation of data size which is a power of 2 to that of a non-power of 2 (notice the performance difference between matrix order 4096 to 6144, which is a non-power of 2).

Although FFT is not very compute intensive (order of $5NlogN$), we observe that it achieves better performance while porting to multiple GPUs.

C. Monte Carlo Methods: Pseudo Random Number Generators

Pseudo random numbers are used extensively in application fields involving statistics. The Mersenne Twister algorithm is considered to be one of the best available PRNGs that guarantees a long repetition-free sequence performance [23]. The Mersenne Twister GPU implementation and optimization is discussed in [31], and forms a basis for this experiment. `GNumbers` is the amount of random numbers generated per nanosecond. We have used GNumbers to represent throughput of Mersenne Twister in our figures.

For the CPU version of the program, the C-language implementation of Mersenne Twister was adopted from the original authors website [23]. A common approach to make the algorithm parallel is to individual twisters or streams of random numbers, thereby making this operation embarrassingly parallel. However, Mersenne Twister relies on a set of parameters, that apart from the initial seed needs to be different

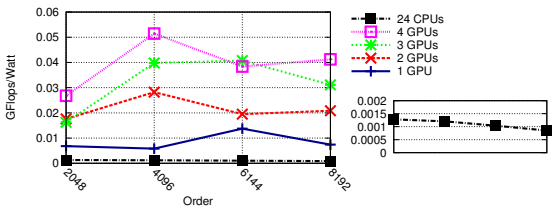

Fig. 13. *FFT* Energy Efficiency

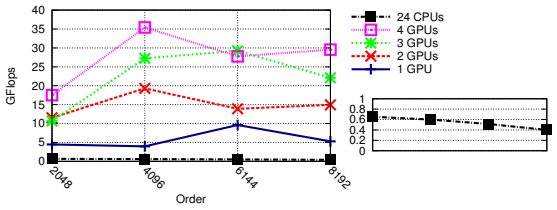

Fig. 14. *FFT* Performance

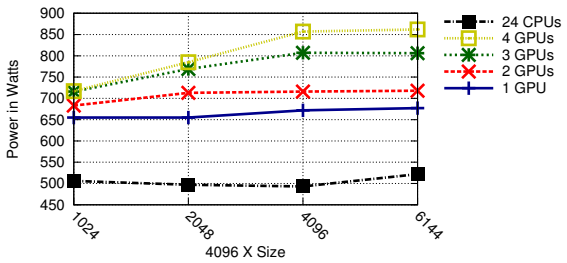

Fig. 15. *Mersenne Twister* Power Consumption

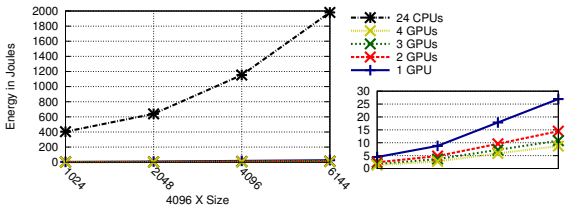

Fig. 16. *Mersenne Twister* Energy Consumption

for each thread to create a set of diverging random numbers. For creation of individual *twister* (i.e random number generator) parameters at runtime for MPI like parallel processing, the authors of Mersenne Twister created another library, called `dcmt` [22] (Dynamic Creation Mersenne Twister). This library is used to generate twister configurations, and then each of the twisters are given a different seed. This ensures a parallel Mersenne Twister PRNG, and these steps were performed before we computed random number generation using mersenne twister on multicore CPUs and multiple GPUs. However, this parameter generation process is extremely time consuming, but this is a one time effort only. Input parameters could be shared by all the threads, minimizing DRAM accesses and leaving only bit-wise arithmetic computations. The low arithmetic and data intensity reflects on the power and energy values for the GPUs, as evident from the results in Fig.15 and Fig.16. The maximum energy consumption using 1 GPU was observed to be 25 Joules, whereas 4 GPUs consumed around 10 Joules - an improvement of 200X as compared to multicore CPUs.

As shown in Fig.17, the multi-GPU implementations of Mersenne Twister are many factors greater than that of the parallel CPU implementation. Fig.18 shows that single GPU performance is around 80 times more than the multicore CPU throughput, whereas, 4 GPUs gave a speedup of approximately 300 times to that of multicore CPUs. The algorithm has relatively low computation overhead, which makes it an ideal application to parallelize using a large number of threads, and also there is no host to device transfer, that contributes towards the efficiency of multi-GPU workloads.

The results indicate that a PRNG is an ideal application for a massively parallel system like GPUs.

D. Structured Grids: Finite Difference using Stencils

Finite difference stencils are used to solve partial differential equations (PDEs). A given point in space is operated with the weighted contribution from a subset of neighbors around it. It can also be summarized as a triply nested loop updating an array based on the values of its neighbor. It forms the basis for *Reverse Time Migration* algorithm, that is one of the fundamental algorithms for seismic computation [18]. A 3D stencil is applied on a volume (array `v` in equation 6) having constant coefficients (denoted by `c` in equation 6).

$$
\begin{aligned}
u_{i,j,k} = c_{0,i,j,k} * v_{i,j,k} + \sum_{l=1}^{O/2} \\
(c_{l,i-l,j,k} * v_{i-l,j,k} + c_{l,i+l,j,k} * v_{i+l,j,k} \\
+ c_{l,i,j+l,k} * v_{i,j+l,k} + c_{l,i,j-l,k} * v_{i,j-l,k} \\
+ c_{l,i,j,k+l} * v_{i,j,k+l} + c_{l,i,j,k-l} * v_{i,j,k-l})
\end{aligned}
\tag{6}
$$

This stencil is referred to as a `(3*O+1)`-point stencil in space and second order in time. In our experiments, we have discussed an order-8, 25 point (in space) isotropic stencil (3-dimensional) and second order in time (total 27 points). The GPU implementation is based on the kernel described in [24], that was modified to support multiple GPUs. Shared memory utilization is encouraged for GPUs to reuse data between threads that access common data. With multi-GPU implementations, data needs to be exchanged in the border regions, which doubles when every GPU is added (as shown in Fig.4 for two devices). Since stencil-based computations are memory bound, multi-GPU computation would be beneficial only when the data slices on each GPU would take more time to compute than the time taken for the data transfer operation itself. In our case, since we only consider a single node, we were limited by the amount of memory available.

Fig.19 shows that there is little change in power when different number of GPUs are used, which is due to lower computation overheads per operation for the input data sizes. Fig.20 shows that energy consumption shows similar behavior to power consumption. The energy efficiency of the kernel for

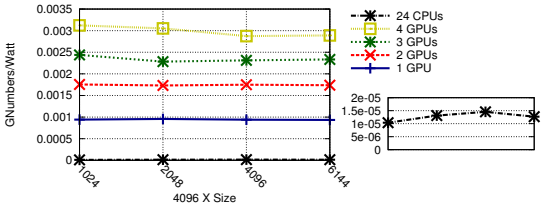

Fig. 17. *Mersenne Twister* Energy Efficiency

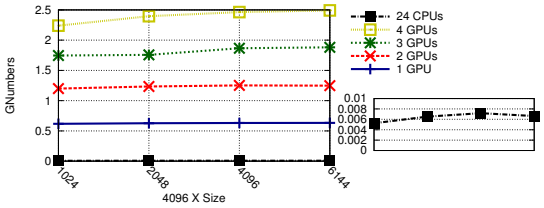

Fig. 18. *Mersenne Twister* Performance

multiple CPUs as shown in Fig.21 was found to be slightly more than that of the GPUs (around 1.5x) for the input data sizes. This suggests that that for stencil computation, domain decomposition to multiple GPU is viable, only when the amount of data is large enough to have a significant impact on computation.

Since stencil computations are memory intensive, GFlops loses its relevance as a performance metric. For this reason, a more appropriate measure that relates to the performance is the number of points operated on per nanosecond (called Giga-points). Hence, in all the figures GPoints is used as an evaluation criteria.

As our data sets were not large enough, we only observe a nominal speedup in GPU implementations as compared to multicore CPUs as shown in Fig.22.

In general, since the input data is many times the size of the caches, a blocking technique is employed for the CPU computation to ensure that the accessed data points could be pre-fetched into the cache. For the multicore CPU version OpenMP parallel constructs are used for parallel computation on 24 CPU cores.

V. Relation between Performance Counters and Energy

In the previous section, we studied the power/energy characteristics of some of the application kernels from four dwarf categories, and observed that GPUs are quite energy efficient as compared to CPUs as long as the data transfer latency could be hidden by computation.

In this section we have collected hardware performance counters results applied to application kernels and established a correlation with the energy consumption. These results indicate if the kernels are limited by computation or memory bandwidth and help in identifying further scope of optimizations. For hardware accelerators like GPUs, the computation to data transfer time determines the performance of an application.

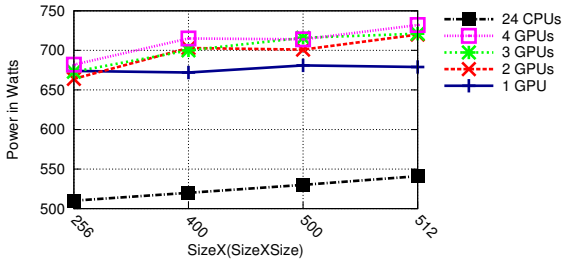

Fig. 19. *3D FD stencil* Power Consumption

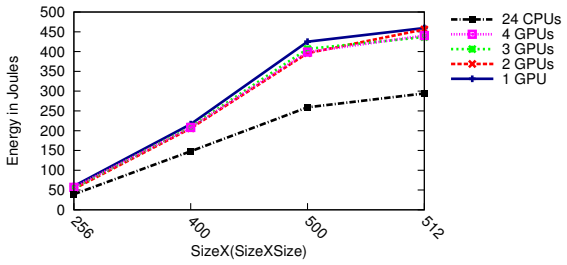

Fig. 20. *3D FD stencil* Energy Consumption

Table III shows GPU instructions issued, hardware performance counter results, % of kernel execution and memory copy times along with corresponding power and energy consumption of the test kernels on 2 GPUs. Assessing the performance limiters is a vital step in trying to understand potential areas of optimizations. Instruction and memory throughput of a device governs the performance of an application [30]. The theoretical instructions per byte of memory transfer of the GPU (Tesla M2050) in our testbed is calculated as - $\frac{\text{Theoretical Instruction Throughput}}{\text{Theoretical Memory Throughput}}$, which for Tesla M2050 approximately is $\frac{515.2}{148.4} = 3.47$. We compare the instructions per byte metric of each of the kernels to the ideal instructions per byte (3.47), to infer if the kernel is arithmetic or memory bound. A value lesser than the ideal is considered to be a behavior of memory bound applications, whereas values greater than the ideal metric would suggest that the kernels are compute intensive (arithmetic bound).

The Instructions/Byte ratio in TableIII is calculated as:

```
(number of SMs) * 32 * instructions_issued
:  32B * (dram_reads + dram_writes)
```

The number of streaming multiprocessors (SM) in Tesla M2050 is 14. The `dram_reads` and `dram_writes` counter values (obtained from CUDA Visual Profiler) are incremented once in every 32 byte access, and instructions issued is incremented once per warp (group of 32 threads).

In Section II-B, we had discussed that the global memory traffic of GPU is one of the primary reasons for the increase in the overall energy consumption. Table III shows that the `dram_reads` counter value for DGEMM is the largest amongst all the other kernels. Correspondingly, DGEMM was the only application kernel reaching the peak power of the test

TABLE III
PERFORMANCE PROFILE OF KERNELS FROM CUDA PROFILER USING TWO GPUS

GPU Kernel	Allocated Data Size	Instructions Issued	DRAM Reads	DRAM Writes	% Kernel	% Memory Copy	Power (Watts)	Energy (Joules)	Instructions / Byte Per GPU
DGEMM	1.67 GB	1980502000	6588490000	20285300	86.37	6.6	805	1731	4.20
FD	4.22 GB	44000000	169837900	45591900	6.1	67	720	456.174	2.86
FFT	2 GB	38842033	178646600	166991500	0.24	6.1	717	423	1.57
MT	128 KB	7476530	153383	14736920	18.29	64.5	718	14.477034	7.03

TABLE IV
CORRELATION MATRIX OF GPU PERFORMANCE COUNTERS WITH POWER/ENERGY

Correlation	Instructions Issued	DRAM Reads	DRAM Writes	Energy	Operations Per Unit Time	Power
Instructions Issued	1	0.98381594	-0.1018084	0.777515	0.73547236	0.5684821
DRAM Reads	0.98381594	1	-0.08339367	**0.73400324**	0.74639634	0.5749922
DRAM Writes	-0.1018084	-0.08339367	1	0.3196255	-0.2268728	-0.1265411
Energy	0.77751496	**0.73400324**	0.31962547	1	0.4544755	0.3413445
Operations Per Unit Time	0.73547236	0.74639634	-0.22687275	0.4544755	1	**0.8374266**
Power	0.56848213	0.57499217	-0.12654108	0.3413445	**0.8374266**	1

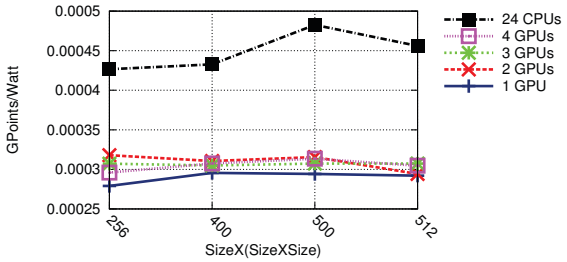

Fig. 21. *3D FD stencil* Energy Efficiency

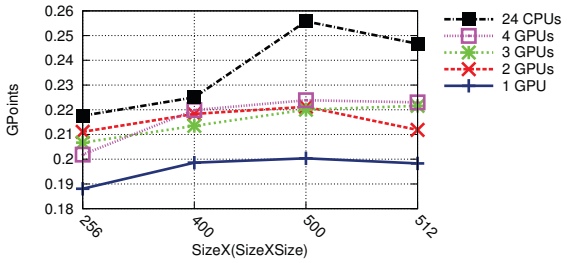

Fig. 22. *3D FD stencil* Performance

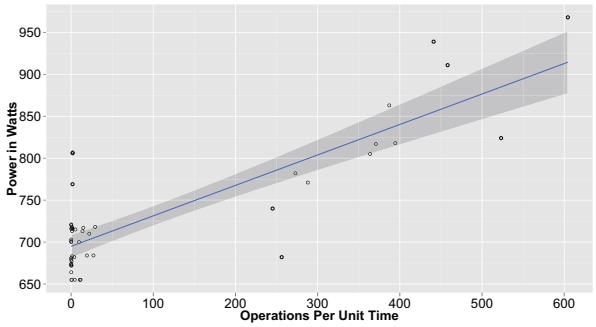

Fig. 23. Scatter plot of Power and Operations per Unit Time, r = 0.837

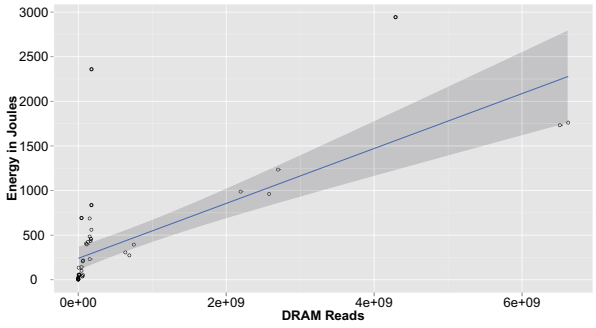

Fig. 24. Scatter plot of Energy and DRAM Reads, r = 0.734

node (refer to Fig.5).

Batched FFT computation across multiple GPUs were observed to have a limited impact on power (Fig.11), and the high `dram_reads` value and low `Instructions/Byte` signifies the application to be memory bound. On the other hand, despite the highest instruction per byte metric, the mersenne twister kernel shows the lowest energy consumption. The reason is that the `dram_reads` counter value (since no data transfer occurs from host to device) is the least amongst all the test kernels.

The performance of FD on multiple GPUs depends on the input data size. Each GPU should have sufficient data to compute on, so that the data transfer time is insignificant compared to computation time. Largest input data transfer was for 3D FD (around 4 GB), but since there is more data reuse, global memory accesses were minimized. Hence, we observe limited variation in power with respect to using a number of GPUs as shown in Fig.19.

A. Correlation Tests

Table IV lists the correlation of the GPU hardware performance counters with power and energy consumption. A correlation value closer to 1 indicates a strong relationship between two variables. We observe a very strong relationship between energy consumption of the kernels to the amount of global memory accesses (a correlation coefficient of 0.734) and total instructions issued (corrleation coefficient of 0.746). On the other hand, a strong correlation of power to that of operations per unit time (expressed in GFlops, GNumbers and GPoints in the experiments) is also evident (correlation coefficient of 0.837). Through regression analysis, we find a linear association between power & operations per unit time and Energy & DRAM reads, this could be used in predicting the values of power and energy for a given application code. Fig. 23 and Fig. 24 show a strong association between power & operations per unit time and energy & DRAM reads respectively. The linear regression lines in Fig.23 and Fig.24 serve as a base model to analyze the behavior of power and energy with variations in DRAM Reads and operations per unit time.

Based on what has been discussed so far, we see that different kernels exhibit different communication/computation patterns resulting in varied power/energy behavior. Our results indicate that we are also able to establish a relationship between the energy consumed by the overall system and the global memory access overheads of an application kernel. A similar observation was also made with respect to power consumption of the system and the total number of operations in a specific kernel, per unit time.

VI. Related Work

Using DVFS (Dynamic Voltage Frequency Scaling), the per core processor frequency could be modulated by reducing their operation frequency linearly as discussed in [21], [35], [37]. Contrary to the popular belief that lower frequency operating points would lead to greater energy efficiency, under-clocking processors may lead to decreased energy efficiency as well [25]. Hence, without thorough application investigation, DVFS might adversely affect the performance. There are several efforts to analyze power values, one of them is using hardware performance counters.

Data on power consumption at the system runtime can be collected by using *special purpose* hardware counters [7], [14], but this approach is presently limited to certain architectures. Other studies such as [5], [20], [27], [33] show estimation of power using *existing* hardware counters and establish a relationship between system power and certain hardware counters. For examples, authors in [27] have proposed a simple linear regression model to estimate power consumption of GPU kernels based on hardware performance counters (for a single GPU). A design for an energy efficient GPU cluster was proposed in [32] where the authors have analyzed the power/energy characteristics of multi-GPU DGEMM [2] for running Linpack on AMD GPUs.

Some of the general procedures/optimizations (for e.g. loop unrolling) are adopted to improve performance/power relationship and energy efficiency for a given application as discussed in [17]. Previous efforts like [16], [17], [26] list some tools (like PowerTracer, SPEC PTDaemon and Intel Energy Checker SDK [10], [13], [15]) and best practices that could be used to measure power in high performance computing systems. An analysis of GPU GEM package using properties of CUDA on GPU to evaluate the GEM implementation using metrics such as performance and energy efficiency is discussed in [11]. In [19], the authors have proposed a power management architecture for high performance computing systems, and have conducted experiments on Tianhe-1A supercomputer. Their results on implementing the proposed power provision and capping policies indicate power reduction by 10% with an overall performance degradation of only 2%.

While most of the existing work deals with power management and effects of optimization on system power on a single GPU system or only CPUs, we have focused on profiling energy efficiency of some application categories in a multi-GPU platform. Since all the experiments have been performed on multicore CPUs as well, it is possible to directly compare the CPU-GPU performance and energy efficiency. We have considered application kernels that exhibit varied communication-computation requirements and analyzed the relationship between performance profiles and energy efficiency. The collected information could be used in assessing the efficacy of porting an application to CPU+GPU systems with respect to power/energy consumption. Power consumption of HPC systems depend on the resource (memory, CPU, GPU, storage, networking, cooling) utilization over time. Since application kernels are typically memory bound, compute intensive or I/O bound, the kernels differ in their use of the available components. So, there is a scope of energy conservation for underutilized components if the right kind of analysis is performed. Extensive power measurements of different application kernels is a must to evaluate the effectiveness of a power management scheme.

VII. Conclusion and Future Work

In this paper, we study the energy profiles of some common HPC kernels. Multi-GPU systems provide great opportunities in improving performance of compute-intensive kernels considerably, compared to multicore CPUs (as we see in Fig.7). However, applications with significant overheads in data transfer due to domain decomposition could only notice marginal performance increase. Hence, the computation to communication ratio per device is a key factor that contributes to the overall efficacy of deploying applications in a multi-GPU environment. Our experiments show that there is not much variation in the power consumption across the test kernels, but it is energy which is the key differentiator (mainly, because the kernels have different execution times). Power is definitely an important component, but it only provides a general notion about the effectiveness of a hardware platform. To have a clearer idea about the productivity of an application

kernel on a hardware, the energy efficiency factor needs careful consideration. In order to establish a relationship between the performance of an application kernel and its energy usage, we have analyzed the hardware performance counters and observe a strong correlation between GPU global memory accesses & energy consumption and power & operations per unit time.

The analysis of case studies could assist in modeling power/energy behavior of a wide spectrum of application kernels on multi-GPU platforms. To facilitate a thorough understanding of application kernel performance and energy efficiency, more case studies are required to be looked at, and we are in the process of considering test kernels from rest of the dwarf categories. The API to interface with the power analyzer could be improved to extract thermal characteristics of the system, to assist in deeper analysis.

VIII. Acknowledgment

Development at the University of Houston was supported in part by the NSFs Computer Systems Research program under Award No. CRI-0958464. We would also like to acknowledge NSF CCF-0917285. This work is inspired from experiences of the first author during summer 2011, when he was an intern at PNNL, and wishes to thank Dr. Darren Kerbyson, Dr. Abhinav Vishnu and Dr. Kevin Barker for providing guidance. We wish to thank our colleagues, Dr. Yonghong Yan and Priyanka Ghosh, for reviewing this paper and suggesting corrections.

References

[1] K. Asanovic, R. Bodik, B.C. Catanzaro, J.J. Gebis, P. Husbands, K. Keutzer, D.A. Patterson, W.L. Plishker, J. Shalf, S.W. Williams, et al. The landscape of parallel computing research: A view from berkeley. Technical report, Technical Report UCB/EECS-2006-183 EECS Department University of California, Berkeley, 2006.

[2] M. Bach, M. Kretz, V. Lindenstruth, and D. Rohr. Optimized hpl for amd gpu and multi-core cpu usage. *Computer Science-Research and Development*, pages 1–12, 2011.

[3] L.S. Blackford, A. Petitet, R. Pozo, K. Remington, R.C. Whaley, J. Demmel, J. Dongarra, I. Duff, S. Hammarling, G. Henry, et al. An updated set of basic linear algebra subprograms (blas). *ACM Transactions on Mathematical Software*, 28(2):135–151, 2002.

[4] S. Borkar. Major challenges to achieve exascale performance. In *Salishan Conference on High-Speed Computing*, 2009.

[5] V. Bui, B. Norris, K. Huck, L.C. McInnes, L. Li, O. Hernandez, and B. Chapman. A component infrastructure for performance and power modeling of parallel scientific applications. In *Proceedings of the 2008 compFrame/HPC-GECO workshop on Component based high performance*, page 6. ACM, 2008.

[6] B. Chapman, G. Jost, and R. Van Der Pas. *Using OpenMP: portable shared memory parallel programming*, volume 10. The MIT Press, 2007.

[7] G. Contreras and M. Martonosi. Power prediction for intel xscale® processors using performance monitoring unit events. In *Low Power Electronics and Design, 2005. ISLPED'05. Proceedings of the 2005 International Symposium on*, pages 221–226. IEEE, 2005.

[8] J.J. Dongarra, H.W. Meuer, and E. Strohmaier. Top500 supercomputer sites. *Supercomputer*, 13:89–111, 1997.

[9] M. Frigo and S.G. Johnson. Fftw: An adaptive software architecture for the fft. In *Acoustics, Speech and Signal Processing, 1998. Proceedings of the 1998 IEEE International Conference on*, volume 3, pages 1381–1384. IEEE, 1998.

[10] J.L. Henning. Spec cpu2000: Measuring cpu performance in the new millennium. *Computer*, 33(7):28–35, 2000.

[11] 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*, IPDPS '09, pages 1–8, Washington, DC, USA, 2009. IEEE Computer Society.

[12] Intel. Intel power gadget 2.0. http://software.intel.com/en-us/articles/intel-power-gadget/.

[13] Intel. Intel energy checker sdk. http://software.intel.com/en-us/articles/intel-energy-checker-sdk/, 2010.

[14] C. Isci and M. Martonosi. Runtime power monitoring in high-end processors: Methodology and empirical data. In *Proceedings of the 36th annual IEEE/ACM International Symposium on Microarchitecture*, page 93. IEEE Computer Society, 2003.

[15] J. Kunkel. Hdtrace–a tracing and simulation environment of application and system interaction.

[16] K.D. Lange. Identifying shades of green: The specpower benchmarks. *Computer*, 42(3):95–97, 2009.

[17] P. Larsson. Energy-efficient software guidelines. 2011.

[18] S.A. Levin. Principle of reverse-time migration. *Geophysics*, 49(5):581–583, 1984.

[19] Y. Liu, H. Zhu, K. Lu, and Y. Liu. A power provision and capping architecture for large scale systems.

[20] K. London, J. Dongarra, S. Moore, P. Mucci, K. Seymour, and T. Spencer. End-user tools for application performance analysis using hardware counters. In *International Conference on Parallel and Distributed Computing Systems*, pages 8–10, 2001.

[21] P. Macken, M. Degrauwe, M. Van Paemel, and H. Oguey. A voltage reduction technique for digital systems. In *Solid-State Circuits Conference, 1990. Digest of Technical Papers. 37th ISSCC., 1990 IEEE International*, pages 238–239. IEEE, 1990.

[22] M. Matsumoto and T. Nishimura. Dynamic creation of pseudorandom number generators. *Monte Carlo and Quasi-Monte Carlo Methods*, pages 56–69, 1998.

[23] M. Matsumoto and T. Nishimura. Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 8(1):3–30, 1998.

[24] P. Micikevicius. 3d finite difference computation on gpus using cuda. In *Proceedings of 2nd Workshop on General Purpose Processing on Graphics Processing Units*, pages 79–84. ACM, 2009.

[25] A. Miyoshi, C. Lefurgy, E. Van Hensbergen, R. Rajamony, and R. Rajkumar. Critical power slope: understanding the runtime effects of frequency scaling. In *Proceedings of the 16th international conference on Supercomputing*, pages 35–44. ACM, 2002.

[26] D. Molka, D. Hackenberg, R. Schöne, T. Minartz, and W.E. Nagel. Flexible workload generation for hpc cluster efficiency benchmarking. *Computer Science-Research and Development*, pages 1–9, 2011.

[27] H. Nagasaka, N. Maruyama, A. Nukada, T. Endo, and S. Matsuoka. Statistical power modeling of gpu kernels using performance counters. In *Proceedings of International Green Computing Conference*, 2010.

[28] A. Naveh, E. Rotem, A. Mendelson, S. Gochman, R. Chabukswar, K. Krishnan, and A. Kumar. Power and thermal management in the intel core duo processor. *Intel Technology Journal*, 10(2):109–122, 2006.

[29] National Institute of Standards. Nist. http://www.nist.gov/.

[30] Nvidia Paulius Micikevicius. Analysis driven optimization. http://www.nvidia.com/content/PDF/sc_2010/CUDA_Tutorial/SC10_Analysis_Driven_Optimization.pdf, 2010.

[31] V. Podlozhnyuk. Parallel mersenne twister. *NVIDIA white paper*, 2007.

[32] D. Rohr, M. Bach, M. Kretz, and V. Lindenstruth. Multi-gpu dgemm and hpl on highly energy efficient clusters. *Micro, IEEE*, (99):1–1, 2011.

[33] K. Singh, M. Bhadauria, and S.A. McKee. Real time power estimation and thread scheduling via performance counters. *ACM SIGARCH Computer Architecture News*, 37(2):46–55, 2009.

[34] V. Volkov and J.W. Demmel. Benchmarking gpus to tune dense linear algebra. In *High Performance Computing, Networking, Storage and Analysis, 2008. SC 2008. International Conference for*, pages 1–11. IEEE, 2008.

[35] M. Weiser, B. Welch, A. Demers, and S. Shenker. Scheduling for reduced cpu energy. *Mobile Computing*, pages 449–471, 1996.

[36] Stanford University William Dally, NVidia. Power efficient supercomputing. http://www.lbl.gov/cs/html/Manycore_Workshop09/.

[37] Q. Wu, P. Juang, M. Martonosi, L.S. Peh, and D.W. Clark. Formal control techniques for power-performance management. *Micro, IEEE*, 25(5):52–62, 2005.

[38] X.J. Yang, X.K. Liao, K. Lu, Q.F. Hu, J.Q. Song, and J.S. Su. The tianhe-1a supercomputer: its hardware and software. *Journal of Computer Science and Technology*, 26(3):344–351, 2011.