Measuring the Energy Efficiency of Transactional Loads on GPGPU

Jóakim von Kistowski
joakim.kistowski@uni-wuerzburg.de
University of Würzburg
Germany

Johann Pais
Advanced Micro Devices
USA
johann.pais@amd.com

Tobias Wahl
tobias.wahl@stud-mail.
uni-wuerzburg.de
University of Würzburg, Germany

Klaus-Dieter Lange
Hewlett Packard Enterprise
USA
klaus.lange@hpe.com

Hansfried Block
SPECpower Committee
Germany
hansfried.block@web.de

John Beckett
Dell Inc.
USA
john_beckett@dell.com

Samuel Kounev
samuel.kounev@uni-wuerzburg.de
University of Würzburg
Germany

ABSTRACT

General Purpose Graphics Processing Units (GPGPUs) are becoming more and more common in current servers and data centers, which in turn consume a significant amount of electrical power. Measuring and benchmarking this power consumption is important as it helps with optimization and selection of these servers. However, benchmarking and comparing the energy efficiency of GPGPU workloads is challenging as standardized workloads are rare and standardized power and efficiency measurement methods and metrics do not exist. In addition, not all GPGPU systems run at maximum load all the time. Systems that are utilized in transactional, request driven workloads, for example, can run at lower utilization levels. Existing benchmarks for GPGPU systems primarily consider performance and are intended only to run at maximum load. They do not measure performance or energy efficiency at other loads. In turn, server energy-efficiency benchmarks that consider multiple load levels do not address GPGPUs.

This paper introduces a measurement methodology for servers with GPGPU accelerators that considers multiple load levels for transactional workloads. The methodology also addresses verifiability of results in order to achieve comparability of different device solutions. We analyze our methodology on three different systems with solutions from two different accelerator vendors. We investigate the efficacy of different methods of load levels scaling and our methodology's reproducibility. We show that the methodology is able to produce consistent and reproducible results with a maximum coefficient of variation of 1.4% regarding power consumption.

ACM ISBN 978-1-4503-6239-9/19/04…$15.00
https://doi.org/10.1145/3297663.3309667

CCS CONCEPTS

• **Hardware → Enterprise level and data centers power issues**; • **Computing methodologies** → *Graphics processors*; • **Software and its engineering** → *Software performance*;

KEYWORDS

Power, Energy Efficiency, GPGPU, Performance, Benchmarking, Measurement, Load Level, SPEC

ACM Reference Format:
Jóakim von Kistowski, Johann Pais, Tobias Wahl, Klaus-Dieter Lange, Hansfried Block, John Beckett, and Samuel Kounev. 2019. Measuring the Energy Efficiency of Transactional Loads on GPGPU. In *Tenth ACM/SPEC International Conference on Performance Engineering (ICPE '19), April 7–11, 2019, Mumbai, India.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3297663.3309667

1 INTRODUCTION

The energy efficiency of servers and data centers has become an ever more important issue over the last decades. In 2010, the U.S. Environmental Protection Agency (U.S. EPA) estimated that 3% of the entire energy consumption in the U.S. is caused by data center power draw [19]. According to a New York Times study from 2012, data centers worldwide consume about 30 billion watts per hour. This is equivalent to the approximate output of 30 nuclear power plants [5].

Improving the energy efficiency of data centers and servers requires the ability to measure and rate that efficiency. Methodologies, workloads, and metrics for server energy efficiency can enable more informed purchasing and provisioning decisions. This is the case for both regular servers and servers utilizing additional accelerator devices, such as General Purpose Graphics Processing Units (GPGPUs). GPGPUs have become more common over the last years and, consequently, the importance of rating their efficiency alongside the efficiency of other types of servers has increased.

GPGPUs are mostly used in High Performance Computing (HPC) environments, which are run mostly at full load. In contrast, most regular compute servers in modern day data centers are not being utilized to their full capacity. Instead, these compute servers are used to serve requests that arrive over time and are provisioned with additional capacity in order to be able to cope with variations in load, such as unexpected bursts. This leads to an average load somewhere between 10% and 50% [7]. With the proliferation of servers utilizing GPGPU devices, we pose that such servers will also see use in such request-based computing contexts in the future. In this case, benchmarking and rating methods for GPGPU servers must be adapted to consider such transactional loads.

Existing benchmarks for GPGPUs do not consider transactional loads or multiple load levels. Even energy efficiency itself is usually just an afterthought for these benchmarks. Benchmark suites, such as Rodinia [11], Parboil [27], LonestarGPU [10], and SHOC [13] focus on performance measurements that can be used to indicate HPC performance. All of these GPGPU benchmarks focus on determining the maximum performance level of the system under test (SUT) and its accelerators; they do not measure at lower load levels or utilization ranges.

This work presents a rating methodology for the energy efficiency of GPGPU servers that considers transactional loads. It is based on the *SPEC Power and Performance Benchmark Methodology* [26] and designed to execute GPGPU kernels at different request rates for a thorough energy-efficiency characterization. Our methodology considers different vendor-specific workload implementations, transaction scheduling, kernel parallelization, and GPGPU result verification. We present a single workload, implementing FFT, using the methodology, and showing how it can be used for transactional execution.

The major contributions of this paper are as follows:

(1) A measurement methodology for measuring the power consumption and energy efficiency of servers with GPGPUs running transactional loads at multiple load levels,
(2) A reference workload for the methodology, using FFT,
(3) A framework that allows for different workload implementations, allowing for the use of GPGPU vendor-specific technologies, and
(4) An analysis of GPGPU power and energy-efficiency scaling over transaction rate and size of processed data.

We evaluate our methodology using the reference FFT workload on both AMD and NVIDIA devices. We investigate the reproducibility of results and analyze if power consumption and energy efficiency scale with transaction rates and/or processed dataset sizes. We show that the methodology is able to produce consistent and reproducible results with a maximum coefficient of variation of 1.4% regarding power consumption and a maximum coefficient of variation of 4.4% regarding performance.

The rest of this article is structured as follows: Section 2 details the underlying general power rating methodology employed in this work and Section 3 covers related work. Section 4 then introduces the GPGPU benchmarking methodology and the reference workload. The methodology is evaluated in Section 5, which also provides an analysis on GPGPU power and energy-efficiency scaling. Finally, Section 6 concludes the paper.

2 SPEC POWER METHODOLOGY

The work of this paper uses the *SPEC Power and Performance Benchmark Methodology* [26] as its basis. The methodology has been developed by the SPEC OSG Power Committee as a tool for the analysis and evaluation of the energy efficiency of server systems. It was first implemented in SPECpower_ssj2008 [17] and later in the SPEC SERT [19] and SPEC Chauffeur WDK [4]. SPECpower_ssj2008 is a benchmark which emulates a transactional enterprise application, whereas SERT does not execute an application from a specific domain. It does not aim to emulate real world end user workloads, but instead provides a set of focused synthetic micro-workloads called *worklets* that exercise selected aspects of the server (or system) under test (SUT). The worklets have been developed to exercise the processor, memory, and storage I/O subsystems. The SPEC Chauffeur WDK is a development kit for such workloads and can be used by researchers and industry practitioners for creation of their own specialized worklets.

2.1 Device Setup

The SUT is at the center of the power measurement setup. It is a physical system that runs the workloads used for evaluation. The SUT's power consumption and its performance during testing are used to derive the energy-efficiency score. Performance metrics are gathered from the SUT using a testing software harness. We refer to the actual test execution software on the SUT as the *host* software. For CPU-bound workloads, the host in turn spawns separate on-SUT processes, referred to as *clients*, for each logical CPU (hardware thread). For most workloads, a transactional workload is executed on the clients. Clients can run work units in parallel. However, parallelism also can be achieved by running multiple clients concurrently. The clients collect the performance metrics of their workload and forward this information to the host.

The workload is controlled by the *controller* system. It coordinates which workload to run at which load level. It also collects all measurements both from the SUT as well as external measurement devices, and it calculates the metrics and scores. We refer to the collection of software managing all instances and measurement devices as the *director*. They are illustrated in Fig. 1.

We require at least one external power analyzer and one temperature sensor. The power analyzer measures the power consumption of the entire SUT, whereas the temperature sensor verifies the validity of measurements by making sure that all experiments are conducted under similar environmental conditions. External power and temperature instrumentation are used, as opposed to potential internal instrumentation, as the methodology makes no assumptions about the internal structure of the SUT, allowing for maximum portability. Reliance on external power measurement devices also enables the definition of tight constraints on the accuracy of the power measurement devices. Specifically, power measurement devices must feature a maximum measurement uncertainty of 1% or better.

2.2 Load Levels

According to [7], servers nowadays spend most of their time in a CPU utilization range between 10% and 50%. As a result, workloads within the *SPEC Power and Performance Benchmark Methodology* are

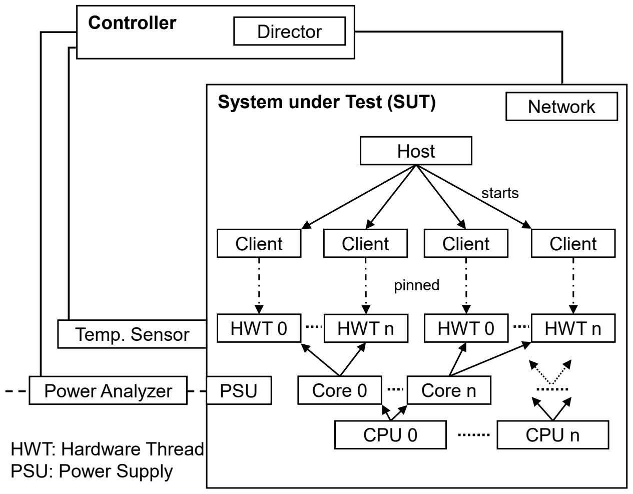

Figure 1: Typical server power measurement device setup.

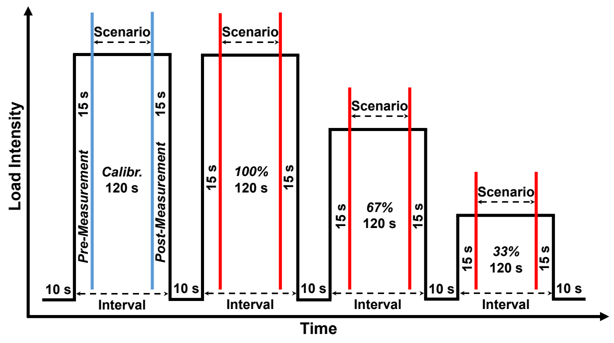

Figure 2: Intervals for calibration and measurement phase [29].

designed for the measurement of system energy efficiency at multiple load levels. This sets benchmarks implementing the methodology apart from conventional performance benchmarks, such as SPEC CPU [9], or other energy-efficiency benchmarks, such as JouleSort [25], and the TPC-Energy benchmarks [24], which target maximum load and performance. To achieve workload execution at different load levels, a methodology-compliant benchmark calibrates the load by determining the maximum transaction rate for the given workload on the SUT. The maximum transaction rate is measured by running as many of the worklet's transactions as possible, concurrently on each client. This calibrated rate is then set as the 100% load level for all consecutive runs. For each target load level (e.g., 100%, 75%, 50%, 25%), the benchmark calculates the target transaction rate and derives the corresponding mean time from the start of one transaction to the start of the next transaction. During the measurement interval, these delays are randomized using an exponential distribution that statistically converges to the desired transaction rate. As a result, lower target loads consist of short bursts of activity separated by periods of inactivity. Fig. 2 shows how calibration and the following measurement intervals would run using intervals at 100%, 67%, and 33% as an example. Note that the load levels are throughput-based and do not indicate CPU utilization (this is a common misconception).

2.3 Transactions and Users

The implementations of the SPEC power methodology have developed a common way to handle input data and transaction state. This method is relevant for this work, as it must be extended to work with external accelerators. In addition, we must consider and handle the state of accelerator execution contexts. In general, the methodology considers any transaction to be stateless. Therefore, states are stored separately from transactions. To achieve this, the methodology implementations consider two types of objects: *Transactions* and *Users*. *Transactions* are the stateless representation of a transaction's operations. *Users*, on the other hand, are passive resources that are allocated by transactions as they begin the active execution of an individual transaction. *Users* are again released after a transaction's execution has finished. This way, the *Users* can be used to store any data needed for the transaction's execution, including the execution context itself. Note that sufficient *Users* must be available for transactions to utilize all tested devices (e.g., CPU cores or GPGPUs).

3 RELATED WORK

We categorize related work into three major non-exclusive categories: GPGPU benchmarks, works on GPGPU energy efficiency, and server energy-efficiency benchmarks in general.

GPGPU benchmarks are mostly performance benchmarks. They are used to measure performance of GPUs in common GPGPU application scenarios. For example, NAMD [8], LAMMPS [23], and GROMACS [22] implement algorithms used in scientific computing for various fields of study and are used to measure the performance of accelerators in HPC environments. Similarly, the Rodinia [11], Parboil [27], LonestarGPU [10] and SHOC [13] suites focus on performance measurements that can be used to indicate HPC performance. All of these GPGPU benchmarks focus on determining the maximum performance level of the SUT and its accelerators; they do not measure at lower load levels or utilization ranges.

Works on GPGPU energy efficiency often employ these GPGPU benchmarks for analysis of energy efficiency at maximum load. [12] performs such an analysis to compare the most common GPGPU benchmark suites. Similarly, [14] analyzes the energy efficiency of multi-GPU systems. Expanding on these analyses, works such as [21], [15], and [31] are able to predict the power consumption of GPGPU devices. Expanding on these prediction mechanisms, contributions, such as [3] and [16], optimize energy efficiency of GPGPUs. Notably, both these works attempt to increase efficiency by avoiding low load conditions. [3] collects instructions of the same type to execute them in a way that prevents low load, whereas [16] coexecutes different computing kernels in order to better utilize the GPU's computing hardware at high load. However, we argue that low load is not completely avoidable, especially in transactional execution scenarios, and must thus be examined by benchmarking and testing solutions.

Server energy-efficiency benchmarks measure and rate the energy efficiency of servers. In general, server-efficiency benchmarks run a stable load for a period of time and then compute energy efficiency as a function of the benchmark's performance and power consumption measured during the benchmark run. JouleSort [25] is one of the earliest benchmarks of this type, always running at maximum speed. Similarly, TPC explicitly allows for energy measurement during the execution of all of their current benchmarks [24]. These benchmarks are designed for database performance testing and often are executed in highly distributed systems. In contrast to JouleSort and the TPC benchmarks, SPECpower_ssj2008 [17], a benchmark that emulates a transactional business application, features ten different load levels that scale with the application's throughput. This idea is expanded on in the SPEC SERT [19], which runs multiple different mini-workloads (called worklets) and uses SPECpower_ssj load level scaling for its CPU [30] and Storage [20] worklets. Notably, SERT uses a different method of load level scaling for its memory worklets [18], as the standard load level scaling method is difficult to apply in this context.

Our work in this paper differs from the related work in several aspects: We differ from existing GPGPU benchmarks and energy-efficiency work by considering transaction workloads which enable the investigation of energy efficiency at low load scenarios. With this, we resemble classical server energy-efficiency benchmarks, which have not yet addressed GPGPUs. However, the related work on the memory worklets in the SPEC SERT [18] shows that the standard load level scaling mechanism is not equally applicable to all device types. Investigating if and how it can be used for GPGPUs is one of the major contributions of this paper.

4 GPGPU ENERGY-EFFICIENCY BENCHMARKING METHODOLOGY

The goal of the GPGPU energy-efficiency benchmarking methodology is to run transactional workloads on GPGPU devices in order to rate their energy efficiency. To this end, we structure the execution of our GPGPU computing kernels in a way that allows us to execute the kernel multiple times on the GPU. The transaction rate is then the number of times the kernel was executed, allowing for transaction rate-based load level scaling. We must also consider verifiability [28] of the work that is executed on the separate GPGPU device. The methodology should enable testing of different device types. However, in practice, vendors use specific implementation languages and technologies for their accelerator devices. Our methodology and its implementation must thus allow for the execution of these different vendor-specific kernel implementations.

4.1 Methodology Building Blocks

Our GPGPU benchmarking methodology is based on the *SPEC Power and Performance Benchmark Methodology* [26] and the Chauffeur WDK [4] implementation of the methodology. Within this framework, we place a translation layer that passes work from the workload transaction within Chauffeur to the GPU. This kernel harness interface also must be able to handle multiple execution contexts and *User* objects. Finally, we must consider the different, sometimes vendor-specific implementation options for our kernels.

4.1.1 SPEC Chauffeur WDK. The SPEC Chauffeur Worklet Development Kit (WDK) is a framework designed to enable easy development of energy-efficiency workloads to be executed and measured using the *SPEC Power and Performance Benchmark Methodology* [26]. Chauffeur thus handles instrumentation, result collection and reporting, timing of transaction executions, and transaction rate calibration. It is implemented in Java, allowing portability accross multiple server device types.

4.1.2 GPGPU Kernel. The GPGPU executes code packaged in so-called *kernels*. A kernel contains a detachable executable compiled before it is passed to the GPU for execution. Kernel code primarily differs from regular C or C++ in matters regarding program flow and memory management. Designed for parallelization, kernel developers must always consider synchronization. Each kernel instance has a unique Thread-ID and Block-ID, which it can use as part of its program flow in order to behave differently from other kernel instances. In our case, we use this feature as part of our FFT workload (see Section 4.5). A major potential bottleneck in kernel execution is the communication bus that is used for communication between CPU and GPU. Though the PCI-Express bus has enough power to deliver high data transfer rates, the internal memory of the GPU always is significantly faster. To avoid slowdowns, excessive communication between the GPU and CPU executed code should be avoided. This is especially challenging in our context, as we attempt to execute multiple kernel instances in relatively short time-spans, requiring communication between the CPU controlling the experiment run and the GPU performing the work. We partially address this challenge by running multiple parallel execution contexts on the CPU (and GPU), as described in Section 4.3.

OpenCL. OpenCL is used by GPGPUs and CPUs for parallel computation and is widely supported on every major accelerator platform. OpenCL is an open standard maintained by several major companies and standardized by the Khronos Group. OpenCL provides a rich API for memory management and transmissions and divides the memory into several address spaces: local, global, constant, and image memory. The memory to be used must be specified by the workload programmer. The workload in this paper uses the global, local, and constant memory. An OpenCL kernel is compiled during run-time, which would allow run-time source code modification by the surrounding code. For our benchmarking framework, this means that the surrounding GPGPU benchmarking code can be independent fully from the actual kernel code, as this code can be passed to it right before its first execution. This, in turn, allows for different OpenCL kernel implementations, which might be better suited for different accelerator devices, but still might all run within the same framework.

CUDA. CUDA is a vendor-specific kernel implementation language developed by NVIDIA. It is a proprietary language used by NVIDIA and only compatible with its GPUs. However, this technology has become very popular due to relatively simple usage and many libraries available for, among other things, deep learning, FFT, graph search, and physics simulation. CUDA uses a special compiler, called NVCC, for its kernels. As a result, and in contrast to OpenCL, CUDA kernels have to be compiled ahead-of-time by the developer, offering less flexibility in adjusting the code before

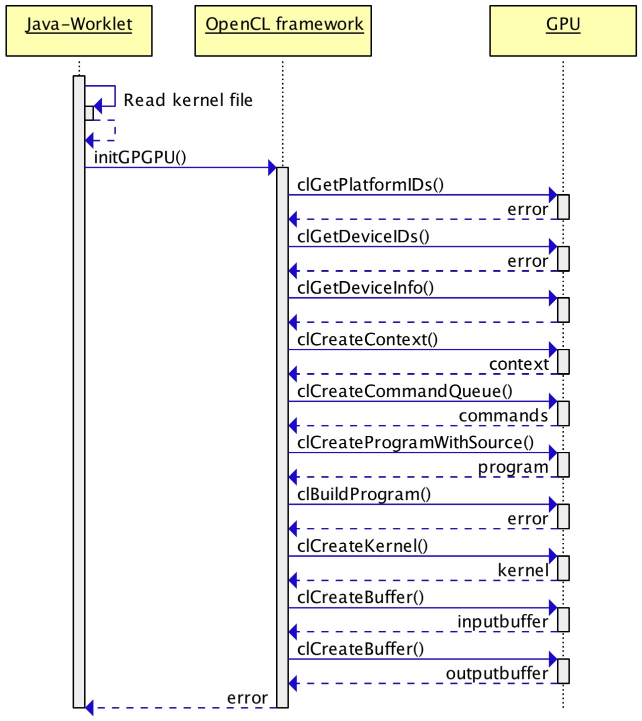

Figure 3: Initialization of OpenCL benchmark interval.

execution. However, many of the CUDA libraries, such as the FFT library used by our workload in Section 4.5, are provided by the driver and not compiled into the developer's code. This allows the driver to optimize the library for any specific device types, which contrasts with the developer optimizing the code in OpenCL. This driver-level optimization is challenging for benchmarkers, as they may potentially lose control over parts of the workload to the hardware vendor.

4.1.3 Kernel-Harness Interface. Every command for the GPGPU coming from the Chauffeur Worklet has to pass through our kernel harness interface. In general, this part is an interface between Java code running within Chauffeur and a native library written in C++. From a technical perspective, the kernel-harness interface utilizes the Java Native Interface (JNI) [2]. It converts input data passed from Chauffeur's *Users* into datatypes to be passed to the GPU. It also must manage device-specific initialization and cleanup routines. Generally, the initialization and the transaction execution are generalized to be able to work with all available datatypes. However, the pre-processing and result-reading methods depend on the specific data types utilized by the workload. Consequently, the framework specifies separate pre-processing and result-reading methods for each supported datatype. This is necessary to avoid type conflicts during conversion of Java-types to C++-types, and then to OpenCL/CUDA-types and vice versa.

4.2 Interval Benchmark Execution Flow

We consider the execution of each benchmarking interval (i.e., the execution of the benchmark at a certain load level) separately. For each interval, we reinitialize the execution contexts for all considered devices at the start of each interval. As a result, an interval's benchmark execution flow is composed of the following steps:

- **Initialization**: We initialize all kernel execution contexts and memory segments at the start of each interval.
- **Kernel execution**: After initialization, the actual benchmarking work is performed by executing the kernel multiple times (optionally in parallel) for the pre-measurement, measurement, and post-measurement phases of the interval.
- **Cleanup**: The execution contexts are reset by removing and cleaning allocated in-memory objects at the end of each interval.

4.2.1 Initialization. The initialization takes care of the correct initialization of the hardware and detection of the target platform. It usually initializes the GPU, but may also initialize an OpenCL context for the CPU. After determining the execution platform, and in case of OpenCL, the harness compiles the loaded OpenCL kernel code to create the executable kernel and then binds the kernel to the execution context, which describes the GPU's execution parameters. The execution context also uses several buffers that are created to hold the workload's data in memory. If any dynamic variables are used, they have to be stored in a special buffer to be accessible to the kernel. This includes any point of data to where the GPU is expected to write. We employ read-only buffers for larger arrays of

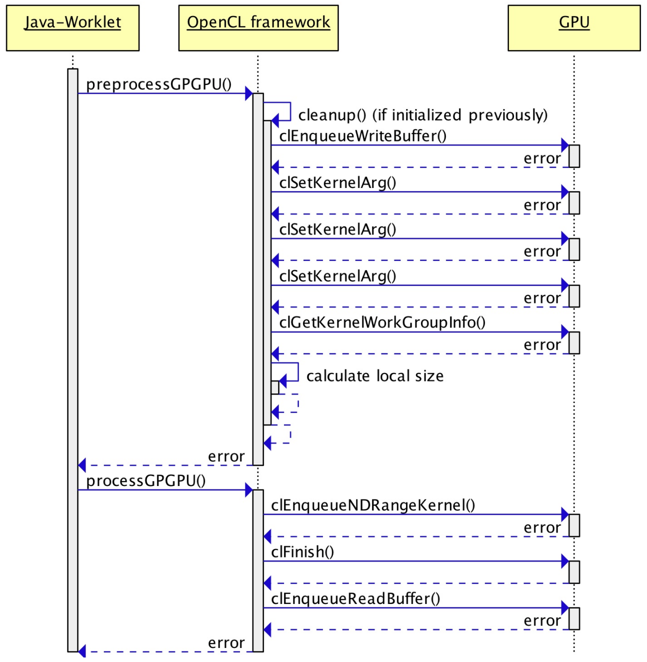

Figure 4: Kernel execution during an OpenCL benchmark interval.

data (e.g., large datasets for processing) only. Fig. 3 shows the steps taken during the initialization of a benchmark interval for OpenCL.

4.2.2 Kernel Execution. The execution itself is divided in two steps to accelerate the overall code execution by performing the expensive first initialization separately and running only a small part for re-initialization after each execution cycle. This reinitialization is necessary for the variables and buffers to be filled with the actual values and can be rerun after each execution without the need for the entire heavy-weight initialization to run. After data has been pushed to the GPU memory, the kernel gets enqueued in the execution queue of the GPU and begins processing the buffers and variables. After execution, the code pulls the output-buffer from GPU memory and stores the returned values in the host machine's memory. Fig. 4 shows the processing steps during a kernel's execution with OpenCL.

4.2.3 Cleanup. The cleanup deconstructs the context of an execution cycle by releasing all buffers and bindings. As a result, the GPU has to run a full initialization after cleanup if it is again to be prepared for execution. Fig. 5 shows the steps performed during cleanup with OpenCL.

4.3 Further Parallelization

In practice, we need the ability to schedule several transactions on the GPU devices in a parallel fashion. Running only a single transaction at a time results in too many downtimes to achieve high device utilization at full load, even though transactions themselves run a parallelized workload. To achieve this, the kernel and the kernel-harness interface need to be transaction-aware. For this purpose, they assign a transaction ID and allocate separate memory spaces to each transaction. This enables transaction-wise initialization, execution, and cleanup, which is necessary to maintain the character of a transaction when handling multiple transactions in parallel.

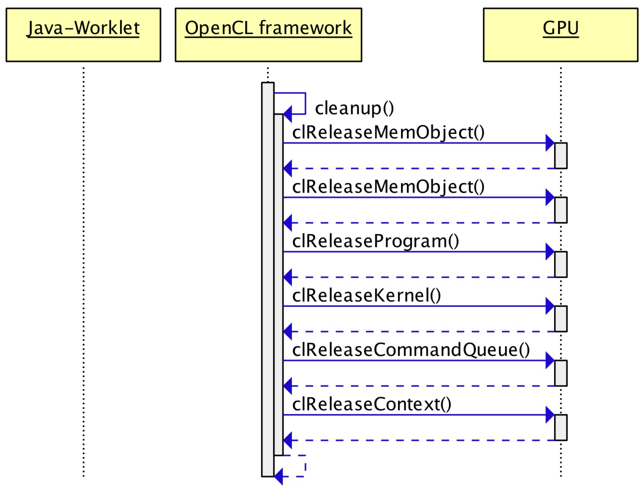

Figure 5: Cleanup after an OpenCL benchmark interval.

4.4 Result Verification

For verification, we create background tasks with low priority within the benchmark harness. These tasks run on the CPU and calculate a reference result for the corresponding dataset passed to the GPU. After calculation of the reference result the difference between the reference result and the GPU result is calculated and compared within a margin to compensate for precision errors. To minimize the impact on the GPU performance measurement the amount of verification tasks is limited and the results to verify are picked at random. As only a single failed verification is sufficient to invalidate the result, there is no need to check every result as long as the GPU cannot determine which result will be checked. Calculating the reference result on the harness side enables our methodology to be easily adopted for other future workload types or upcoming technologies, as the interfaces used for validation are mostly generic and independent of GPGPU technology and workload.

4.5 Workload: FFT

As an initial workload, we implement the FFT (Fast Fourier Transform) algorithm. FFT often is used in signal processing, encryption, and other application areas to map a discrete time signal to the frequency spectrum. Due to finite memory and step width, the Fourier Transform cannot be calculated continuously and has to be approximated by an algorithm such as Radix-2 or Radix-4. FFT has the benefit of being data independent between its internal steps and it is therefore highly parallelizable. In addition, we see a usage scenario for FFT in which it is run multiple times as part of serving requests or as part of other transactional workloads, which makes it a candidate for use in our approach to benchmark transaction GPGPU work units.

For CUDA, FFT already is implemented in a predefined library [1], which is used in our implementation to ensure consistent data and well-optimized code for the best possible performance. However, using this pre-implemented vendor-specific code runs the risk of decreasing comparability, as the internal workings of the in-driver library are unknown and may even perform power management operations. For this reason, we self-implement our OpenCL-kernel code. It is an adapted version of [6] with some major differences, such as the integrated conversion of one-dimensional values to vectors. As described in [6], using vectors instead of scalar values results in faster processing. Every FFT algorithm used in our work uses Radix-2, which enhances comparability of results. In general, we assume that the OpenCL kernel code features better comparability compared to the CUDA version, especially regarding the power consumption behavior it triggers, as it can run on all platforms and contains no hidden optimizations.

5 EVALUATION AND ANALYSIS

We evaluate our methodology regarding two major aspects: Reproducibility and power/efficiency scaling. The goal of this evaluation is to show that our methodology can not only produce reproducible results, but also help to show different power related behavior of servers with GPGPU devices. This would show the relevance of a transaction-based benchmarking approach for GPGPU. In addition, we perform an analysis on the factors that affect power and

Table 1: SUT specification.

	CPU-only	NVIDIA GPU	AMD GPU
Model	Fujitsu TX1320 M1	Dell R730	Reference Platform
Sockets	1	2	2
CPU	Intel Xeon	Intel Xeon	AMD EPYC
Model	E3-1281 v3	E5-2699 v4	7601
Cores	4	2 x 22	2 x 32
Threads	8	88	128
Base Clock	3.7 GHz GHz	2.2 GHz	2.2 GHz
Turbo/Boost	4.1 GHz GHz	3.6 GHz	3.2 GHz
Memory	32 GB	128 GB	128 GB
GPU	None	NVIDIA Tesla	AMD Radeon
Model		K40m	Instinct MI25
Memory		12 GB GDDR5	16 GB HBM2
OS	RHEL 6.7	RHEL 7.5	Ubuntu 18.04

efficiency scaling. In this analysis, we consider two major factors: *1)* Transaction rate and *2)* Dataset size. The transaction rate is the number of FFT transactions per second handled by the GPU, whereas the dataset size is the size of dataset to be analyzed by each single FFT transaction. We intend to see different types of power and efficiency scaling when increasing dataset size or transaction rate and also may see different types of power scaling depending on the GPU model.

We use three physical machines for our experiments. One machine is equipped with an NVIDIA Tesla K40m GPU, running CUDA version 9.2 (Update 1), one with an AMD Radeon Instinct MI25, utilizing OpenCL version 2.1. and finally, we use one machine to run CPU-only experiments as a base point of comparison for the power-scaling analysis. The SUT specifications are shown in Table 1. Each system is equipped with a dedicated power meter and the experiment is controlled by a separate controller machine.

5.1 Impact of CPU on GPGPU Result

First, we analyze the impact of our CPU configuration on the GPGPU results. This primarily addresses the number of CPU cores running kernel execution contexts. The goal is to analyze if the number of contexts placing load on the GPU impacts the GPU's performance and power consumption. We run our FFT workload with a dataset size of 2048.

Table 2 shows the throughput (transaction rate), average power consumption, and energy efficiency for the 100% load level, measured using the NVIDIA Tesla K40m GPU. For these measurements, four CPU contexts feature both the best performance and highest

Table 2: Power and Performance at 100% load depending on #CPU execution contexts on K40m GPU.

#CPU Contexts	Throughput	Power	Efficiency
2	15177.05 s^{-1}	318.9 W	47.59 J^{-1}
4	15916.62 s^{-1}	330.5 W	48.16 J^{-1}
8	15708.20 s^{-1}	341.7 W	45.97 J^{-1}
44	13872.09 s^{-1}	518.5 W	26.75 J^{-1}

Table 3: Power and Performance at 25% load depending on #CPU execution contexts on K40m GPU.

#CPU Contexts	Throughput	Power	Efficiency
2	3865.5 s^{-1}	260.0 W	14.9 J^{-1}
4	3967.2 s^{-1}	265.1 W	15.0 J^{-1}
8	3932.8 s^{-1}	265.8 W	14.8 J^{-1}
44	3463.1 s^{-1}	262.3 W	13.2 J^{-1}

energy efficiency. However, they do not feature the lowest power consumption. Running with fewer contexts results in a lower power consumption and performance. However the decrease in performance is greater than the decrease in power, resulting in an overall diminished energy efficiency.

Increasing the CPU context count from four to eight and beyond again results in a decreased performance; however, power consumption still increases. The performance decrease is very small when increasing the context count from four to eight results, as this increase results only in a performance drop of 1.31%. Increasing from four contexts to the total number of physical cores (44) results in a performance drop of 12.85%. The effect of increasing the number of contexts has a far greater impact on the power consumption, though. Increasing to eight contexts increases the power consumption by 3.37%, which is a greater relative increase than the performance from four to eight contexts. However, increasing the number of contexts to match the number of physical CPU cores has the greatest effect on power and results in an increase in power consumption of 56.87%. This significant increase in power consumption results in an energy-efficiency drop of 44.56%.

These effects can be explained by bus and/or GPU contention. Performance increases as long as additional CPU contexts are able to put more load onto the GPU. This comes at an increase in work performed on the CPU, which, in turn, increases power consumption. Once the bus/GPU is utilized at its maximum capacity, adding additional CPU contexts still increases the work performed by the CPU, thus increasing power consumption. However, it does not increase the work performed by the GPU; instead, it adds contention on both devices, reducing performance a bit. The additional work performed by the CPU results in a greater CPU load and fewer idle CPU cores, increasing overall power consumption. As a result, power can increase significantly, as shown in our experiments.

Table 3 supports this analysis. It shows the throughput, power consumption, and energy efficiency at the 25% transaction rate. When running at 25% of maximum load, the bottleneck dissipates. As a result, power consumption drops to similar levels for all CPU context counts, with power consumption of 44 contexts being even slightly lower than with four contexts. Of course, some performance differences persist as the 25% load level features 25% of the 100% load level performance by definition. This also results in slightly lower energy efficiency for the 44 context run.

We confirm our observation by repeating the experiments on the AMD system with its Radeon accelerator. Table 4 shows performance and power consumption normalized to the power and performance with a single CPU context. The results are very similar to those obtained using the NVIDIA GPU, with a small difference. On the AMD system, performance does not always degrade when increasing the number of contexts beyond four. Instead, increasing the number of contexts to 128 leads to the best performing result. However, as in the NVIDIA case, the power consumption increases significantly with additional CPU load. Again, the sweet spot seems to be with four CPU contexts. Running with four contexts increases performance to 271.9% of the single context performance at a power consumption of only 107.9% of this base setting.

From these CPU context experiments, we can conclude that the number of CPU contexts hits a sweet spot with the minimum number of parallel CPU contexts that are able to fully utilize the GPU. Increasing the number of contexts beyond this point may decrease performance by a small amount but will increase power consumption significantly. This effect can be observed with significant impact on CPUs with a high core count, as they have the greatest potential of overshooting the optimum CPU context setting.

5.2 Reproducibility

Next, we analyze the reproducibility of our methodology. To do so, we perform repeat measurements on each of our three SUTs. We analyze the coefficient of variation (CV) and the relative difference between the minimum and maximum measured value for both performance and power. We use measurements on our CPU-only system as the baseline in order to get an idea about the variation inherent to our workload.

Table 5 shows the CVs and relative min-max differences for OpenCL FFT running on our CPU-only system for 20 measurement re-runs. Package size is set to 64 and parallelization level is set to the number of logical CPU cores. Performance and power variations are very low. Performance has a maximum CV of 0.8% and a maximum relative min–max difference of 3.4%. As a point of comparison: the SPEC SERT [19] sets its maximum acceptable performance CV at 5%. Power consumption also varies little. FFT power consumption has a maximum CV of 1.1% and maximum relative min–max difference of 3.2%. This is less than half of the variation observed simply when keeping the system in an idle state. Overall, FFT seems to feature low variation running on a CPU.

Expanding on the CPU-only results, we analyze the performance and power variations when running our methodology and the FFT workload on GPUs. Table 6 shows the CVs and relative min–max differences for performance and power consumption on both the NVIDIA Tesla K40m and the AMD Radeon Instinct MI25 SUTs. These variations are generally comparable to the ones measured on the CPU. The NVIDIA system even features significantly lower performance variations than the CPU. It has a maximum CV of 0.2%, which also might be due partly to the CUDA FFT implementation.

Table 4: Normalized power and performance at 100% load depending on # of CPU execution contexts on Radeon GPU.

#CPU Contexts	Norm. Throughput	Norm. Power
1	100%	100%
2	183.5%	101.6%
4	271.9%	107.9%
64	265.5%	130.5%
128	283.1%	214.7%

Table 5: Run-to-run variations for OpenCL FFT running on CPU only.

Load	Perf. CV	Pwr. CV	Perf. Diff.	Pwr. Diff.
Idle		2.2%		7.4%
25%	0.8%	1.1%	3.3%	3.1%
50%	0.8%	0.8%	3.3%	3.0%
75%	0.8%	0.8%	3.1%	3.2%
100%	0.8%	0.6%	3.4%	2.5%

The OpenCL implementation features a slightly greater performance variation on the AMD system compared to running on the CPU, but clearly is still below the SERT limit of 5%. Interestingly, the relationship is the opposite when looking at the power variation. Variation in power consumption between runs is the lowest for the OpenCL workload running on the AMD GPU with a maximum CV of 0.7%, which is even lower than the power variation on the CPU. In turn, the CUDA workload running on the NVIDIA GPU results in the greatest power variation recorded in our measurements.

It is also noteworthy that the power and performance variations change with increasing load level on the GPU systems, but not on the CPU system. On the AMD system, variations decrease with increasing load. This is mostly due to the absolute variations remaining similar over load levels, meaning that the standard deviation remains the same, but average power and performance change. Of course, these changes in average power or performance with the same standard deviation result in a lower CV, which describes the relative deviation. The reverse effect can be observed on the NVIDIA system, where absolute deviations increase to a bigger extent than the mean power and performance of each load level.

Concluding, our run-to-run variation tests show that our methodology is able to achieve reproducible results on GPGPU accelerators. In general, our GPU methodology achieves a reproducibility comparable to that achieved by the underlying *SPEC Power and Performance Benchmark Methodology* [26]. While there are some small differences in power and performance variation between different GPGPU devices, these differences are very small and not significant enough to challenge reproducibility of benchmark results using our methodology.

Table 6: Run-to-run variations for CUDA/OpenCL FFT running on NVIDIA and AMD GPUs.

	Load	Perf. CV	Pwr. CV	Perf. Diff.	Pwr. Diff.
K40m	25%	0.2%	0.9%	0.5%	2.5%
	50%	0.2%	1.0%	0.6%	2.8%
	75%	0.2%	1.1%	0.5%	3.2%
	100%	0.2%	1.4%	0.5%	4.8%
MI25	25%	4.4%	0.7%	7.9%	1.3%
	50%	2.6%	0.7%	5.0%	1.4%
	75%	3.0%	0.3%	5.6%	0.6%
	100%	1.2%	0.6%	2.5%	1.2%

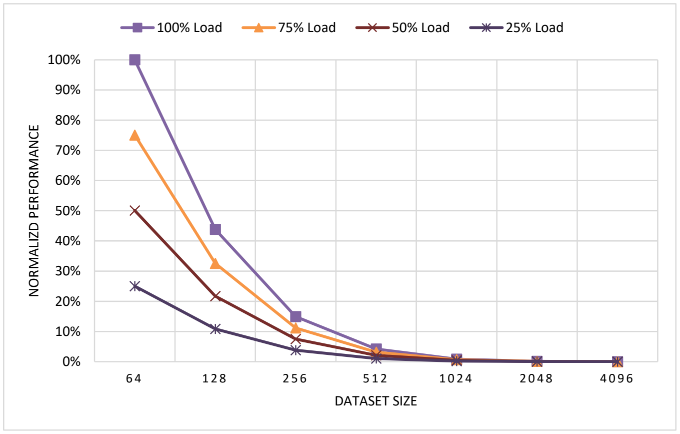

Figure 6: Performance scaling for OpenCL FFT running on CPU.

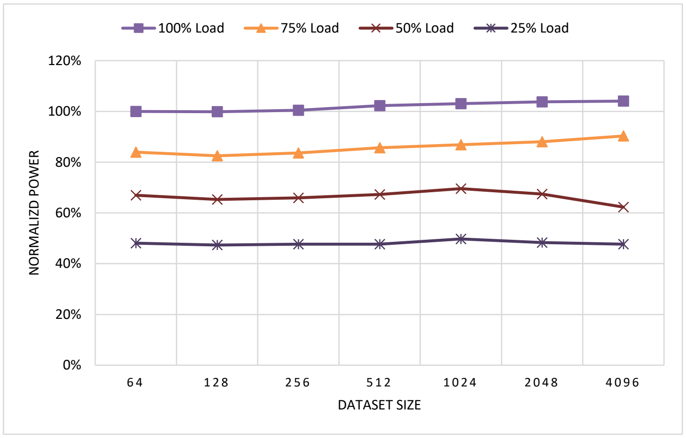

Figure 7: Power consumption scaling for OpenCL FFT running on CPU.

5.3 Power and Efficiency Scaling

Finally, we analyze how power consumption and energy efficiency scale with increasing load level and dataset size. The goal of this analysis is to show the relevance of our transaction-based benchmarking approach for GPGPU. In this part of the evaluation, we investigate the different types of power and efficiency scaling when increasing dataset size or transaction rate and how they depend on the GPU model. As a reference point, we run the OpenCL variant of our workload on the CPU of our CPU-only system (see Table 1). Based on this analysis, we compare the power and efficiency scaling of our GPU models.

In this analysis, we normalize all power and performance results in order to achieve comparability for the scaling behavior across device types. We normalize against the 100% load level result of the smallest dataset size. All other results are represented as the percentage performance/power/efficiency compared to this baseline result.

5.3.1 Power and Efficiency Scaling on CPU. First, we investigate the power and performance scaling for our FFT workload when running on a CPU using its OpenCL implementation. CPU power and performance scaling behavior is generally well known [30]. Thus this analysis serves as a reference point for the following GPU power and energy scaling analyses. We configure FFT to utilize all logical cores of the SUT in parallel.

Fig. 6 shows the relative performance of the different load levels and dataset sizes. By definition, performance scales linearly with the load levels, which is a result of the calibration and load level definition. In addition, performance decreases linearly with increasing dataset sizes (or rather exponentially with exponentially increasing dataset sizes). Even when considering the throughput of data packets (meaning transaction rate times dataset size), performance still decreases linearly, just at a smaller magnitude.

Power consumption also scales with load levels, as shown in Fig. 7. It shows a difference of 62% between the 25% load level and the 100% load level with a dataset size of 64. In contrast, power consumption does not seem to scale significantly with dataset size. For most sizes, it is almost constant, with only small deviations at a dataset size of 4096. This largest dataset size shows some variation in power consumption with power at 75% load increasing a bit and 50% load power decreasing. However, the transacation rates with this large dataset are so low that these observations are not very indicative of anything specific.

This almost constant power scaling with increasing dataset size leads to the energy-efficiency scaling observed in Fig. 8, where energy efficiency, just like the performance, scales linearly with increasing dataset sizes.

Notably, performance increases if we run FFT on a parallelization level equal to the number of physical CPU cores instead of logical cores. For example, the 100% load level features a mean performance increase of 25.3% when setting the parallelization level to the number of physical CPU cores. However, all our observations on power and performance scaling remain the same.

Concluding, the FFT workload's performance scales linearly with increasing load levels and dataset sizes, but power consumption scales only with load levels and does not seem to scale with dataset size.

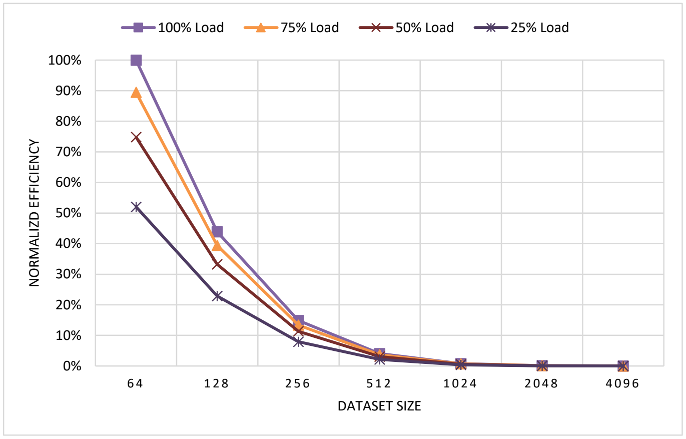

Figure 8: Energy efficiency scaling for OpenCL FFT running on CPU.

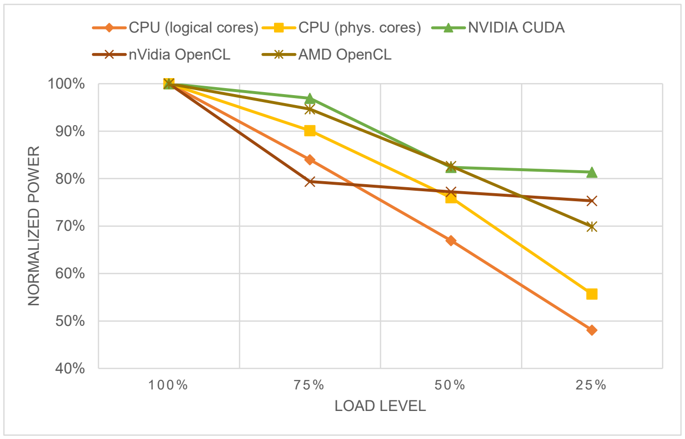

Figure 9: Power consumption scaling over load levels.

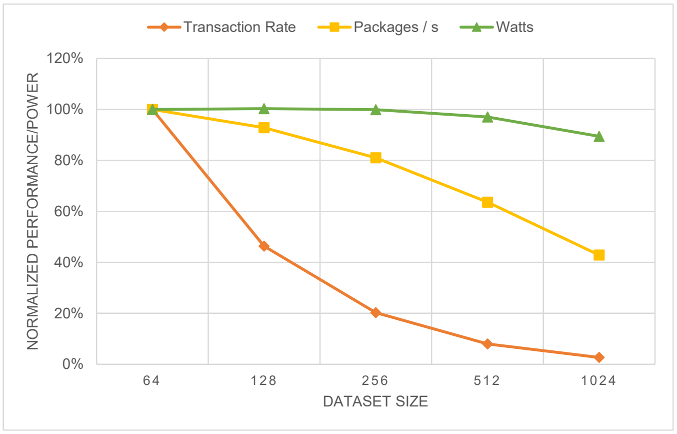

Figure 10: Power and performance with dataset size on AMD GPU with OpenCL FFT.

5.3.2 Scaling with Transaction Rate. We analyze if power scales with transaction rate on the GPGPU systems and how this compares to our baseline CPU system. We use a dataset size of 64, which is the smallest size measured on our systems. This dataset size performs best on the CPU, resulting in sufficient power differences between load levels to enable comparison.

Of course, performance scales linearly with increasing load levels, as this is the load level definition. However, power consumption scales differently depending on configuration and device used. Fig. 9 shows the normalized power consumption of the different setups over the four load levels. It shows that power scales to some degree for all considered devices. The smallest difference is observed on the NVIDIA SUT when running the CUDA variant of the FFT workload. In this case, the 25% load level still consumes 81.4% of the maximum power consumption. In turn, the biggest difference can be observed when running FFT on the CPU, with the parallelization level set to the number of logical cores. In this case, the 25% load level consumes only 48.08% of the system power consumed at full load.

All of the GPGPU servers feature a lower difference between low and high load power consumption. We can attribute this to two main factors: *(1)* The GPGPU systems are equipped with one additional device (the accelerator itself), containing a CPU that actually does not handle the workload. This way, the CPU adds to the static power consumption of the server system under consideration, decreasing the relative difference between minimum and maximum power. *(2)* GPUs are mostly designed to run at full load, handling graphics or HPC workloads. In contrast, CPUs are routinely run at middling loads and have seen significantly more optimization in this regard. However, even with these factors in mind, the GPGPU systems do show some significant scaling with load levels.

We also can observe two types of scaling behavior in Fig. 9. FFT's power consumption decreases with decreasing load level for both the OpenCL and the CUDA variant when running on the NVIDIA SUT. However, at 50% load for CUDA and 75% for OpenCL, power consumption stops decreasing and instead bottoms out at an almost constant value. Looking at it from another perspective, variations in low load don't seem to change power consumption a lot; instead, power consumption starts to scale only once a minimum load level has been achieved. This is different to the power-scaling behavior of the CPU-only and AMD GPGPU systems. Both these systems see a decrease in power consumption between each of the load levels.

5.3.3 Scaling with Dataset Size. Finally, we analyze how power consumption and performance scale with increasing dataset size. We focus this analysis on the 100% load level in order to determine the best achievable performance and efficiency based on the dataset configuration. We analyze the transaction rate (throughput), which is the number of datasets handled per second, the power consumption in watts, and the number of data points (packages) computed per second. The latter is the throughput multiplied by the dataset size. The idea behind investigating this metric is that it might give us an additional insight into the amount of work that actually is being performed.

Fig. 10 shows how normalized power consumption, transaction rate, and packages per second scale with increasing dataset size on the AMD GPU running the OpenCL variant of FFT. It shows that performance decreases significantly with increasing dataset size. In this sense, the scaling follows a similar path as observed in our CPU-only analysis in Section 5.3.1. Transaction rate drops linearly with increasing dataset size (or exponentially with exponentially increasing size). Notably, the number of handled data points also drops, yet it does not drop nearly as quickly as the transaction rate. This indicates that the performance per additional datapoint within the dataset drops beyond the work needed simply to handle this data point. Again, this is similar to the behavior observed on the CPU (see Fig. 11). However, the packages per second metric decreases significantly slower on the GPU. At a dataset size of 1024 packages, the AMD GPU still offers 42.9% of its performance at a dataset size of 64. In contrast, the CPU's packages per second drop to 13.2% compared to the dataset size of 64. In addition, the power consumption of the AMD system decreases alongside with the performance. This power consumption drop seems somewhat analogous to the drop in package per second performance, which may perhaps indicate some sort of idle time due to bus overheads. These idle times, in turn, are likely a result of unoptimized code in our OpenCL FFT library. The power consumption of the CPU-only system seems to support this hypothesis, as it does not decrease

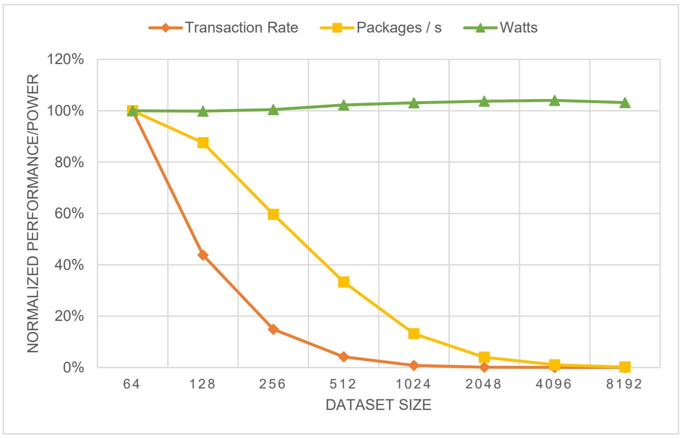

Figure 11: Power and performance with dataset size on CPU with OpenCL FFT.

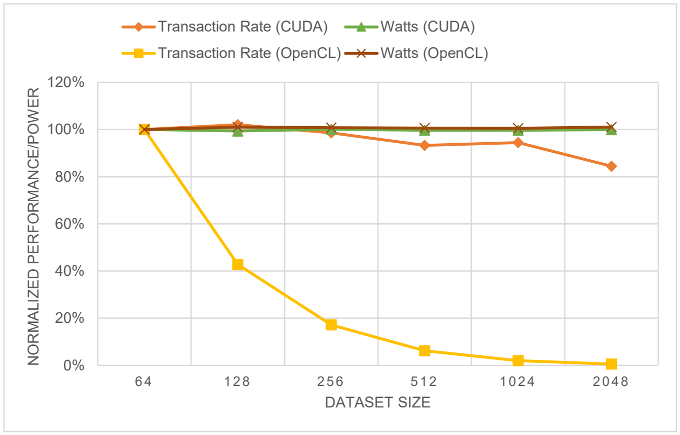

Figure 12: Power and performance with dataset size on NVIDIA GPU with CUDA and OpenCL FFT.

with additional data. Of course, the CPU does not need to copy main memory over the PCI-Express bus, as the GPU does.

Fig. 12 shows a different behavior. It shows the power consumption and transaction rate of the NVIDIA GPU system when running both the OpenCL and CUDA FFT implementations. Again the OpenCL implementation shows the same performance behavior observed on the AMD and CPU-only systems. Transaction rate drops linearly with increasing dataset size and packages per second drops (not shown in the figure), but to a lesser degree. This does further support our hypothesis on some optimization issues in the OpenCL FFT library employed by our workload. The CUDA FFT implementation, utilizing the cuFFT library [1], features an entirely different performance scaling behavior. The transaction rate stays constant when increasing the dataset size from 64 to 128 and then starts to degrade slowly to only 84.4% of its baseline performance at a dataset size of 2048. With the transaction rate decreasing so little, packages per second increases significantly with a larger dataset size. At a size of 2048, CUDA FFT manages to achieve 2702% of its baseline performance. Power consumption, however, behaves very similarly for both the OpenCL and CUDA implementations on the NVIDIA system. It remains almost constant for both despite the changes in performance. Absolute differences between power consumption of the OpenCL and CUDA implementation also are very small, with OpenCL drawing 2% additional power.

Concluding, our analysis of performance and power scaling with dataset size analysis shows that this scaling behavior is very device-specific regarding how power consumption behaves with increasing dataset size. However, as dataset size is a metric specific to the FFT workload, workload implementation details account for most of the performance behavior observed.

5.4 Concluding Remarks

Our evaluation shows that our transactional GPGPU benchmarking methodology is able to produce consistent and reproducible results with a maximum coefficient of variation of 1.4% regarding power consumption and a maximum coefficient of variation of 4.4% regarding performance. Our analysis shows that the methodology can be used to display scaling behavior in power consumption over load levels, using our transaction rate-based load level definition. We also show that the dataset size of our test workload has a significant impact on performance results, yet this impact seems to be more workload-specific compared to the general transaction-based load level scaling.

6 CONCLUSIONS

This paper presents a methodology for measuring and benchmarking the energy efficiency of servers with GPGPUs. The methodology runs transactional loads at different transaction rates in order to achieve a power and efficiency scaling characterization of the server and GPU under test. The methodology also is designed to consider different vendor-specific workload implementations, transaction scheduling, kernel parallelization, and GPGPU result verification.

We present a reference workload implementing FFT using the methodology. The reference workload is implemented with both OpenCL and CUDA variants. With this workload, we show that our benchmarking methodology is able to produce consistent and reproducible results with a maximum coefficient of variation of 1.4% regarding power consumption. We also demonstrate that it is able to be used to analyze how performance and power scale with transaction rate-based load levels.

The methodology in this paper is useful for server provisioners, developers, and researchers. Provisioners can use it to weigh different configuration options in order to configure data centers that include GPGPU systems in a more energy-efficient manner. Developers, in turn, can use our framework, which implements the methodology, to test the efficiency of their workloads. Finally, researchers can use the methodology and the power scaling that it can test to evaluate novel GPGPU power management, power prediction, and power saving methods.

For future research, we see a need for additional workloads within our methodology. We also will require new metrics that cover the different power-scaling behaviors regarding transaction rate and dataset sizes.

ACKNOWLEDGMENT

The authors wish to acknowledge current and past members of the SPECpower Committee and SPEC Research Power Working Group who have contributed to the design, development, testing, and overall success of SERT: Sanjay Sharma, Jeremy A. Arnold, Mike Tricker, Nathan Totura, Greg Darnell, Karl Huppler, Van Smith, Ashok Emani, Paul Muehr, David Ott, David Reiner, Karin Wulf, Cathy Sandifer, Jason Glick, and Dianne Rice, as well as the late Alan Adamson and Larry Gray.

REFERENCES

[1] 2018. CUDA cufft Toolkit Documentation. http://docs.nvidia.com/cuda/cufft/index.html. (2018). Last accessed: 10.2018.

[2] 2018. Java Native Interface Specification. https://docs.oracle.com/javase/8/docs/technotes/guides/jni/spec/jniTOC.html. (2018). Last accessed: 10.2018.

[3] Mohammad Abdel-Majeed, Daniel Wong, and Murali Annavaram. 2013. Warped gates: gating aware scheduling and power gating for GPGPUs. In *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*. ACM, 111–122.

[4] Jeremy Arnold. 2013. *Chauffeur: A framework for measuring Energy Efficiency of Servers*. Master Thesis. University of Minnesota.

[5] C. Babcock. 2012. NY Times data center indictment misses the big picture. *InformationWeek Cloud*.

[6] Eric Bainville. 2011. Bealto FFT. http://www.bealto.com/home.html. (2011). Last accessed: 10.2018.

[7] L.A. Barroso and U. Holzle. 2007. The Case for Energy-Proportional Computing. *Computer* 40, 12 (Dec 2007), 33–37. https://doi.org/10.1109/MC.2007.443

[8] Abhinav Bhatele, Sameer Kumar, Chao Mei, J. C. Phillips, Gengbin Zheng, and L. V. Kale. 2008. Overcoming scaling challenges in biomolecular simulations across multiple platforms. In *2008 IEEE International Symposium on Parallel and Distributed Processing*. 1–12. https://doi.org/10.1109/IPDPS.2008.4536317

[9] James Bucek, Klaus-Dieter Lange, and Jóakim v. Kistowski. 2018. SPEC CPU2017: Next-Generation Compute Benchmark. In *Companion of the 2018 ACM/SPEC International Conference on Performance Engineering (ICPE '18)*. ACM, New York, NY, USA, 41–42. https://doi.org/10.1145/3185768.3185771

[10] M. Burtscher, R. Nasre, and K. Pingali. 2012. A quantitative study of irregular programs on GPUs. In *2012 IEEE International Symposium on Workload Characterization (IISWC)*. 141–151. https://doi.org/10.1109/IISWC.2012.6402918

[11] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S. Lee, and K. Skadron. 2009. Rodinia: A benchmark suite for heterogeneous computing. In *2009 IEEE International Symposium on Workload Characterization (IISWC)*. 44–54. https://doi.org/10.1109/IISWC.2009.5306797

[12] J. Coplin and M. Burtscher. 2016. Energy, Power, and Performance Characterization of GPGPU Benchmark Programs. In *2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. 1190–1199. https://doi.org/10.1109/IPDPSW.2016.164

[13] Anthony Danalis, Gabriel Marin, Collin McCurdy, Jeremy S Meredith, Philip C Roth, Kyle Spafford, Vinod Tipparaju, and Jeffrey S Vetter. 2010. The scalable heterogeneous computing (SHOC) benchmark suite. In *Proceedings of the 3rd Workshop on General-Purpose Computation on Graphics Processing Units*. ACM, 63–74.

[14] Y. Gao, S. Iqbal, P. Zhang, and M. Qiu. 2015. Performance and Power Analysis of High-Density Multi-GPGPU Architectures: A Preliminary Case Study. In *2015 IEEE 17th International Conference on High Performance Computing and Communications, 2015 IEEE 7th International Symposium on Cyberspace Safety and Security, and 2015 IEEE 12th International Conference on Embedded Software and Systems*. 66–71. https://doi.org/10.1109/HPCC-CSS-ICESS.2015.68

[15] Sunpyo Hong and Hyesoon Kim. 2010. An Integrated GPU Power and Performance Model. In *Proceedings of the 37th Annual International Symposium on Computer Architecture (ISCA '10)*. ACM, New York, NY, USA, 280–289. https://doi.org/10.1145/1815961.1815998

[16] Qing Jiao, Mian Lu, Huynh Phung Huynh, and Tulika Mitra. 2015. Improving GPGPU Energy-efficiency Through Concurrent Kernel Execution and DVFS. In *Proceedings of the 13th Annual IEEE/ACM International Symposium on Code Generation and Optimization (CGO '15)*. IEEE Computer Society, Washington, DC, USA, 1–11. http://dl.acm.org/citation.cfm?id=2738600.2738602

[17] K.-D. Lange. 2009. Identifying Shades of Green: The SPECpower Benchmarks. *Computer* 42, 3 (March 2009), 95–97. https://doi.org/10.1109/MC.2009.84

[18] K.-D. Lange, Jeremy A. Arnold, Hansfried Block, Nathan Totura, John Beckett, and Mike G. Tricker. 2013. Further Implementation Aspects of the Server Efficiency Rating Tool (SERT). In *Proceedings of the 4th ACM/SPEC International Conference on Performance Engineering (ICPE '13)*. ACM, New York, NY, USA, 349–360. https://doi.org/10.1145/2479871.2479926

[19] K.-D. Lange and Michael G. Tricker. 2011. The Design and Development of the Server Efficiency Rating Tool (SERT). In *Proceedings of the 2nd ACM/SPEC International Conference on Performance Engineering (ICPE '11)*. ACM, New York, NY, USA, 145–150. https://doi.org/10.1145/1958746.1958769

[20] Klaus-Dieter Lange, Mike G. Tricker, Jeremy A. Arnold, Hansfried Block, and Christian Koopmann. 2012. The Implementation of the Server Efficiency Rating Tool. In *Proceedings of the 3rd ACM/SPEC International Conference on Performance Engineering (ICPE '12)*. ACM, New York, NY, USA, 133–144. https://doi.org/10.1145/2188286.2188307

[21] Jingwen Leng, Tayler Hetherington, Ahmed ElTantawy, Syed Gilani, Nam Sung Kim, Tor M Aamodt, and Vijay Janapa Reddi. 2013. GPUWattch: enabling energy optimizations in GPGPUs. In *ACM SIGARCH Computer Architecture News*, Vol. 41. ACM, 487–498.

[22] Erik Lindahl, Berk Hess, and David van der Spoel. 2001. GROMACS 3.0: a package for molecular simulation and trajectory analysis. *Molecular modeling annual* 7, 8 (01 Aug 2001), 306–317. https://doi.org/10.1007/s008940100045

[23] Steve Plimpton, Paul Crozier, and Aidan Thompson. 2007. LAMMPS-large-scale atomic/molecular massively parallel simulator. *Sandia National Laboratories* 18 (2007), 43.

[24] Meikel Poess, Raghunath Othayoth Nambiar, Kushagra Vaid, John M Stephens Jr, Karl Huppler, and Evan Haines. 2010. Energy benchmarks: a detailed analysis. In *Proceedings of the 1st International Conference on Energy-Efficient Computing and Networking*. ACM, 131–140.

[25] Suzanne Rivoire, Mehul A. Shah, Parthasarathy Ranganathan, and Christos Kozyrakis. 2007. JouleSort: A Balanced Energy-efficiency Benchmark. In *Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data (SIGMOD '07)*. ACM, New York, NY, USA, 365–376. https://doi.org/10.1145/1247480.1247522

[26] Standard Performance Evaluation Corporation. 2014. SPEC Power and Performance Benchmark Methodology. http://spec.org/power/docs/SPEC-Power_and_Performance_Methodology.pdf. (December 2014).

[27] John A Stratton, Christopher Rodrigues, I-Jui Sung, Nady Obeid, Li-Wen Chang, Nasser Anssari, Geng Daniel Liu, and Wen-mei W Hwu. 2012. Parboil: A revised benchmark suite for scientific and commercial throughput computing. *Center for Reliable and High-Performance Computing* 127 (2012).

[28] Jóakim von Kistowski, Jeremy A. Arnold, Karl Huppler, Klaus-Dieter Lange, John L. Henning, and Paul Cao. 2015. How to Build a Benchmark. In *Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering (ICPE 2015) (ICPE '15)*. ACM, New York, NY, USA. https://doi.org/10.1145/2668930.2688819

[29] Jóakim von Kistowski, John Beckett, Klaus-Dieter Lange, Hansfried Block, Jeremy A. Arnold, and Samuel Kounev. 2015. Energy Efficiency of Hierarchical Server Load Distribution Strategies. In *Proceedings of the IEEE 23nd International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS 2015)*. IEEE.

[30] Jóakim von Kistowski, Hansfried Block, John Beckett, Klaus-Dieter Lange, Jeremy A. Arnold, and Samuel Kounev. 2015. Analysis of the Influences on Server Power Consumption and Energy Efficiency for CPU-Intensive Workloads. In *Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering (ICPE 2015) (ICPE '15)*. ACM, New York, NY, USA. https://doi.org/10.1145/2668930.2688057

[31] G. Wu, J. L. Greathouse, A. Lyashevsky, N. Jayasena, and D. Chiou. 2015. GPGPU performance and power estimation using machine learning. In *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*. 564–576. https://doi.org/10.1109/HPCA.2015.7056063