Effects of Dynamic Voltage and Frequency Scaling on a K20 GPU

Rong Ge, Ryan Vogt, Jahangir Majumder, and Arif Alam
Dept. of Mathematics, Statistics and Computer Science
Marquette University
Milwaukee, WI, USA
{*rong.ge, ryan.vogt*}*@marquette.edu*
{*akmjahangir.majumder, mohammadariful.alam*}*@marquette.edu*

Martin Burtscher and Ziliang Zong
Dept. of Computer Science
Texas State University
San Marcos, TX, USA
{*burtscher, ziliang*}*@txstate.edu*

***Abstract*—Improving energy efficiency is an ongoing challenge in HPC because of the ever-increasing need for performance coupled with power and economic constraints. Though GPU-accelerated heterogeneous computing systems are capable of delivering impressive performance, it is necessary to explore all available power-aware technologies to meet the inevitable energy efficiency challenge.**

In this paper, we experimentally study the impacts of DVFS on application performance and energy efficiency for GPU computing and compare them with those of DVFS for CPU computing. Based on a power-aware heterogeneous system that includes dual Intel Sandy Bridge CPUs and the latest Nvidia K20c Kepler GPU, the study provides numerous new insights, general trends and exceptions of DVFS for GPU computing. In general, the effects of DVFS on a GPU differ from those of DVFS on a CPU. For example, on a GPU running compute-bound high-performance and high-throughput workloads, the system performance and the power consumption are approximately proportional to the GPU frequency. Hence, with a permissible power limit, increasing the GPU frequency leads to better performance without incurring a noticeable increase in energy. This paper further provides detailed analytical explanations of the causes of the observed trends and exceptions.

The findings presented in this paper have the potential to impact future CPU and GPU architectures to achieve better energy efficiency and point out directions for designing effective DVFS schedulers for heterogeneous systems.

***Keywords*-DVFS in GPU Computing, Energy-Efficient Computing, Dynamic Voltage and Frequency Scaling**

I. Introduction

In recent years, GPU accelerated heterogeneous computing has become an important high performance computing technology. A recent survey of the top 500 supercomputers suggests that about 8% of them are heterogeneous systems based on General Purpose Graphics Processing Units (GPGPUs). In addition to typical high performance microprocessors, these systems also include a large number of GPGPUs, each of them consisting of thousands of simple and highly specialized processing cores running at a relatively low frequency. During computation, CPUs offload compute-intensive, highly parallelizable code segments to GPUs for execution acceleration.

By utilizing the aggregated computational capacity of $100\times$ more GPU cores, it has been shown that a hybrid HPC system with CPUs and GPUs working in tandem can improve both performance and energy efficiency. However, improving energy efficiency is an ongoing challenge in High-Performance Computing (HPC) because of the ever-increasing need for performance in combination with constraints in the power budget and economic cost. As a result, it is necessary to explore all available power-aware technologies in GPU-based heterogeneous systems.

Dynamic Voltage and Frequency Scaling (DVFS) is a power-saving technology whose aim is to lower a component's power state while still meeting the performance requirement of the running workload. DVFS has been widely employed in mobile devices, desktop computing, and HPC. Recently, high-end GPUs that support DVFS, such as Nvidia's K20c, also began shipping.

Previous studies have established that the DVFS technology provided by CPUs can improve energy efficiency without degrading performance in HPC. These studies share a common key idea: judiciously scaling down the CPU frequency in the presence of processor slack caused by intensive memory accesses, communications, disk I/O accesses or synchronization. However, it is largely unknown whether DVFS on GPUs can bring similar benefits to hybrid HPC platforms and it remains unexplored how to design effective DVFS strategies for GPU-based systems.

In this study, we take an experimental approach to investigate the effects of GPU DVFS on the performance, power, energy, and energy efficiency of compute-intensive workloads on a heterogeneous system that is built upon Intel's Sandy Bridge architecture and Nvidia's Kepler architecture. Particularly, we seek to quantify the impacts of GPU DVFS on an actual system, contrast them with those of CPU DVFS, analyze the factors and causes behind the general trends, and devise effective DVFS strategies.

In summary, our major findings are the following:

- By quantitatively evaluating the performance, power, and energy effects of the DVFS technology on a K20c GPU, we have been able to experimentally demonstrate that the GPU DVFS impact differs from CPU DVFS. Overall, on the matrix multiplication benchmark used in our study, the performance is linearly proportional to the GPU frequency when using a high memory speed. Both the system power and the GPU power can be approximated by a linear function of the GPU core frequency for all supported frequencies. The resulting energy consumption remains constant regardless of the selected frequency.
- For the studied workload on our platform, GPU

acceleration provides better performance (up to $6.7\times$) and is more energy efficient (up to $8.7\times$). Thus, if a workload can effectively run on a GPU, offloading this workload to the GPU is recommended for improved performance and energy efficiency.
- As a general trend, a higher GPU frequency delivers better performance without necessarily consuming more energy. In contrast, higher CPU performance states typically consume more energy. These trends imply that a higher GPU frequency is recommended as long as the corresponding power consumption falls within a practical power limit and the GPU memory subsystem is not a performance bottleneck. Note that the same conclusion does not apply to CPU DVFS.
- Through detailed power profiles, we observe that GPU DVFS only affects the power consumption on GPU devices but CPU DVFS impacts the CPU and other system components. Hence, GPU DVFS affects system energy efficiency less than CPU DVFS.
- The problem size is a key factor impacting the achieved system performance and energy efficiency. Thus, intelligently scheduling GPU workloads based on the problem size may result in improved performance and energy efficiency.

The remainder of this paper is organized as follows. In Section II, we provide a brief overview of related work. Section III describes our experimental methodology followed by a detailed analysis of the experimental results in Sections IV and V. Finally, our findings and conclusions are summarized in Section VI.

II. Related Work

As a powerful energy-saving technology, DVFS is available on multiple computer components, including processors [5], [12], memory modules [4], network devices [13], and hard drives [15]. Various types of computing systems, ranging from low-power mobile devices to high-performance servers have been using DVFS for power management. Because the literature on DVFS-related approaches is broad and we are primarily interested in studying the impact of DVFS on high-performance GPU computing, we only highlight the most relevant work.

A substantial amount of related work exists that focuses on DVFS-based power-aware high performance computing. Previous studies have shown that intelligent DVFS is able to save energy with minimal performance impact for a certain class of high performance applications that exhibit considerable CPU slack during execution. Whereas this slack can be caused by inter-process communication, off-chip memory access, disk I/O, or process synchronization, the general idea of an effective DVFS strategy is to detect such CPU-slack phases correctly and timely and then apply appropriate power-performance state transitions. For example, MPI calls can be intercepted in applications to locate communication phases during which scaling down the CPU speed may reduce power without impacting performance [11], [12]. Compilers can detect memory-bound regions and insert DVFS control commands into these regions [7], [14]. System-level scheduling can exploit various types of CPU slack for energy saving by monitoring and predicting system activities and demands [5], [9].

Due to fundamental differences in architecture and workload, an in-depth study of the impact of GPU DVFS in heterogeneous HPC environments is necessary. Realizing this need, a few groups have explored the effects of power-aware technology on early generations of GPUs. For example, Jiao et al. [10] have studied the power and performance behavior of three GPU kernels on an Nvidia GeForce GTX 280 under various GPU frequencies. They observed that increasing the GPU frequency bound leads to higher power consumption and better performance except for applications involving primarily memory accesses.

Our study differs from previous work in three main aspects. First, it is based on the experimental evaluation of a state-of-the-art system that includes the latest GPU and CPU technology. The Nvidia Kepler GPU and the Intel Sandy Bridge CPU provide several architectural features not available in earlier generations. For example, the user can select the GPU frequency on the K20c but not on the GTX 280. Second, we analyze the performance and energy impact of both GPU DVFS and CPU DVFS and compare the two approaches. Third, our study has led to several new findings that have not been reported previously.

III. Methodology

We use a combination of experimental investigation and comparative analysis to study the impact of GPU DVFS. On the one hand, we collect performance and power profiles of representative applications under various DVFS settings on a heterogeneous platform and quantify the actual impacts of GPU DVFS. On the other hand, we compare the results of GPU DVFS and CPU DVFS side by side to show their similarities and differences.

A. Experimental Platform

We conduct all experiments on a server node consisting of dual 8-core Xeon Sandy Bridge E5-2670 processors, one Nvidia Tesla K20c Kepler GPU card plugged into a PCIe slot, and 32 GB system memory. Each core of the E5-2670 processors has a 32 kB L1 instruction cache, a 32 kB L1 data cache, and a 256 kB unified L2 cache. All eight cores on a same die share a 20 MB L3 cache. The cores are DVFS capable and have 16 performance states, ranging from 1.2 GHz to 2.6 GHz in 0.1 GHz increments and, additionally, 2.601 GHz. Whereas each core of the E5-2670 supports two threads, we disabled this hyper-threading feature in our experiments to simplify the DVFS control and the performance analysis.

The K20c GPU implements the Nvidia GK110 Kepler architecture. It includes 13 SMX units with a total of 2496 CUDA cores, six 64-bit memory controllers, and 5 GB global memory. The GPU cores and memory are capable of DVFS and support the clock frequencies shown in Table I. Note that the core and memory speeds must be set together as a pair.

The server runs CentOS 6.1 64-bit Linux. For the Sandy Bridge CPU DVFS scheduling, we use the `cpufreq`

Table I
SUPPORTED MEMORY AND CORE FREQUENCY PAIRS ON K20C

Memory Freq. (MHz)	GPU Core Freq. (MHz)
2600	758 705 666 640 614
324	324

interface included in the Linux kernel. For the K20c GPU DVFS scheduling, we use the `nvidia-smi` utility provided by Nvidia with the `-ac` option.

B. Benchmark Selection

We consider two types of workloads. The first type reflects the characteristics of traditional high performance computing (HPC) applications, which can be decomposed into a large number of dependent small tasks. It is suitable for computing system capability tests by measuring the time to complete a given problem size and the largest problem the system can run. The second type of workload reflects typical high-throughput computing (HTC) applications, which can be decomposed into a large number of independent small tasks. It is suitable for computing system tests that benchmark the overall throughput.

For the HPC workload, we use a modified version of the matrix multiplication benchmark included in the CUDA SDK. The code can distribute the computation to GPUs and CPUs according to a distribution ratio specified by the user. The code uses a CUDA implementation for GPU computation and the OpenBLAS implementation [2] for CPU computation. For the HTC workload, we use TSP and FSM codes running on the ILCS Framework [3], which allows the user to specify whether to carry out the computation by the CPUs, GPUs, or both. A summary of these benchmark programs is provided in Table II.

C. Performance, Power, and Energy Profiling

We directly collect all performance and power data. Performance data, including the execution time and derived computation rates like GFLOPS and the task completion rate, are obtained by instrumenting the programs. Power data are collected by the eTune power-performance profiling framework [6]. Multiple streams of power samples are recorded, including the system power measured via two power supplies, the power of the two CPU packages, the power of the two memory controllers, and the power of the GPU card. eTune provides scripts to control the profile starting, stopping, annotating and logging. These power data are synchronized with the application execution and with each other via timestamps.

Specifically, we sampled power streams from the following sources. System power data are sampled by two external WattsUp power meters that are plugged in between the computer power supplies and a power outlet. The CPU power data stem from embedded power sensors on the Sandy Bridge processors and are gathered via the Running Average Power Limit (RAPL) interface [8]. GPU power data come from the embedded power sensor on the K20c card via Nvidia's System Management Interface (`nvidia-smi`). In this study, we use a sampling interval length of one second.

D. Performance Metrics

To examine the impact of DVFS, we examine four classes of performance metrics in our study.

Performance Time-to-solution or execution time is an essential performance measure. However, to compare performance across different problem sizes or system settings, a compute rate such as GFLOPS (billion floating-point operations per second) or a task completion rate is better. To reveal performance trends, a normalized compute rate relative to a base configuration is often helpful.

Power Power describes the energy consumption rate and is related to several physical constraints including power limits and the thermal envelope. We analyze four power measures including the system power, CPU power, GPU power, and the power consumption of all system components excluding CPUs and GPUs to evaluate the local and system-wide impact of CPU and GPU DVFS.

Energy Energy is the integral of power over time. Similar to the power metrics, we collect four types of energy measures corresponding to the above four types of power measures. As energy is not a rate, we only compare energy values that refer to the same computational problem.

Energy Efficiency Energy efficiency is a combined metric of performance and power. Among various combinations, we use the ratio of the compute rate to power, or, equivalently, the completed work per unit of energy to characterize the energy efficiency. The energy efficiency metric has a unit of GFLOPS/Watt or #Operations/Joule.

IV. IMPACT OF DVFS ON MATRIX MULTIPLY

To analyze the impact of DVFS on GPU-based heterogeneous computing systems, we evaluate the performance, energy consumption, and energy efficiency of the Matrix Multiply application on the platform described above under various GPU and CPU DVFS settings.

A. Experimental Settings

Matrix multiplication is a compute-intensive application that is representative of many scientific workloads. This benchmark computes the product of two dense matrices, i.e., $C_{n\times m} = A_{n\times k} * B_{k\times m}$. We use square matrices where $n = m = k$ in our experiments. For a given matrix size n, the running time complexity is $O(n^3)$.

We have modified the benchmark such that the computation can be distributed across the CPUs and the GPU in the system according to a user-specified distribution ratio ρ. When $\rho = 0\%$, all computation is performed by the GPU; when $\rho = 100\%$, all computation is performed by the CPUs. For GPU-only computation, the CPU speed is fixed at 2.6 GHz and the GPU speed is varied between all available frequencies shown in Table I. For CPU-only computation, the GPU speed is fixed at the default speed of 705 MHz and the CPU speed is varied between all available CPU frequencies. Moreover, the GPU energy is

Table II
APPLICATIONS AND BENCHMARKS UNDER STUDY

Benchmark	Description	HPC/HTC	Characterization	Origin
MatrixMultiply	Dense matrix multiplication	HPC	Floating-point, compute-intensive	CUDA SDK [1]
TSP	Traveling salesman problem	HTC	Integer, compute-intensive	ILCS [3]
FSM	Finite state machine	HTC	Integer, compute-intensive	ILCS [3]

Table III
NORMALIZED GFLOPS WITH VARYING MATRIX SIZE AND GPU FREQUENCY

Freq.	758M	705M	666M	640M	614M	324M
Rel. Freq.	(1.08)	(1.00)	(0.94)	(0.91)	(0.87)	(0.46)
400	1.07	1.00	0.94	0.91	0.86	*0.43*
800	1.07	1.00	0.95	0.91	0.87	*0.35*
1600	1.07	1.00	0.95	0.91	0.87	*0.26*
3200	1.07	1.00	0.94	0.91	0.87	*0.25*
6400	*1.02*	1.00	0.94	0.91	0.87	*0.24*
9600	*1.01*	1.00	0.94	0.91	0.87	*0.25*
12800	*1.05*	1.00	0.94	0.91	0.87	*0.26*

subtracted from the total system energy to reflect the actual energy cost of a typical system without a GPU.

B. Impact of GPU DVFS on Performance

Figure 1(a) shows the performance of Matrix Multiply for various matrix sizes and different GPU frequencies. Based on these results, we observe the following trends.

1) The matrix size significantly impacts the GPU performance, which increases rapidly up to a matrix size of 1600, above which it remains almost constant for all GPU frequencies. The maximum achieved performance is 1.1 TFLOPS, which is close to the double-precision peak performance of the K20c.
2) For all matrix sizes, the achieved performance is directly proportional to the selected GPU frequency except at 758 MHz and at 324 MHz. This trend is also reflected in Table III, which presents the normalized performance for each matrix size relative to that at the default frequency of 705 MHz.
3) At 324 MHz, the performance begins to satiate at a matrix size of $n = 800$ and then stays at roughly $1/4$ of the performance at 705 MHz.
4) The performance at 758 MHz follows the normal trend for matrix sizes under 3200 but drops below the expected trend for larger matrices.

Discussion: There are three distinct limitations that determine the GPU performance on the matrix multiplication code: 1) compute-bound execution where the performance is determined by the computing capability of the processing units, 2) memory-bound execution where the performance is limited by the available memory bandwidth, and 3) power-bound execution where the performance is constrained by the maximum allowed power consumption.

Compute-bound execution: As shown in Table III, when the memory frequency is 2600 MHz, the performance is proportional to the GPU's core frequency, implying the execution is compute-bound. By "compute-bound", we mean the performance of an application for a fixed problem size is determined by the GPU's maximum computing capability. Clearly, increasing GPU frequency will increase the GPU's computing capability and thereby deliver higher performance.

Memory-bound execution: If the memory frequency were 2600 MHz, the normalized performance at a 324 MHz core frequency would be 0.46 for all matrix sizes according to the previous discussion. However, when the memory frequency is also reduced to 324 MHz, the normalized performance at 324 MHz core frequency is only about 0.25 for $n \geq 1600$, implying the execution is memory-bound. For memory-bound execution, increasing the GPU's core frequency does not increase performance as it would for compute-bound execution.

Power-bound execution: The GPU performance trend at 758 MHz is the result of the third scenario. As shown in Figure 2(a) for a matrix size of 9600, after 60 seconds the power consumption at 758 MHz drops to the same level as that of the 705 MHz experiment. When the GPU runs in the maximum performance state for a sufficiently long time, its internal power and temperature management automatically lowers the frequency to protect the hardware. This explains why the relative performance shown in Table III decreases for input sizes above 3200, which result in a long enough runtime to trigger a frequency reduction. The fact that the K20c cannot maintain the maximum frequency of 758 MHz for an extended period of time also explains why the default frequency is 705 MHz.

C. Impact of GPU DVFS on System Energy

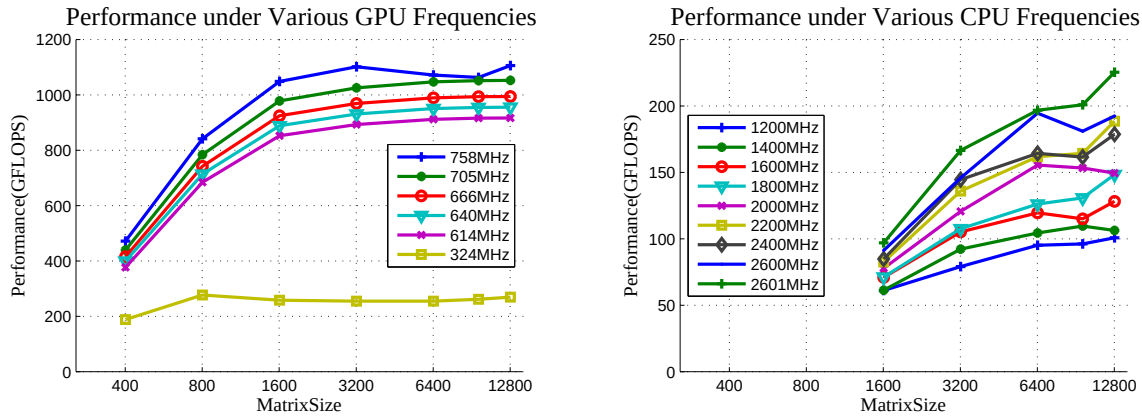
As shown in Figure 1(c), all performance states with a 2600 MHz memory frequency result in similar total system energy. The largest variation is less than 4%. However, the energy consumption at 324 MHz is $2.3\times$ as large as that at the other frequencies.

The similar energy consumptions can be explained by three relations: (1) the energy equation $E = P \times t$, (2) the inverse linear relation between execution time and GPU frequency, i.e., $t \propto \frac{1}{f}$, and (3) the roughly linear relation between the system power and the GPU frequency for the available GPU performance states with a 2600 MHz memory frequency.

D. Impact of GPU DVFS on System and GPU Power

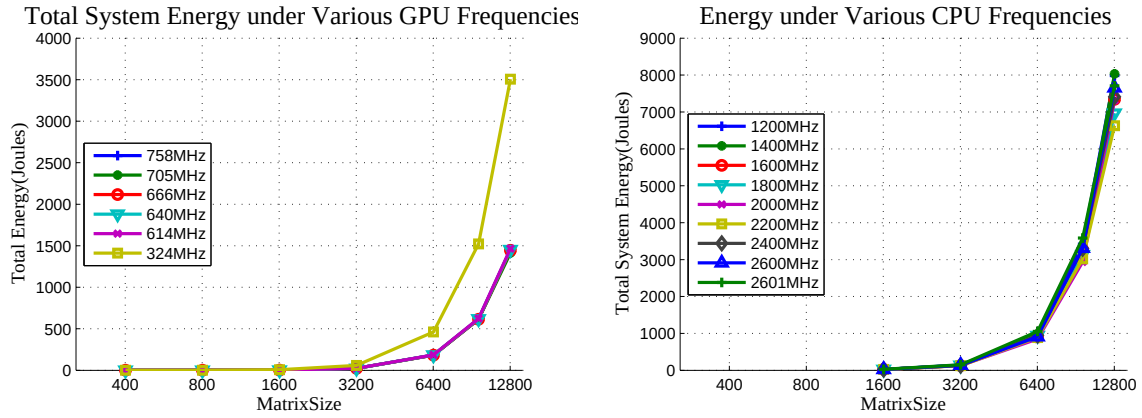
As mentioned in Section III, we collect power profiles from the power supplies, CPU packages, and the GPU card for all test cases. Figure 2 provides the GPU and CPU power profiles at various frequencies for a matrix size of 9600. These power profiles reveal the following.

1) GPU DVFS only affects the power consumption of the GPU card. During computation, the GPU power increases to 56 Watts at 324 MHz and 218 Watts



(a) Performance Impact of GPU DVFS

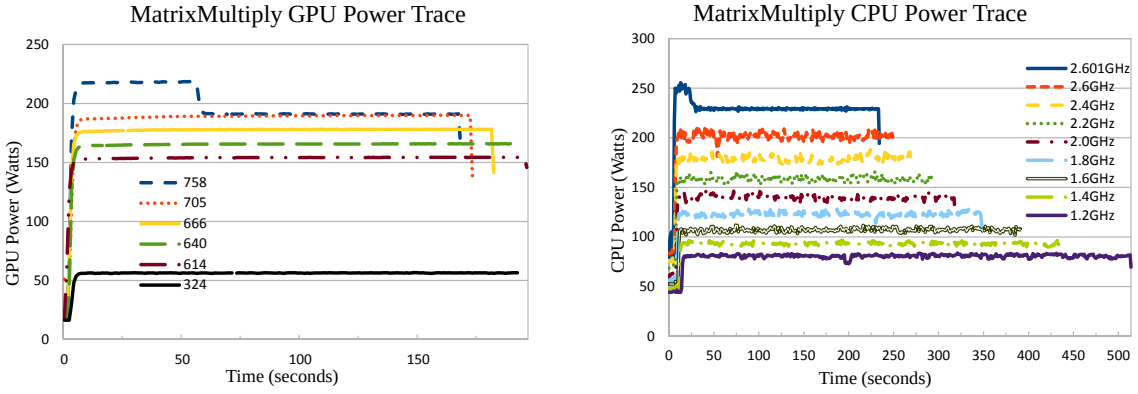
(b) Performance Impact of CPU DVFS



(c) Energy Impact of GPU DVFS

(d) Energy Impact of CPU DVFS

Figure 1. Matrix Multiply performance and energy for different GPU and CPU clock frequencies. In (c), the energy is the total system energy. In (d), the energy is the total system energy without the GPU energy. In (b) and (d), the performance and energy for small matrix sizes are not presented because the runtime is too short to obtain accurate measurements.



(a) GPU power profile for a matrix size of 9600

(b) CPU power profile for a matrix size of 9600

Figure 2. Power profiles of Matrix Multiplication. The matrix size is 9600 x 9600. 100 iterations are shown for GPU computing in Figure 2(a) and 30 iterations are shown for CPU computing in Figure 2(b). The power profile for the GPU running at 324 MHz is incomplete.

at 758 MHz. Meanwhile, the power consumption of all other components (excluding the GPU) stays at around 178 Watts. In contrast, CPU DVFS causes a considerable power variation not only in the CPUs but also in other system components. During computation, the CPU package power ranges between 81 Watts at 1.2 GHz and 209 Watts at 2.6 GHz, and the power consumption of the other system components excluding the CPUs and GPUs increases from 85 Watts at 1.2 GHz to 129 Watts at 2.6 GHz.

2) When idle, the GPU power consumption is roughly 16 Watts for all performance states whereas the CPU idle power is much higher. In addition, the CPU idle power increases with increasing processor frequency

from 45 Watts at 1.2 GHz to 83 Watts at 2.6 GHz.

3) Under load, the GPU power linearly increases with the GPU frequency at about 0.5 W/MHz excluding 324 MHz performance state.
4) During execution, the GPU power is constant over time for all available GPU frequencies except 758 MHz, where it drops by 18 Watts after 60 seconds. This power drop coincides with the aforementioned lowering of the GPU frequency from 758 MHz to 705 MHz triggered by the GPU's internal thermal management and power capping.
5) Coincidentally, the CPU packages and the GPU card consume similar amounts of power when all cores are running at their default speeds. In this case, the GPU card consumes 190 Watts of power and the two CPU packages consume 200 Watts. However, GPU computing causes a higher system power consumption (362 Watts) than CPU computing (338 Watts) because GPU computing still uses the host CPUs as the GPU is only a co-processor.

E. Impact of GPU DVFS on Energy Efficiency

To quantify the combined effects of DVFS on performance and energy, we derived energy efficiency numbers from the performance and energy data and summarize the results in Figure 3. We characterize the energy efficiency by the metric of Performance/Power, which is equivalent to #Operations/Energy. Because individual component DVFS affects component and system performance metrics differently, we distinguish energy efficiency at the component and system levels. Our findings are as follow.

First, the matrix size significantly impacts the energy efficiency at the component and system levels. For GPU computing, the energy efficiency rapidly increases from 3.5 GFLOPS/Watt to 5 GFLOPS/Watt as the matrix size grows from 400 to 1600. The efficiency remains relatively stable when the matrix size is further increased.

Second, at the GPU device level, there are obvious differences between the DVFS settings. Among the available settings with a memory frequency of 2600 MHz, 614 MHz delivers the highest energy efficiency and 758 MHz the lowest for fairly large data sizes ($n > 1600$). 324 MHz is the least efficient for large data sizes. A noteworthy observation is that 758 MHz is not most energy efficient even though it delivers the highest performance. The maximum energy efficiency at the device level is 6.28 GFLOPS/Watt, which is achieved at 614 MHz for a matrix size of 12800.

Third, the energy efficiencies at the system level show different trends. With the exception of 324 MHz being the worst due to a significantly longer execution time, all other performance states follow a similar energy efficiency curve. This similarity implies: (1) Since all GPU frequencies except 324 MHz are equally energy efficient, performance metrics such as execution time or GFLOPS are needed to determine the optimal configuration; and (2) effective DVFS scheduling at the component level does not necessarily lead to a noticeable improvement in efficiency at the system level.

Fourth, the impact of CPU DVFS on the energy efficiency significantly differs from that of GPU DVFS in the following aspects. Overall, GPU computing is $5.6\times$ as energy efficient as CPU computing when using the default settings and $4.6\times$ when using the optimal settings. Unlike GPU DVFS, CPU DVFS significantly affects the system level energy efficiency, implying practical benefits of CPU DVFS on system energy optimization. In addition, the growth of the energy efficiency with matrix size does not stop within the range of matrix sizes we have studied.

F. Comparison between GPU DVFS and CPU DVFS

The effect of CPU DVFS on the performance, energy, power, and energy efficiency as shown in Figures 1(b), 1(d), 2(b), 3(b) are quite different from those of GPU DVFS in the following aspects.

First, GPU computing and CPU computing result in different absolute values in performance, energy, and efficiency for the same matrix size. When the GPU and CPUs run at their default frequencies, the two CPUs only deliver $\frac{1}{5}$ of the GPU's performance but consume about $5\times$ more energy. In our experiments, the maximum achieved performance is 213 GFLOPS on the studied matrix sizes when all 16 CPU cores run at 2.6 GHz with hyperthreading disabled. For reference, the theoretical peak performance of two SandyBridge E5-2670 CPUs is 332.8 GFLOPS when hyperthreading is enabled. This performance gap between GPU and CPU computing is the primary cause for the gaps in energy consumption and energy efficiency.

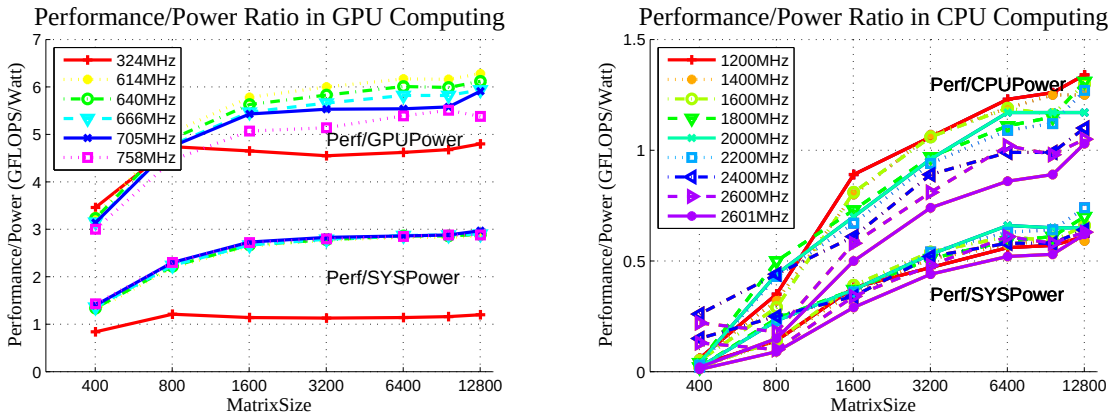
Second, while a higher frequency generally results in higher performance for both GPU and CPU computing, CPU DVFS has a larger impact on application performance with a swing amplitude of up to 50%. The larger impact is mainly due to the larger range of available CPU frequencies. Due to the memory hierarchy, CPU computing performance is not directly proportional to the CPU frequency. As different CPU performance states may deliver similar application performance, CPU DVFS has the potential to save energy with little performance impact.

Third, the CPU's computing performance increases with the matrix size and does not plateau out for the set of matrix sizes we studied. This implies that, on a single system with the current configuration, CPU computing is capable of solving larger problems than GPU computing.

Fourth, CPU DVFS has a larger impact on the system energy efficiency in CPU computing than GPU DFVS in GPU computing. Consequently, CPU DVFS provides more energy saving opportunities than GPU DVFS in high-performance computing, even though, in general, GPU accelerated heterogeneous computing is much more energy efficient than traditional CPU computing.

V. Impact of DVFS on TSP and FSM

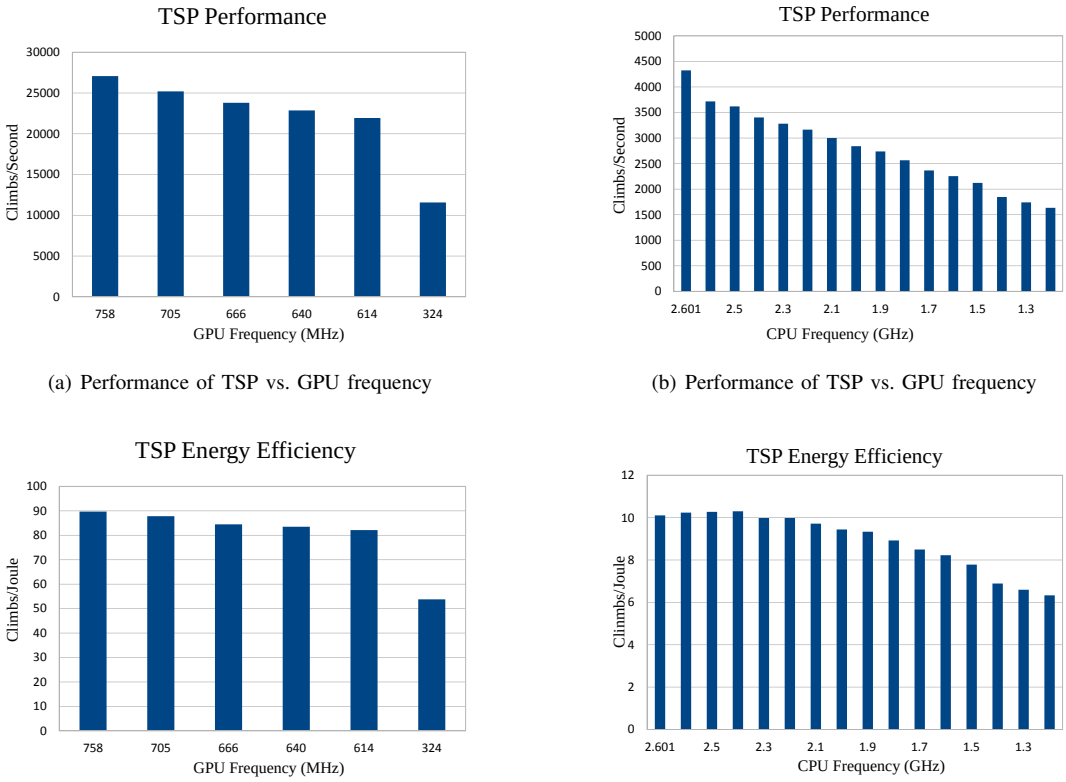
We use TSP and FSM to demonstrate the impact of DVFS on compute-intensive high-throughput computing. The performance is presented as a compute rate in work/time and energy efficiency as a work/energy ratio. We focus on a single large data size that satiates all



(a) GPU and system energy efficiency under GPU DVFS

(b) GPU and system energy efficiency under CPU DVFS

Figure 3. The energy efficiency of GPU and CPU DVFS for MatrixMultiply. The figures show both GFLOPS over GPU or CPU component power and system power.



(a) Performance of TSP vs. GPU frequency

(b) Performance of TSP vs. GPU frequency

(c) Energy efficiency of TSP vs. GPU frequency

(d) Energy efficiency of TSP vs. CPU frequency

Figure 4. TSP performance and energy efficiency with various DVFS settings

processing units and delivers high throughput. Figures 4(a) and 4(c) correspond to GPU computing on a K20c with various GPU DVFS settings and Figures 4(b) and 4(d) correspond to CPU computing on 16 CPU cores with various CPU DVFS settings. In the latter figures, energy efficiency is total completed work divided by system energy, which excludes the GPU energy consumption. In GPU computing, all CPUs run at the default 2.6 GHz. We focus on TSP as the DVFS trends on FSM are very similar.

The GPU frequency proportionally affects the performance without the usual exception at 324 MHz, as shown in Table IV. This proportional relation indicates that TSP's performance is predominated by computation and not memory accesses. TSP's energy efficiency also increases with frequency but not in a directly proportional fashion due to the energy consumption of other components. For TSP, the highest GPU frequency delivers optimal performance and energy efficiency and consumes the least amount of energy for a given amount of work.

In CPU computing, the CPU frequency also significantly affects the performance and the relation is close to directly proportional. The highest frequency delivers

Table IV
NORMALIZED PERFORMANCE AND ENERGY EFFICIENCY FOR TSP

Freq.	758M	705M	666M	640M	614M	324M
Rel. Freq.	(1.08)	(1.00)	(0.94)	(0.91)	(0.87)	(0.46)
Performance	1.07	1.00	0.94	0.91	0.87	*0.46*
Energy Eff.	1.02	1.00	0.96	0.95	0.94	*0.61*

the best application performance, which is expected as TSP is compute intensive. The energy efficiency first increases as the frequency increases but then decreases as frequency increases further. The most energy efficient state is between 2.4 GHz and 2.5 GHz. The highest performance state is not the most energy efficient and thus consumes more energy for a given amount of work.

Overall, when the GPUs and CPUs run at the default performance states, GPU computing is $6.7\times$ as fast and $8.7\times$ as energy efficient as CPU computing. Note that these numbers are larger than those for MatrixMultiply because all GPUs are fully utilized for TSP. Even the worst GPU computing performance is about $3\times$ as good as the best CPU computing performance, while the lowest GPU computing energy efficiency is $5\times$ as good as the highest CPU computing energy efficiency.

VI. CONCLUSIONS

In this paper, we experimentally study the effects of CPU and GPU DVFS on the performance and energy efficiency of scientific workloads on a GPU-accelerated heterogeneous system built upon the state-of-the-art Intel Sandy Bridge architecture and the Nvidia Kepler architecture. This study provides first-hand quantitative data describing the general trends and exceptions of the impact DVFS has on heterogeneous platforms.

We find that DVFS in GPU computing behaves significantly differently from DVFS in CPU computing regarding energy efficiency. However, both technologies result in similar performance trends: high performance states generally deliver better application performance. For compute-intensive high performance and throughput applications with fairly large data sizes, all GPU DVFS settings except the lowest one consume the same amount of energy with similar system energy efficiency. Consequently, the highest available GPU DVFS setting is optimal in terms of both performance and energy efficiency. In practice, the default setting, which is the second highest, seems to be the one to go with because the highest setting may cause thermal and power emergencies.

In the future, we will expand our work to a wider spectrum of GPU applications that stress different behaviors, including compute- and memory-bound codes as well as regular and irregular codes, and develop analytical models to guide energy-aware GPU computing.

ACKNOWLEDGEMENTS

This work is supported in part by the U.S. National Science Foundation under Grants No. CNS-1116691, CNS-1118043, DUE-1141022, and CNS-1217231 as well as by donations from Nvidia Corporation.

REFERENCES

[1] CUDA Toolkit. https://developer.nvidia.com/cuda-toolkit.

[2] OpenBLAS. http://xianyi.github.io/OpenBLAS/.

[3] M. Burtscher and H. Rabeti. A Scalable Heterogeneous Parallelization Framework for Iterative Local Searches. May 2013.

[4] X. Fan, C. Ellis, and A. Lebeck. The Synergy between Power-aware Memory Systems and Processor Voltage Scaling. *Power-Aware Computer Systems*, pages 151–166, 2005.

[5] R. Ge, X. Feng, W.-C. Feng, and K. W. Cameron. CPU MISER: a Performance-Directed, Run-Time System for Power-Aware Clusters. In *International Conference in Parallel Processing (ICPP) 2007*, Xian, China, 2007.

[6] R. Ge, X. Feng, T. Wirtz, Z. Zong, and Z. Chen. eTune: A Power Analysis Framework for Data-Intensive Computing. In *Parallel Processing Workshops (ICPPW), 2012 41st International Conference on*, pages 254 –261, sept. 2012.

[7] C.-H. Hsu and U. Kremer. The Design, Implementation, and Evaluation of a Compiler Algorithm for CPU Energy Reduction. In *PLDI '03: Proceedings of the ACM SIGPLAN 2003 conference on Programming language design and implementation*, pages 38–48, New York, NY, USA, 2003. ACM.

[8] Intel. Volume 3B: System Programming Guide, Part 2. *Intel 64 and IA-32 Architectures Software Developers Manual*, June 2013.

[9] C. Isci, G. Contreras, and M. Martonosi. Live, runtime phase monitoring and prediction on real systems with application to dynamic power management. In *MICRO*, pages 359–370, 2006.

[10] Y. Jiao, H. Lin, P. Balaji, and W. Feng. Power and Performance Characterization of Computational Kernels on the GPU. In *Green Computing and Communications (GreenCom), 2010 IEEE/ACM Int'l Conference on Int'l Conference on Cyber, Physical and Social Computing (CPSCom)*, pages 221–228, 2010.

[11] M. Y. Lim, V. W. Freeh, and D. K. Lowenthal. MPI and Communication - Adaptive, Transparent Frequency and Voltage Scaling of Communication Phases in MPI Programs. In *Proceedings of the ACM/IEEE Supercomputing 2006 (SC'06)*, 2006.

[12] B. Rountree, D. Lownenthal, B. de Supinski, M. Schulz, V. Freeh, and T. Bletsch. Adagio: Making DVS Practical for Complex HPC Applications. In *Proceedings of the 23rd international conference on Supercomputing*, pages 460–469. ACM, 2009.

[13] V. Soteriou, N. Eisley, and L.-S. Peh. Software-directed Power-aware Interconnection Networks. *ACM Transactions on Architecture and Code Optimization (TACO)*, 4(1):5, 2007.

[14] Q. Wu, V. Reddi, Y. Wu, J. Lee, D. Connors, D. Brooks, M. Martonosi, and D. W. Clark. A Dynamic Compilation Framework for Controlling Microprocessor Energy and Performance. In *the 38th IEEE/ACM International Symposium on Microarchitecture*, pages 271–282, Barcelona, Spain, 2005.

[15] Q. Zhu, Z. Chen, L. Tan, Y. Zhou, K. Keeton, and J. Wilkes. Hibernator: Helping Disk Array Sleep Through the Winter. In *the 20th ACM Symposium on Operating Systems Principles (SOSP'05)*, 2005.