A survey and measurement study of GPU DVFS on energy conservation

Xinxin Mei  
*Hong Kong Baptist University*

Qiang Wang  
*Hong Kong Baptist University*

Xiaowen Chu  
*Hong Kong Baptist University, chxw@comp.hkbu.edu.hk*

This document is the authors' final version of the published article.  
Link to published article: https://doi.org/10.1016/j.dcan.2016.10.001

APA Citation

A survey and measurement study of GPU DVFS on energy conservation

Xinxin Mei\textsuperscript{a}, Qiang Wang\textsuperscript{a}, Xiaowen Chu\textsuperscript{a,b,}\textsuperscript{*}

\textsuperscript{a} Department of Computer Science, Hong Kong Baptist University, Hong Kong, China
\textsuperscript{b} Institute of Research and Continuing Education, Hong Kong Baptist University, Shenzhen 518057, China

ARTICLE INFO

Keywords:
Graphics processing unit
Dynamic voltage and frequency scaling
Energy efficiency

ABSTRACT

Energy efficiency has become one of the top design criteria for current computing systems. The Dynamic Voltage and Frequency Scaling (DVFS) has been widely adopted by laptop computers, servers, and mobile devices to conserve energy, while the GPU DVFS is still at a certain early age. This paper aims at exploring the impact of GPU DVFS on the application performance and power consumption, and furthermore, on energy conservation. We survey the state-of-the-art GPU DVFS characterizations, and then summarize recent research works on GPU power and performance models. We also conduct real GPU DVFS experiments on NVIDIA Fermi and Maxwell GPUs. According to our experimental results, GPU DVFS has significant potential for energy saving. The effect of scaling core voltage/frequency and memory voltage/frequency depends on not only the GPU architectures, but also the characteristic of GPU applications.

1. Introduction

The Graphics Processing Units (GPUs) have become prevalent accelerators in current high performance clusters. They substantially boost the performance of a great number of applications in many commercial and scientific fields, such as bioinformatics\cite{1,2}, computer communications\cite{3,4,5}, machine learning\cite{6,7,8}, especially the emerging deep learning\cite{9,10,11}. In the TOP500 supercomputer list\cite{12} as of June, 2016, 94 systems are equipped with accelerators and 69 out of them are equipped with GPUs\cite{13}. The CPU-GPU hybrid computing is more energy efficient than traditional many-core parallel computing\cite{13,14}. However, this kind of high performance clusters still consume a lot of energy. To power the clusters remains a great expense. For example, the Titan supercomputer, 3rd in the TOP500 list as of this writing, is accelerated by 18,688 NVIDIA Tesla K20X with a power supply of 8.21 million Watts, which cost about 23 million dollars per year\cite{15}. Given the fact that saving even a few percent of energy can reduce a large amount of electricity cost, efficient GPU power management becomes indispensable for GPU-accelerated data centers and supercomputers.

One of the promising power management strategies is the Dynamic Voltage and Frequency Scaling (DVFS)\cite{16,17}, which refers to changing the processor voltage/frequency during task processing. It is effective in either saving energy or improving performance. The CPU DVFS technology is well developed and has been adopted in both personal computing devices and large scale clusters\cite{18}. Despite the maturity of CPU DVFS, the GPU DVFS study started only a few years ago. According to existing studies, simply transplanting the CPU DVFS strategy to GPU platforms could be ineffective\cite{19,20}. For example, scaling up the processor frequency (described as “racing” in\cite{20}) is proved to be energy efficient for the CPUs but not always for the GPUs\cite{20,21}. We summarize some challenges of the GPU DVFS study as below. Firstly, the GPU hardware and power management information is very limited. Second, there lacks accurate quantitative GPU DVFS performance/power estimation tools. Lastly, the GPU architecture design is being advanced very fast, and performing the same DVFS strategy may have different outcomes on different generations of GPUs.

In\cite{21}, Mittal et al. surveyed the research work on analyzing and improving energy efficiency of GPUs, including the GPU DVFS. Different from their broad scope, in this paper we are more focused to investigate the current status of GPU DVFS study. We aim at understanding the impact of GPU DVFS on the performance or power consumption, especially for recent NVIDIA GPU products. We consider our contributions of two aspects. On one hand, we summarize the most up-to-date GPU DVFS studies and the GPU performance and power modeling techniques. Our work provides state-of-the-art investigation and observation on GPU DVFS. On the other hand, we conduct DVFS measurement experiments on recent NVIDIA Fermi and Maxwell platforms. Our experimental results can serve as GPU DVFS benchmarks and our experimental findings reveal the similarities and the differences of GPU DVFS effects on two generations of GPU platforms.

\bibitem{1} Corresponding author.

\textit{E-mail addresses:} xxmei@comp.hkbu.edu.hk (X. Mei), qiangwang@comp.hkbu.edu.hk (Q. Wang), chxw@comp.hkbu.edu.hk (X. Chu).

http://dx.doi.org/10.1016/j.dcan.2016.10.001
Received 16 May 2016; Received in revised form 10 September 2016; Accepted 24 October 2016
Available online 02 December 2016
2152-8648/ © 2017 Chongqing University of Posts and Telecommunications. Production and hosting by Elsevier B.V. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).
The rest of this paper is organized as follows. Section 2 presents the GPU architecture and voltage/frequency scaling interface across five generations of NVIDIA GPUs. Section 3 characterizes the impact of GPU voltage and frequency scaling. Section 4 demonstrates the latest GPU DVFS power modeling, including both empirical and statistical ones. The GPU DVFS performance modeling is discussed in Section 5. In Section 6, we conduct real frequency scaling on the Maxwell GPU and voltage/frequency scaling on the Fermi GPU. We analyze the scaling effects and summarize the findings. Our work is concluded in the last section.

2. Background

In this section, we introduce some fundamental knowledge on the GPU architecture and the GPU voltage/frequency scaling interface.

2.1. GPU architecture

Fig. 1 shows a brief block diagram of the NVIDIA Maxwell GTX980 GPU. A GPU board contains the GPU chipset and GPU memory. The GPU consists of the voltage and the cache processes of the multiprocessors (SMs). Fig. 2 illustrates the block diagram of the SM of a Maxwell GPU. In the literature, a Floating Point (FP) process unit is usually referred to as a GPU core. The number of FP units and other micro-units on an SM varies, depending on the GPU products. The SMs and the L2 cache are connected to the GPU memory module, which includes multiple GDDR RAM or the recent HBM stacked RAMs, via memory controllers.

Till 2016, NVIDIA has launched five generations of GPUs, as listed in Table 1. The Compute Capability (CA) is used by NVIDIA to distinguish different architectures of the GPU products. While the micro-architectures of Fermi to Pascal GPUs are based on similar designs, there is a big difference between the Tesla GPUs and later GPUs: the cache system. For the 1st generation Tesla GPUs, normal data access is not cached [24, 25]. By introducing cache system, a great number of applications have received further boosted performance. It is vital to consider the influence of the GPU caches on the application performance as well as energy consumption [26, 27].

Table 1
Generations of NVIDIA GPUs.

<table>
<thead>
<tr>
<th>Micro-architecture</th>
<th>Year</th>
<th>CA</th>
<th>Feature size (nm)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tesla</td>
<td>2006</td>
<td>1.x</td>
<td>&gt;55</td>
</tr>
<tr>
<td>Fermi</td>
<td>2009</td>
<td>2.x</td>
<td>45</td>
</tr>
<tr>
<td>Kepler</td>
<td>2012</td>
<td>3.x</td>
<td>28</td>
</tr>
<tr>
<td>Maxwell</td>
<td>2014</td>
<td>5.x</td>
<td>28</td>
</tr>
<tr>
<td>Pascal</td>
<td>2016</td>
<td>6.x</td>
<td>16</td>
</tr>
</tbody>
</table>

Table 2

<table>
<thead>
<tr>
<th>State</th>
<th>P0</th>
<th>P2</th>
<th>P5</th>
<th>P8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default/range of core voltage (V)</td>
<td>0.987 [0.987, 1.187]</td>
<td>0.987 [0.987, 1.187]</td>
<td>0.85 [0.85, 1.187]</td>
<td>0.85 [0.85, 1.187]</td>
</tr>
</tbody>
</table>

2.2. GPU DVFS

For the CMOS circuits, the total power consumption, denoted by $P_{\text{total}}$, is decomposed into dynamic and static parts. We list the power partition in Eq. (1), where $P_{\text{dynamic}}$ stands for the dynamic power, which is generated when transistors switch their states; $P_{\text{leakage}}$, $P_{\text{short-circuit}}$, and $P_{\text{DC}}$ together stand for the static power. [34] provides the elaboration of the power decomposition.

$$P_{\text{total}} = P_{\text{dynamic}} + P_{\text{leakage}} + P_{\text{short-circuit}} + P_{\text{DC}}$$

Eq. (2) gives the general form of the dynamic power, where $a$ denotes the average utilization ratio, $C$ denotes the total capacitance, $V$ denotes thechip supply voltage and $f$ denotes the operating frequency [16].

$$P_{\text{dynamic}} = aCV^2f$$

DVFS changes the runtime supply voltage and the frequency, and it mainly affects the dynamic power. For the early processing units, the dynamic power accounts for the majority of power consumption, but nowadays the static power is also contributing considerably [35, 36].

In general, the GPU boards have two sets of adjustable voltage/frequency: the core voltage/frequency, and the memory voltage/frequency. The core and memory voltage refer to the supply voltage of the GPU SMs and the DRAM. The core frequency affects the SM execution speed, while the memory frequency actually affects the DRAM I/O throughput.

On some NVIDIA products, the GPU core/memory voltage and frequency are almost continuously scalable within a wide range, with the help of proper over-clocking tools [37, 38]. Recently, NVIDIA has introduced the concept of P-states. A P-state defines a combination of GPU voltage and frequency settings. For example, on our ASUS Strix GeForce GTX 980 (OC edition), there are at least 4 P-states: P0, P2, P5 and P8, whose default voltage/frequency settings as well as the allowed

Fig. 1. The block diagram of NVIDIA GTX980 GPU board.

Fig. 2. The stream multiprocessor of NVIDIA GTX980 GPU.
scaling ranges are listed in Table 2. P8 is the idle state which consumes little energy but cannot run any tasks. P5 offers a wide scaling range for core voltage and frequency; but on the other hand it can only support very low memory frequency. P2 is a powerful state which can provide highest voltage and frequency. P0 has the same scaling ability with P2 except its higher default memory frequency. Notice that the scaling range can be vendor-dependent, and the GPU may not work reliably if the voltage is set too high.

NVIDIA offers the NVIDIA Management Library (NVML) [39] and the NVIDIA System Management Interface (nvidia-smi) [40] to monitor its GPU P-states. Some third-party softwares, like the NVIDIA Inspector [41] and the Afterburner [42], can manually adjust the voltage/frequency with a certain level of flexibility. NVIDIA has also launched GPU Boost [43], an embedded thermal constrained DVFS system, which makes the manual voltage scaling rather tough. In contrast, AMD and some SoC platforms provide more user-friendly GPU voltage/frequency scaling interfaces.

3. GPU DVFS characterization

There have been a number of recent studies on GPU DVFS, which are conducted through either real experiments or computer simulations. The experimental studies refer to those scaling the voltage or frequency of GPUs in reality, and in this paper, we mainly focus on the NVIDIA GPUs. The simulation studies refer to those scaling on simulators, like GPUPWatch, or those lacking practical experimental results. Most of the experimental studies applied the GPU frequency scaling only, due to the limited support of GPU voltage scaling tools. The simulation studies discuss various scaling approaches, including the GPU core number scaling, per-core DVFS, etc., benefiting from the more flexible scaling interfaces. Both the experimental and simulation results suggest that the GPU DVFS is effective in conserving energy.

3.1. Experimental studies

Jiao et al. scaled the core frequency and the memory frequency of a NVIDIA Tesla GTX280 GPU with three typical applications: the compute-intensive dense matrix multiply, the memory-intensive dense matrix transpose, and the hybrid fast Fourier transform (FFT) [44]. The three applications showed different performance and energy efficiency curves with the same core-memory frequency settings: the dense matrix multiply was insensitive to memory frequency scaling, FFT benefited from low core frequency and high memory frequency, while dense matrix transpose needed both high core and memory frequency. They also found that the energy efficiency was largely determined by the Instructions Per Cycle (IPC) and the ratio of the amount of global memory transactions over the amount of computation transactions.

Ma et al. designed an online management system that integrated the GPU dynamic core and memory frequency scaling and the CPU-GPU workload division [45]. On their testbed, NVIDIA GeForce8800, the GPU frequency scaling alone saved about 6% of system energy and 14.5% of GPU energy.

Ge et al. applied fine-grained GPU core frequency and coarsely-grained GPU memory frequency on a Kepler K20c GPU, and compared its effect to the CPU frequency scaling [19]. They found that for dense matrix multiply, both the GPU power and the GPU performance were linear to the CPU core frequency, and the GPU energy consumption was insensitive to frequency scaling. For their three tested applications, the highest GPU frequency always resulted in best energy efficiency, differing from the CPU DVFS.

In our previous work, we scaled the core voltage, the core frequency and the memory frequency of the Fermi GTX560Ti GPU, with a set of 37 GPU applications [38]. We found that the effect of GPU DVFS depends on the application characteristics. The optimal setting to consume the least energy was a combination of appropriate GPU memory frequency and core voltage/frequency. We observed an average of 20% reduction of energy consumption with only 4% of performance loss.

Abe et al. combined the GPU core frequency, the GPU memory frequency and the CPU core frequency scaling together, on the NVIDIA Fermi GTX480 GPU [20]. They performed the frequency scaling with dense matrix multiply of various matrix sizes. They could save as much as 28% of the system energy with a small matrix size, low GPU memory frequency and high GPU core frequency. They then extensively scaled the GPU core and memory frequency of 4 GPU products, including the Tesla GTX285, Fermi GTX460/GTX480, and Kepler GTX680, with 33 popular applications [22]. They set both of the core and memory frequency to low, medium and high values, and searched for the optimal core-memory frequency combination that offered the best power efficiency. Surprisingly, they found that, for the Kepler GTX680, the default frequency configuration was never optimal, while the opposite for the Tesla GTX285. They could reduce as much as 75% of system energy within 30% of performance loss, for a compute-intensive workload on the Kepler GPU. Their results suggested that DVFS was even more appealing for recent GPUs.

You and Chung designed a performance-guaranteed DVFS algorithm for the Mali-400MP GPU on a SoC platform [46]. They found that the GPU utilization ratio was not tightly correlated to the GPU performance, and the on-demand DVFS provided by the SoC system was inadequate by wasting a certain amount of power.

Jiao et al. studied the GPU core and memory frequency scaling for two concurrent kernels on the Kepler GT640 GPU [47]. They took a set of kernels from the CUDA SDK and Rodinia benchmark and measured their energy efficiency (GFlops/Watt) with different core-memory frequency settings. They demonstrated that the concurrent kernel execution in combination with GPU DVFS can improve energy-efficiency by up to 34.5%.

The above measurement studies offer the ground truth that the GPU DVFS is effective in saving energy, and meanwhile does not sacrifice much performance. For recent Kepler GPUs, DVFS is even more promising in energy-efficient computing.

3.2. Simulation studies

In [48], Lee et al. simulated the GPU DVFS as well as the core number scaling in GPGPU-Sim, based on the 32 nm prediction technology model [49], with the objective to improve the throughput. Their scaling scheme can provide an average of 20% higher throughput than the baseline GPU.

Leng et al. developed GPUPWatch, which could simulate the cycle-level GPU voltage/frequency scaling, based on the Fermi GTX480 GPU [33]. They configured the various GPU voltage/frequency settings according to the 45 nm prediction technology model [49,50], and simulated both slow off-chip and prompt on-chip DVFS. They gained an average of 13.2% energy saving with off-chip DVFS and 14.4% energy saving with on-chip DVFS, both within 3% performance loss. For either scaling scheme, they found that the memory-bounded kernels benefited a lot but the purely compute-bounded kernels did not take much advantage of the DVFS.

Sethia et al. designed a dynamic runtime GPU core number, core and memory frequency scaling system, to either conserve the energy or improve the performance [51]. They categorized the GPU applications into 3 types: compute-intensive, memory-intensive, and cache sensitive, according to GPUPWatch characterizations. For each application category and scaling objective, they designed different scaling strategies. Their system reduced 15% energy in the energy-saving mode.

Sen et al. applied the fine-grained per-core DVFS in GPUPWatch, in view of the diverse execution time and workload of different GPU cores [52]. They found the per-core DVFS had good potential to save more power than the overall DVFS.
Motivated by the fact that scaling down the core voltage was vital to conserve energy, Gopireddy et al. designed a new architecture that enabled a lower operating voltage in the energy-efficiency mode other than the normal voltage in the high-performance mode [53]. Their simulation results showed that the new hardware could reduce as much as 48% of energy consumption, compared to the conventional hardware with normal DVFS.

In summary, GPU DVFS is proved to be effective in energy conservation for a variety of applications, but the impact on different applications are very diverse. Researchers need to design specific scaling algorithm based on the application characteristics.

4. GPU DVFS runtime power modeling

In this section, we survey the runtime GPU power modeling work. We classify the studies into either empirical or statistical, where the former one relies on the binary code analysis and the latter one relies on the program performance counters. The empirical method is a bottom-up approach and requires break-up of GPU micro-architectures, while the statistical method treats the GPU hardware as a black box and seeks statistical relationships between the GPU power and the runtime performance counters.

4.1. Empirical methods

The empirical power modeling method was first presented by Isci and Margaret to measure Pentium 4 power consumptions [54]. It manually decomposed a whole board into separate hardware components. For each component, they estimated the maximum power consumption and computed the access rate. The total power consumption was the summation of these components. Eq. (3) shows the mathematical form of the empirical power model, where $P_i$, $P_2$, ..., $P_n$ are the maximum power consumption of the $n$ sub-components, $r_1$, $r_2$, ..., $r_n$ are the access rates of the sub-components, and $P_0$ is a constant parameter.

$$P = P_0 + P_1 r_1 + P_2 r_2 + \cdots + P_n r_n$$

(3)

Hong and Kim utilized this method for a GTX280 GPU [35]. They estimated the access rates based on the dynamic number of instructions and the execution cycles of separate GPU units, where the number of instructions were based on the binary PTX code analysis, and the execution cycles were based on the pipeline analysis. They then designed a suite of micro-benchmarks to search for $P_i$, ..., $P_n$, that gave the minimum error between the measured power and the computed power. With the above two approaches, they got the baseline runtime GPU power consumption. They also built a power/temperature increase model to account for the fact that the GPU runtime power increases as the chip temperature rises. The final GPU power consumption was the sum of the baseline power consumption and the increment. They achieved 2.5% of prediction error when evaluating the micro-benchmarks, and 9.2% of error when evaluating the integrated GPGPU kernels. Besides, the model also considered the influence of the active SM numbers, and the authors used the model to study the GPU energy conservation with fewer SMs. The authors also extended the study to the Fermi GPU, by involving the cache-stressed micro-benchmarks and adjusting the model parameters [36].

Leng et al. packed Hong and Kim’s power modeling with GPGPUSim to form GPUWattch, which could estimate the runtime GPU power with different voltage/frequency settings at cycle-level [33]. The authors refined Hong and Kim’s model with a large amount of micro-benchmarks, to overcome the power uncertainties brought by the new Fermi hardware. On the Fermi GTX480 GPU, the prediction error was 15% for the micro-benchmarks, and 9.9% for the general GPU kernels. Besides, the model could capture the runtime power phase behaviours. They validated the model at a rate of 2 mega-samples per second. GPUWattch provided a convenient online scalable simulation platform, and was widely used in recent GPU DVFS studies.

---

Table 3 Summary of statistical GPU power modeling studies.

<table>
<thead>
<tr>
<th>Study Year</th>
<th>Device</th>
<th>Method</th>
<th>Input variables</th>
<th>Benchmarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>2009</td>
<td>NVIDIA 8800GT</td>
<td>SVR</td>
<td>5 busy signals</td>
<td>10 OpenGL benchmarks</td>
</tr>
<tr>
<td>2010</td>
<td>NVIDIA Tesla GTX285</td>
<td>SLR</td>
<td>13 CUDA performance counters</td>
<td>41 kernels in CUDA SDK and Rodinia</td>
</tr>
<tr>
<td>2011</td>
<td>AMD Radeon HD5870</td>
<td>RF</td>
<td>23 Stream Profiler counters</td>
<td>78 OpenCL kernels in ATI Stream SDK</td>
</tr>
<tr>
<td>2013</td>
<td>NVIDIA Fermi GTX480</td>
<td>SLR</td>
<td>12 CUDA performance counters</td>
<td>20 OpenCL applications in CUDA SDK and Rodinia</td>
</tr>
<tr>
<td>2013</td>
<td>A cluster with 4 NVIDIA Tesla M2050 cards</td>
<td>Transformed SLR, GAM</td>
<td>8 CUDA performance counters</td>
<td>33 kernels in CUDA SDK, Rodinia and NVIDIA CUDA Profiler</td>
</tr>
<tr>
<td>2013</td>
<td>NVIDIA Fermi C2075</td>
<td>BP-ANN</td>
<td>10 CUDA performance counters</td>
<td>20 kernels in CUDA SDK, Rodinia</td>
</tr>
<tr>
<td>2015</td>
<td>AMD Radeon HD7970</td>
<td>CodeXL</td>
<td>22 CodeXL performance counters</td>
<td>108 OpenCL kernels in Rodinia, Parboil, etc</td>
</tr>
</tbody>
</table>

X. Mei et al. Digital Communications and Networks 3 (2017) 89–100

---

Digital Communications and Networks 3 (2017) 89–100
Both Hong's and Leng's models had outstanding performance and were widely adopted by the following researchers [51,52,55,56]. On the other hand, some researchers also pointed out that the models were product-specific and it was difficult to tune the parameters when applying them on other GPUs [22]. Sen and Wood derived a simple power model that mainly relied on the processing time of each core [52]. Their model achieved high qualitative similarity with GPWWatch. In [27], Rhu et al. also stated that the power estimation by the simple IPC-based model [57] had more than 90% agreement with that by GPWWatch.

### 4.2. Statistical methods

Some researchers built statistical models for the GPU runtime power consumption. They used software to monitor the runtime signals of the GPU-accelerated applications, and fitted or trained the power model based on the observed signals. This approach treats the GPU micro-architecture as a black box, and seeks for relationships between the GPU runtime power consumption and micro-architecture events. We summarize the related studies in Table 3, including their target devices, statistical methods, studied benchmarks, etc.

In Table 3, SVR, SLR, RF and GAM stand for support vector regression, square linear regression, random forest and generalized additive models, respectively. These traditional statistical models fit the linear relationship tightly. They give the contribution of each input variable directly. Eq. (4) gives the general form of the traditional regression model, where $x_1, x_2, ..., x_n$ are the $n$ input variables, $P$ is the power consumption, and $a_0, a_1, ..., a_n$ are the output contributions. The mathematical representation is similar with that of the empirical methods.

$$P = a_0 + a_1x_1 + a_2x_2 + ... + a_nx_n$$

(4)

Some modern techniques such as Artificial Neural Networks (ANN) and K-means clustering are also applied in the literature. Fig. 3 demonstrates an ANN, where $x_1, x_2, ..., x_n$ denote the $n$ input variables and $P$ denotes the power consumption. Every arrow in the figure represents a model parameter. The researchers configure the ANN structure and train it with a set of training data, and after the training the system will obtain all model parameters that can achieve a certain level of prediction accuracy. Compared to the traditional models, the neural network approach addresses the nonlinear dependency of the input variables.

Ma et al. applied the supported vector regression to build GPU power model based on five signals [58]. Notice that the software, variables and benchmarks in [58] were based on graphics applications, while others in Table 3 were on general-purpose GPU applications. In [37], Nagasaka et al. found that except the constant part (70% of contribution), the instruction count and the global memory accesses contribute to the GPU runtime power the most. In [61], Chen et al. simulated the runtime GPU characteristics in the cycle-level GPU simulator, GPGPUSim, which could decode the kernels to separate hardware instructions. Their random forest model suggested that the registers, single-precision floating-point, global memory, integer and arithmetic logic instructions were the most influential variables. Zhang et al. applied similar techniques to an AMD GPU [62]. Karami et al. measured the power consumption of a Fermi GPU with OpenCL applications. They used the principle component analysis to pick out only a part of the performance counters as the input [65]. Ghosh et al. extended the study to multi-GPU system [66]. They applied some nonlinear transformations on the collected instruction counts, such as logarithm, division, etc, and found that the transformed SLR worked better than the traditional SLR, which might suggest some nonlinear relationships between the power and the input variables. The above regression models all highlight the contribution of the computation instruction counts and the memory (especially register and global memory) instructions.

Abe et al. built DVFS regression models for the NVIDIA Tesla, Fermi and Kepler GPUs [22]. Particularly, they regarded the 3 different core/memory frequency settings as the model inputs. They also chose 10 most relevant performance counters who gave the best fitting results. The prediction error varied from 15% to 23.5%, depending on the generations of GPU, and the newer hardware had larger prediction error.

Song et al. trained the GPU runtime power with an ANN of two hidden layers [67]. Their model achieved better prediction accuracy than that of SLR in [37]. Wu et al. extensively studied the GPU power and performance with different settings of GPU frequency, memory bandwidth and core number [68]. They applied K-means clustering and ANN. In the ANN modeling process, they first used K-means to cluster the set of kernels into groups with similar scaling behaviours. Then for each group, they trained an ANN with two hidden layers. The average power prediction error over all frequency/unit configurations was 10%.

In general, the traditional regression based methods are easier to implement, but they fail to capture the nonlinearity. For the recent generations of GPUs, the prediction errors of the regression models tend to be large. On the contrary, the advanced neural network approaches suit the complicated data dependencies better, but require a great larger amount of training data, and the output models are of high complexity. For the power modeling work with frequency scaling, the prediction accuracy is relatively low, which might call for more effective modeling methods.

### 5. GPU DVFS performance modeling

In this section, we introduce the GPU performance modeling studies, where a number of them consider the GPU frequency scaling. We classify them into two categories: pipeline analysis and statistical methods. The pipeline analysis is a bottom-up empirical method which requires the knowledge of GPU execution principles, while the statistical methods purely rely on the GPU performance counters.

#### 5.1. Pipeline analysis

Many GPU performance modeling studies were based on the GPU pipeline analysis [35,36,55,56,67,70]. They assembled the GPU execution pipeline and analyzed the memory/computation parallelism. We list some popular metrics used to evaluate the pipeline parallelism as below:

- **MWP** (memory warp parallelism [55,70]): the maximum number of warps that can access the memory simultaneously on one SM during the *memory waiting period*, i.e., the period bet-
The demonstrated pipeline is stalled mainly by memory loads. The latency of all memory instructions and the overlapped computation instruction forms the load critical path [56].

We also give an example of the GPU pipeline in Fig. 4. The demonstrated pipeline is stalled due to the limited MWP. In real applications, the pipeline involves more types of instructions and various pipeline stalls.

Hong et al. used MWP and CWP to approximate the GPU execution pipeline in [70]. They computed MWP and CWP according to the global memory latency, memory bandwidth, the warp numbers, etc. They then divided the pipeline status into three categories: CWP>MWP, MWP>CWP and CWP=MWP (caused by insufficiency of concurrent warps). For each category, they derived the rough total execution cycles. In [36], the authors refined the model by considering the cache access, shared memory bank conflict and other related issues. Their model was widely adopted and extended by the literature.

Chen et al. presented a much simpler MWP computation method in [55], based on the average memory access latency, which considered both cache hit and cache miss cases. The parameters of their model were obtained by the PTX code analysis in GPGPUSim.

Song et al. proposed a comprehensive pipeline analysis by assembling the full execution process in [67]. They drew the complete execution pipeline for their 12 tested GPU kernels. Their average prediction error rate was as low as 6.7%. However, for this method, the prediction error was at the cost of being very application-specific and hardware-specific.

Baghsorkhi et al. built a performance model based on the GPU work flow graph, which was a graphic abstraction of the GPU execution pipeline [71]. They estimated the GPU execution time by calculating the total weight of the work flow graph. In their model, the memory latency was alterable, according to different warp executing patterns. The advantage of this model is that it could predict the execution time of diverse warp scheduling patterns in one run.

Nath et al. built a GPU performance model considering the core frequency scaling [56]. They divided the whole GPU executing pipeline into portions either sensitive or insensitive to GPU core frequency scaling, and studied how the sensitive portion changed to frequency. This model achieved impressive high accuracy for all the frequency settings. In addition, it unambiguously highlighted the nonlinear effect of GPU frequency on performance. They also proposed a simplified model by approximating LCP length with memory load stall cycles, and the simplified model showed competitive prediction accuracy.

5.2. Statistical methods

Abe et al. built statistical linear regression performance models with respect to the core and memory frequency scaling, on four NVIDIA GPUs across the Tesla, Fermi and Kepler platforms [22]. They chose variables from the CUDA performance counters, just as they did for the power modeling. However, their average performance prediction errors were large, varying from 33.5% to 67.9% on different generations of GPUs. This may be due to a lack of data sampling, that they only performed the experiments with 3 different core/memory frequencies.

Wu et al. trained a performance model for an AMD GPU, with respect to varying both the core and memory frequency [68]. They used K-means clustering and the ANN modeling and received an average of 15% of performance prediction error across the frequency ranges. So far as we know, this is the only statistical GPU performance modeling involving advanced ANN techniques.

Ardalani et al. also used machine learning to train GPU performance models [72]. Their modeling included two techniques: the forward feature selection stepwise regression and the bootstrap aggregating. Different from the linear regression, their regression automatically applied certain transformations on the input variables, so that the output model could capture some nonlinearity. The authors trained the model with a Maxwell GPU, receiving 27% of prediction error. Then they validated the model with a Kepler GPU, and the prediction error only increased a bit, to 36%. This up-to-date model showed some robustness across different generations of GPU platforms.

In Table 4, we summarize the formulas to describe the impact of frequency scaling on the GPU execution time in the literature. \( f^{C_G} \) and \( f^{M_G} \) denote the GPU core frequency and memory frequency, respectively. \( t \) denotes the execution time, and \( a_1,...,a_9 \) denote the coefficients defined by both the hardware and the application characteristics. In [22], the authors modeled the execution time as the summation of three parts: a static part \( (a_1) \) which is insensitive to frequency scaling, a dynamic part \( (a_2)f^{C_G} \) that is sensitive to GPU core frequency only, and another dynamic part \( (a_3)f^{M_G} \) that is sensitive to GPU memory frequency only. Nath et al. considered the GPU core frequency scaling only. They divided the execution time into two segments: the LCP and

![Fig. 4. An example of GPU execution pipeline, where we define MWP=2, C1 and C2 denote two different compute instructions, and M1 denotes a memory instruction. The instructions are launched at every clock cycle. Assume this pipeline is taken from the 'reduction' kernel, thus the compute instruction C2 depends on the data returned by not only the current warp but also its subsequent warp. We assume a computation instruction takes 2 cycles, and the memory waiting period is 6 cycles. CWP equals the memory waiting period over the length of a computation instruction and then plus 1 [35,70], i.e., CWP=6/2+1=4. The demonstrated pipeline is stalled mainly by memory loads. The latency of all memory instructions and the overlapped computation instruction forms the load critical path [56].](image)
the Compute/Store Path (CSP) [56]. For each segment, there are a static part \((a_1, a_2)\) and a dynamic part \((a_2, f_{Gc}, a_2/f_{Gm})\), where the two parts may be overlapped. When \(f_{Gc}\) varies, the length of each segment equals the larger one. This model stresses the nonlinear relationship between \(t\) and \(1/f_{Gc}\). In [68], the authors modeled the DVFS performance according to ANN, in which \(f_{Gc}\) and \(f_{Gm}\) are regarded as ANN inputs.

The listed mathematical forms in turn support the diverse DVFS effects for different GPU applications. Among the models, [56] depends on the pipeline analysis and the other two depend on statistical methods. [56] shows the best accuracy, yet it considers the core frequency scaling only. The other two models consider both the core and memory frequency scaling, but the overall prediction accuracy is still low. Even the advanced ANN technique does not improve much accuracy.

6. GPU voltage and frequency scaling effects

We present our measurement DVFS study in this section. We scale the GPU core voltage/frequency and memory frequency of the Maxwell GTX980. We also scale the core voltage, core frequency and the memory frequency of the Fermi GTX560Ti. For simplicity, we use their codenames to refer to these two GPUs.

6.1. Experimental methodology

In our previous work [38], we introduce the methodology to scale the core and memory voltage/frequency of the Fermi GPU, with the help of a series of third-party software tools. For the Maxwell platform, we use the NVIDIA Inspector [41] to scale the core/memory frequency. In addition, we disable GPU Boost to fix the GPU core/memory frequency at the selected level. The NVIDIA Inspector reports the power data every second. Since a GPU kernel may take less than one second to finish, we run sufficient rounds of the same kernel to guarantee a total running time of at least 20 min for each kernel which results in more than 1200 power samples. To verify the repeatability of our experiments, we also conduct significance test with t-distribution on the power samples of each kernel and achieve 95% confidence interval. The energy consumption is then calculated as the average power multiplied by the total running time.

On the Maxwell platform, we choose P2 state which allows us to reliably scale the core voltage in range [0.987 V, 1.087 V]. According to Eq. (2), for an application with different voltage settings, higher voltage usually results in more energy consumption. We verify this in Fig. 5. For the six kernels taken from Rodinia benchmark suite, energy consumptions of the higher core voltage are always more. In the following, we fix the core voltage at the lower bound 0.987 V and scale the core frequency only. We use the Maxwell platform as an example to investigate the impact of Dynamic Frequency Scaling (DFS) on energy consumption.

We denote the GPU core voltage, core frequency and memory frequency as \(V_{Gc}, f_{Gc}\) and \(f_{Gm}\), respectively. Table 5 lists the target DFS/DVFS settings of our two GPU platforms. Similar with that in [38], we investigate the core scaling effect and memory scaling effect separately. Namely, when we change the GPU core settings, we fix the memory settings and vice versa. In total, we study 7 core and 6 memory settings for the Maxwell DFS, and 9 core and 5 memory settings for the Fermi DVFS. We also list the default core and memory settings in Table 5. For the Maxwell platform, we use the same default setting for the core and memory scaling; while for Fermi, the default settings of the core and memory scaling are different.

We denote the default energy consumption as \(\bar{E}\), and the minimum and maximum energy consumption under different voltage/frequency settings as \(E_{min}\) and \(E_{max}\), respectively. We use two metrics to evaluate the DVFS effect: \(\bar{R}\) and \(R_{max}\), where \(\bar{R}\) quantizes how much energy could be saved compared to default configuration, and \(R_{max}\) indicates the maximum energy saving capability. We give their definitions in Eqs. (5) and (6). 

\[
\bar{R} = 1 - \frac{E_{min}}{\bar{E}}
\]

\[
R_{max} = 1 - \frac{E_{min}}{E_{max}}
\]

It is noticeable that for the Fermi platform, we measure the power consumption of the whole desktop using an off-board power meter, so that \(\bar{R}\) and \(R_{max}\) for the Fermi refer to the system level energy saving, including the energy savings of the CPU, the interconnect, the motherboard, etc. That is because for the early GPUs, there are no power manage interfaces and it is difficult to get the GPU power consumption directly. On the other hand, the \(\bar{R}\) and \(R_{max}\) for the Maxwell refer to the GPU energy savings only, benefitting from the on-chip power sensors on the Maxwell GPU product. It is possible to know the GPU runtime power without using a meter.

We study the voltage/frequency scaling effect on two platforms with the same suite of 37 applications, taken from Rodinia and CUDA SDK. Note that there is also a little difference in the two sets of experiments. For the Fermi experiments, the studied applications are based on CUDA SDK 4.1; but for the Maxwell, the applications are based on CUDA SDK 6.5.

6.2. Experimental results

6.2.1. Core voltage-frequency relationship

Fig. 6 shows the relationship between the maximum allowed core frequency and the core voltage on the Maxwell platform. It is widely believed that the maximum core frequency increases linearly to the core voltage. In [38], we find that the relationship between the maximum allowed core frequency and the core voltage is sublinear on the Fermi platform. As shown in Fig. 6, we observe similar sublinear relationship on the Maxwell platform. The conservative default setting helps to protect the GPU board and also leaves some room for overclocking.

6.2.2. Maxwell DFS

We first present the experimental results of Maxwell core frequency scaling. Fig. 7 summarizes \(\bar{R}\) and \(R_{max}\) of all benchmark applications on the Maxwell platform. Note that for the applications with multiple kernels, we compute \(\bar{R}\) and \(R_{max}\) for each kernel. The average \(\bar{R}\) is 5.24%, and the average \(R_{max}\) is 10.87%, at single GPU level. Applications saving the most energy include \(\text{eigenvalues}, \text{gaussian}, \text{hotspot}, mn\), etc. In the best case (\(mn\)), up to 34% GPU energy can be saved.

We also show the best core frequency, i.e., the frequency that leads to the minimum energy consumption of all the tested samples, for each kernel in Fig. 7. Among all the kernels, 12 benefit from scaling up the core frequency \((f_{Gc} > 950\) MHz\) while the other 30 benefit from scaling down the core frequency \((f_{Gc} < 950\) MHz\). In particular, half of the kernels achieve their minimum energy at core frequencies between 680 MHz and 880 MHz, where 11 of them at 780 MHz. In [22], the authors state that low or medium core frequency setting improves the energy efficiency obviously on the Kepler GPU platform, where their medium setting corresponds to 680–880 MHz on our Maxwell platform. Therefore for modern GPUs, scaling down the core frequency to some extent is an effective approach to conserving energy, even if it is difficult to scale down the core voltage. Besides, it also reveals that for many applications, the performance is nonlinear to the GPU core frequency.

We plot the normalized energy under different core frequency settings of some kernels in Fig. 8. The kernels exhibit diverse energy consumption behaviours. The energy consumption of \(\text{vectorAddDrv}\) increases linearly to the core frequency. \(\text{MC:EstimatePilelneP}\) and \(\text{binomialOptions}\) are insensitive to core frequency scaling, while \(\text{eigenvalues}\) and \(mn\) have optimal frequency setting in a very narrow
The diverse behaviors coincide with those described in [38, 20, 51]; and it confirms the complexity of the relationship between the runtime energy and the processor frequency, which calls for a more in-depth investigation.

We demonstrate the memory frequency scaling effect in Fig. 9. The average $R_{\text{max}}$ is as high as 35.87% whereas the average $R$ is 0.72%. In this case, $R_{\text{max}}$ denotes the energy increase caused by scaling down the memory frequency. 24 kernels suffer from more than 30% of energy increase by simply scaling down 30% of memory frequency. In the worst case (bfs), up to 53.9% energy can be wasted. The major reason is that the execution time of these kernels will increase significantly when the memory frequency decreases.

In Fig. 9, 34 kernels observe their minimum energy consumption at the default setting by the vendor, which suggests that the default setting is optimal for most cases. Note that even for the special applications which do not benefit from the default memory frequency, such as matrixMulDrv and vectorAddDrv, the difference is no more than 6%. A few kernels benefit from a little higher frequency of 3300 MHz, such as scalarProb and sortingNetworks. It is because the energy saving brought by the reduced running time is more than the extra energy by the higher memory frequency.

### 6.2.3. Fermi DVFS

We demonstrate the Fermi core voltage and frequency scaling effect in Fig. 10. Note that each time we change the voltage and frequency together. By scaling down both the core voltage and core frequency, DVFS can reduce a considerable percent of system energy. The energy savings on our Fermi platform at whole system level are 18.91% on average and 24.4% at maximum. In general, the energy savings are larger than those of Maxwell platform, which stresses the importance of marginally scaling down the core voltage for energy conservation.

---

### Table 5
Target GPU voltage/frequency configurations.

<table>
<thead>
<tr>
<th>Platform</th>
<th>Maxwell DFS</th>
<th>Fermi DVFS</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>white body</strong></td>
<td>([V_{Gc}, f_{Gc}])</td>
<td>([V_{Gc}, f_{Gc}])</td>
</tr>
<tr>
<td></td>
<td>([0.987 \text{ V}, 480 \text{ MHz}])</td>
<td>([0.849 \text{ V}, 880 \text{ MHz}])</td>
</tr>
<tr>
<td></td>
<td>([0.987 \text{ V}, 580 \text{ MHz}])</td>
<td>([0.857 \text{ V}, 896 \text{ MHz}])</td>
</tr>
<tr>
<td></td>
<td>([0.987 \text{ V}, 680 \text{ MHz}])</td>
<td>([0.881 \text{ V}, 932 \text{ MHz}])</td>
</tr>
<tr>
<td></td>
<td>([0.987 \text{ V}, 780 \text{ MHz}])</td>
<td>([0.912 \text{ V}, 952 \text{ MHz}])</td>
</tr>
<tr>
<td></td>
<td>([0.987 \text{ V}, 880 \text{ MHz}])</td>
<td>([0.949 \text{ V}, 972 \text{ MHz}])</td>
</tr>
<tr>
<td></td>
<td>([0.987 \text{ V}, 980 \text{ MHz}])</td>
<td>([0.981 \text{ V}, 982 \text{ MHz}])</td>
</tr>
<tr>
<td></td>
<td>([0.987 \text{ V}, 1080 \text{ MHz}])</td>
<td>([1.012 \text{ V}, 990 \text{ MHz}])</td>
</tr>
<tr>
<td></td>
<td>(f_{Gm}=3000 \text{ MHz})</td>
<td>(f_{Gm}=2100 \text{ MHz})</td>
</tr>
<tr>
<td><strong>Default setting</strong></td>
<td>([V_{Gc}, f_{Gc}])</td>
<td>([V_{Gc}, f_{Gc}])</td>
</tr>
<tr>
<td></td>
<td>([0.987 \text{ V}, 950 \text{ MHz}])</td>
<td>([1.049 \text{ V}, 995 \text{ MHz}])</td>
</tr>
<tr>
<td></td>
<td>(f_{Gm}=3000 \text{ MHz})</td>
<td>(f_{Gm}=2100 \text{ MHz})</td>
</tr>
<tr>
<td><strong>white body</strong></td>
<td>([V_{Gc}, f_{Gc}])</td>
<td>([V_{Gc}, f_{Gc}])</td>
</tr>
<tr>
<td></td>
<td>([2100, 2400, 2700, 3000, 3300, 3600] \text{ MHz}; [V_{Gc}, f_{Gc}])</td>
<td>([1500, 1700, 1900, 2100, 2300] \text{ MHz}; [V_{Gc}, f_{Gc}])</td>
</tr>
<tr>
<td></td>
<td>([0.987 \text{ V}, 980 \text{ MHz}])</td>
<td>([1.049 \text{ V}, 990 \text{ MHz}])</td>
</tr>
<tr>
<td></td>
<td>(f_{Gm}=3000 \text{ MHz})</td>
<td>(f_{Gm}=2100 \text{ MHz})</td>
</tr>
<tr>
<td><strong>Default setting</strong></td>
<td>([V_{Gc}, f_{Gc}])</td>
<td>([V_{Gc}, f_{Gc}])</td>
</tr>
<tr>
<td></td>
<td>([0.987 \text{ V}, 950 \text{ MHz}])</td>
<td>([1.049 \text{ V}, 990 \text{ MHz}])</td>
</tr>
</tbody>
</table>

---

Fig. 5. The effect of scaling the core voltage alone on the Maxwell platform.

Fig. 6. The relationship between the maximum allowed core frequency and the core voltage on the Maxwell platform.
Almost all the applications get the minimum energy consumption at the lowest voltage/frequency. Fig. 11 shows the memory frequency scaling effect on the Fermi platform. The average $R_{\text{max}}$ is 10.2% and average $\bar{R}$ is 3.5%. The energy saving is low, and the default memory frequency works well for many applications. The best memory frequencies for different applications are diverse, depending on the application characteristics, which strongly differ from that of the Maxwell platform. There is no linear relationship between the energy consumption/performance and the memory frequency for the Fermi platform.

To summarize, our experimental results on both platforms suggest the following interesting findings:

1. Appropriate core frequency setting is effective for energy saving.
Both platforms expose the “pacing [21]” feature. The relationship between the performance and the GPU core frequency is very complex and a simple linear model is inadequate;

2. In terms of memory frequency scaling, the early platform exposes the “pacing” feature, while the modern platform exposes the “racing [21]” feature. The performance is highly linear to the GPU memory frequency on our Maxwell platform.

7. Conclusions and future work

In this paper, we survey the GPU DVFS for energy conservation. We focus on the most up-to-date GPU DVFS technologies and their influence on the performance and power consumption. We summarize the methodology and the performance of existing GPU DVFS models. Generally speaking, the nonlinear modeling technique, such as the ANN and the transformed SLR, has better estimation accuracy.

In addition, we conduct real-world DFS/DVFS measurement experiments on the NVIDIA Fermi and Maxwell GPUs. The experimental result suggests that both the core and memory frequency influence the energy consumption significantly. Using the highest memory frequency would always conserve energy for the Maxwell GPU, which is not the case on the Fermi platform. According to the Fermi DVFS experiments, scaling down the core voltage is vital to conserve energy.

Both the survey and the measurements spotlight the challenge of building an accurate DVFS performance model, and furthermore, applying appropriate voltage/frequency settings to conserve energy. We leave these for our future study. Besides, it is another important direction to integrate the GPU DVFS into the large-scale cluster-level power management in the future. It will be interesting to explore how to effectively combine GPU DVFS with other energy conservation techniques such as task scheduling [73], VM consolidation [74], power-performance arbitrating [75], and runtime power monitoring [76–78].

Acknowledgements

This work is partially supported by HKBU FRG2/14-15/059 and Shenzhen Basic Research Grant SCI-2015-SZTIC-002.

References

[7] R. Raina, A. Madhavan, A.Y. Ng, Large-scale deep unsupervised learning using...


