Cactus: Top-Down GPU-Compute Benchmarking using Real-Life Applications

Mahmood Naderan-Tahan
Ghent University
mahmood.naderan@ugent.be

Lieven Eeckhout
Ghent University
lieven.eeckhout@ugent.be

2021 IEEE International Symposium on Workload Characterization (IISWC) | 978-1-6654-4173-5/21/$31.00 ©2021 IEEE | DOI: 10.1109/IISWC53511.2021.00026

***Abstract*—Benchmarking is the de facto standard for evaluating hardware architectures in academia and industry. While several benchmark suites targeting different application domains have been developed for CPU processors over many decades, benchmarking GPU architectures is not as mature. Since the introduction of GPUs for general-purpose computing, the purpose has been to accelerate (a) specific part(s) of the code, called (a) kernel(s). The initial GPU-compute benchmark suites, which are still widely used today, hence consist of relatively simple workloads that are composed of one or few kernels with specific unambiguous execution characteristics. In contrast, we find that modern-day real-life GPU-compute applications are much more complex consisting of many more kernels with differing characteristics. A fundamental question can hence be raised: are current benchmark suites still representative for modern real-life applications?**

In this paper, we introduce *Cactus*, a collection of widely used real-life open-source GPU-compute applications. The aim of this work is to offer a new perspective on GPU-compute benchmarking: while existing benchmark suites are designed in a bottom-up fashion (i.e., starting from kernels that are likely to perform well on GPUs), we perform GPU-compute benchmarking in a top-down fashion, starting from complex real-life applications that are composed of multiple kernels. We characterize the *Cactus* benchmarks by quantifying their kernel execution time distribution, by analyzing the workloads using the roofline model, by performing a performance metrics correlation analysis, and by classifying their constituent kernels through multi-dimensional data analysis. The overall conclusion is that the *Cactus* workloads execute many more kernels, include more diverse and more complex execution behavior, and cover a broader range of the workload space compared to the prevalently used benchmark suites. We hence believe that *Cactus* is a useful complement to the existing GPU-compute benchmarking toolbox.

I. Introduction

GPU devices have become critical components not only in desktop computers but also in server and throughput-class computers thanks to the introduction of general-purpose GPU programming models such as CUDA for Nvidia platforms [42] and OpenCL [41] for vendor-independent platforms. So-called GPU-compute systems provide convenient programming interfaces, so that it is no longer needed to translate general computation models to the graphics programming model to leverage the massive computational power of the GPU. Since the beginning of GPU-compute systems, the focus has been on high-performance and data-parallel applications with a broad range of application domains. Examples include high-performance computing (HPC), graph analytics, as well as machine learning, which all employ GPUs as their accelerator of choice [11], [33], [36], [37], [51], [60], [66]. While different applications have different needs, evaluating different architectures for performance efficiency is a major challenge. Computer architects rely on benchmark suites for this purpose and selecting a set of representative applications is of critical importance when composing a benchmark suite.

The concept of hardware acceleration where a specific part of a problem, called a kernel, is offloaded onto a specific device has triggered a bottom-up approach for designing GPU benchmark suites. In particular, the Berkeley 13 dwarfs model [4] highlights important problems that need parallelization: this includes dwarfs that exemplify traditional numerical methods (e.g., linear algebra, n-body methods) as well as dwarfs for emerging applications such as machine learning and graph traversals. The most widely used GPU-compute benchmark suites, Rodinia [8] and Parboil [58], are designed using the 13 dwarfs as a guideline. More specifically, the Rodinia paper states that 'each application or kernel is carefully chosen to represent different types of behavior according to the Berkeley dwarfs' [8]. Parboil claims to be 'very similar in philosophy and development' [58]. In other words, today's prevalent benchmark suites have been developed in a bottom-up fashion starting from a set of key prototypical kernels.

Over the past decade though, GPU programming interfaces have changed substantially and several highly optimized libraries (e.g., CUDA-X [10] and ArrayFire [67]) have been developed. In addition, new application domains have emerged and a more diverse set of advanced algorithms have been ported to GPU devices. In particular, graph analytics and machine learning have emerged as key drivers for continuing to push GPU-compute performance and system scaling [3], [18]. With such a widespread usage, considerable efforts have been made to create new standard benchmark suites. MLPerf [19], [20] is recognized as the most advanced benchmark suite for machine learning (ML) as it focuses on end-to-end performance benchmarking using state-of-the-art software stacks. While MLPerf is a valuable asset for benchmarking large-scale ML systems, using MLPerf for GPU-compute system evaluation and research is tedious because of lack over control of scale and the limited visibility that it provides into system- and architecture-level performance phenomena. Alternative benchmark suites, e.g., Tango [30] and Darknet [52], use custom implementations that do not rely on state-of-the-art libraries such as CuDNN for their ML models. Other benchmark suites, such as DeepBench [44] and Altis [28], do

978-1-6654-4173-5/21/$31.00 ©2021 IEEE
DOI 10.1109/IISWC53511.2021.00026

use CuDNN to implement some of the basic machine learning operations but they are compatible with specific versions and they do so in a bottom-up fashion — basic machine learning operations are considered as computational kernels, much like Rodinia and Parboil for scientific computing.

We thus conclude that across various GPU-compute workload domains, from scientific computing to machine learning, there is a major gap in the benchmarking space for GPU-compute systems. Existing benchmark suites have been developed in a bottom-up and kernel-centric manner, they do not rely on state-of-the-art libraries, or they are too large a scale, failing computer architects to provide the visibility they need. There is hence a need for GPU-compute benchmarks with the following key properties: (1) they are derived from real-life applications in a top-down fashion, (2) they measure end-to-end application performance, (3) they rely on state-of-the-art libraries, and (4) they provide visibility into system- and architecture-level performance phenomena.

To fill this gap in the benchmarking space, we use a top-down approach by creating a benchmark suite that reflects modern-day GPU-compute applications. We introduce *Cactus*, a collection of real-life GPU-compute applications that are widely used and cover domains as broad as molecular simulation, graph traversal and machine learning. In contrast to existing bottom-up benchmarking methodologies that focus on essential computational kernels, we select and analyze benchmarks in a top-down fashion by considering real-life state-of-the-art applications that rely on state-of-the-art libraries and are of appropriate scale for GPU-compute architecture research. We analyze and characterize the *Cactus* workloads on a modern GPU platform and we quantify the kernel execution time distribution, we analyze their compute versus memory-bound characteristics using the roofline model, and we perform a performance metrics correlation analysis as well as multi-dimensional data analysis. These analyses lead to the following key observations:

- The real-life applications in *Cactus* execute multiple tens of kernels, in contrast to benchmarks from traditional benchmark suites (Parboil, Rodinia and Tango) which execute one or just a few kernels.
- Which kernels get invoked in the real-life applications depends on the application's input.
- The execution behavior in real-life applications is more complex and ambiguous compared to the traditional benchmark suites.
- Kernels from the same real-life application may exhibit widely different execution characteristics, some being compute-intensive while others being memory-intensive. This suggests that optimizing real-life application performance is more challenging than what existing benchmark suites may indicate.
- Compared to the molecular and graph analytics applications in *Cactus*, the ML applications execute many more kernels with a much wider diversity in compute- versus memory-intensive behavior and performance. The ML applications also feature dominant kernels that are bound by memory bandwidth.
- Overall, *Cactus* covers a larger part of the workload space than traditional benchmark suites.

The *Cactus* benchmark suite can be accessed and downloaded following the instructions reported in the artifact appendix.

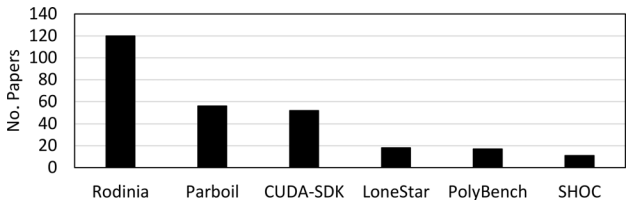


Fig. 1: Popularity of GPU-compute benchmark suites over the past decade in top-four computer architecture conferences (ISCA, MICRO, ASPLOS and HPCA). *Rodinia and Parboil are the most popular GPU-compute benchmark suites.*

II. MOTIVATION

It is of critical importance that benchmarks are representative for their target workloads and systems. We briefly revisit the most popular GPU-compute benchmark suites and discuss their limitations, which further motivates the work presented in this paper.

A. Popular GPU-Compute Benchmark Suites

Since the introduction of GPUs for general-purpose computing, there have been several efforts to create benchmarks for architectural evaluation. Our survey shows that researchers use a wide variety of GPU-compute benchmarks, with some being more popular than others. Figure 1 reports the number of GPU-related papers in the top four computer architecture conferences (ISCA, MICRO, ASPLOS and HPCA) from 2010 until 2020, and the benchmarks they use in their experimental evaluations. Arguably, Rodinia [8] is the most popular benchmark suite, followed by Parboil [58]. These benchmark suites are more widely used compared to other suites due to their application diversity, deployment easiness and simulator-friendly structure. Although CUDA-SDK [55] has also received attention for architectural evaluations, it just includes a set of CUDA examples from various applications which has been unofficially recognized as a benchmark suite. Other popular suites are LoneStar [31], PolyBench [25] and SHOC [12]. However, they mostly contain computational kernels such as FFT, LUD and matrix multiplication with specific properties, e.g., irregularity — some of these benchmarks also appear in Rodinia and Parboil. As argued previously, given their focus on the implementations of specific essential kernels (i.e., dwarfs), we categorize these benchmark suites as bottom-up benchmarking initiatives.

Apart from their kernel-centric focus and bottom-up design principle, and while they have been and remain to be important benchmarks, Rodinia and Parboil suffer from two additional limitations: (1) The benchmarks are based on decade-old algorithms and implementations, and more advanced algorithms

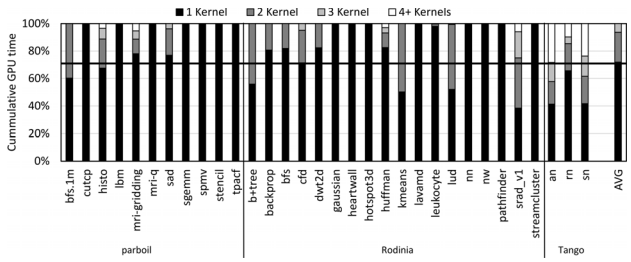


Fig. 2: GPU time distribution for a set of Parboil, Rodinia and Tango benchmarks. *Existing benchmark suites spend the majority of their execution time in one or just a few kernels.*

have been developed over the years — examples are breadth-first-search (BFS) for which better scalable implementations have been proposed for large graphs [39], or GEMM which has more advanced implementations in cuBLAS [9] and CUTLASS [21]; and (2) Rodinia and Parboil focus on the traditional GPU-compute workloads in scientific computing, while GPUs have been recently deployed for new emerging workload domains including graph analytics and machine learning, which to date have no representation in these benchmark suites.

Tango [30] is a recently proposed benchmark suite for machine learning, which is quickly gaining traction for driving architecture research — however, given its very recent release in 2019, only few papers have been using Tango so far (which is why we did not include Tango in Figure 1). The Tango benchmarks, as mentioned before, do not use advanced libraries, such as CuDNN, but instead implement some of the machine learning models to be compatible with architectural simulators. We consider the Tango benchmarks in this paper as a representative for benchmarking studies in the machine learning domain.

B. Kernel-Centric Benchmarks

The key conclusion from the previous section is that existing benchmark suites are kernel-centric, i.e., they are designed in a bottom-up fashion starting from kernels that are easily accelerated on GPUs. Figure 2 shows stacked GPU execution time bars for a set of benchmarks from Parboil, Rodinia and Tango. (More details about the experimental setup are provided in Section IV.) The figure highlights the fact that for approximately 70% of the workloads (23 out of 31), at least 70% of the total execution time is spent in a single kernel; and for approximately 25% of the workloads (7 out of 31), at least 70% of the GPU time is spent in at most two kernels; finally, only two workloads have three kernels that take up at least 70% of the GPU time. The overall conclusion is that the benchmarks from prevalent benchmark suites spend most of their execution time in one or just a few kernels. This is a direct consequence of the bottom-up design philosophy behind these benchmarking methodologies, as argued before.

C. Towards Application-Centric Benchmarks

In contrast to what we observe in prevalent benchmark suites, we find that modern-day GPU-compute applications consist of multiple (a dozen and up to multiple tens of) kernels, as we will quantify later in this paper. This observation has important implications for optimization, especially if those kernels exhibit widely different execution characteristics, e.g., some kernels are predominantly compute-intensive while others are primarily memory-intensive. We use a couple of examples to illustrate this further. Consider first a GPU-compute application that consists of a single kernel. Improving the performance of the kernel by, say, 20% will lead to a 20% performance improvement for the application. However, in case a GPU-compute application consists of multiple kernels, one has to either optimize all kernels by 20% to achieve an average overall application performance improvement of 20%, or one has to optimize the dominant kernel(s) by a margin that is substantially larger than the 20% average improvement. Consider for example a workload that consists of five kernels with the following time distribution: $\{0.25, 0.2, 0.2, 0.2, 0.15\}$ and IPC distribution (at SM sub-partition level): $\{1.2, 1.6, 2, 1.5, 1.2\}$. To realize a 20% average performance improvement at the application level, one has to double the performance of the most dominant kernel (i.e., the first kernel in this example which accounts for 25% of the application execution time), or one has to optimize all kernels by 20% on average. Neither of these two options is easy achievable in practice.

In summary, if GPU time is distributed across multiple kernels, improving overall application performance is not as straightforward as for a single-kernel program. To speed up program execution by a factor X, we have to improve the performance of the dominant kernel(s) by a factor that is much larger than X, or, alternatively, we have to improve the performance of all kernels by a factor X. Furthermore, to make matters even more complicated, optimizing the performance of one (or a few) kernels may create (possibly negative) side effects on other kernels.

III. Cactus Benchmark Suite

In this section, we describe the GPU-compute applications included in the *Cactus* benchmark suite. These applications have the following properties in common: (1) They are popular and widely used; (2) They are open-source and hence we can share the benchmarks with the community; and (3) They have a large user base with active support and regular updates. In the end, we have selected a total of ten workloads from three key application domains including molecular simulation, graph traversal and machine learning. We now describe the benchmarks in more detail.

A. Molecular Simulation

Molecular dynamics is a problem domain that is inherently data-parallel, and hence is an area with suitable candidates for GPU acceleration — in fact, several open-source software

TABLE I: The *Cactus* benchmark suite: benchmarks, inputs, and basic execution characteristics.

Domain	*Workload*	*Abbr.*	*Description*	*Data set*	*Total no. warp instructions*	*Weighted average no. warp instructions per kernel*	*No. kernels*	
							100% execution time	*70% execution time*
Molecular	Gromacs	GMS	NPT Equilibration	T4 lysozyme	306 B	43 M	9	3
	LAMMPS1	LMR	Protein simulation	Rhodopsin (32K atoms)	265 B	46 M	15	2
	LAMMPS2	LMC	Pairwise interactions between particles	Colloid (60K atoms)	53 B	3 M	9	3
Graph	BFS	GST	BFS traversal on Social network	SOC-Twitter10	1.7 B	187 M	12	1
	BFS	GRU	BFS traversal on Road network	Road USA	1.6 B	40 K	8	3
Machine Learning	DCGAN	DCG	Train a GAN network	Celeba	621 B	43 M	50	9
	Neural Style	NST	Train a CNN to generate artistic image	Original and artistic images	153 B	93 M	44	11
	Reinforcement Learning	RFL	Train a CNN with Deep-Q network	Flappy bird game	46 B	2.1 M	50	13
	Spatial Transformation	SPT	Train a spatial transformer network	MNIST	11 B	2.4 M	37	10
	Language Translation	LGT	Train seq2seq model to translate sentences	Spacy German news	125 B	3.2 M	66	14

packages target this domain [57]. We have selected two well-known applications from the domain of molecular simulation:

- *Gromacs* is a versatile package for biochemical molecular simulation with hundreds to millions of particles that have complicated bonded interactions, e.g., lipids and proteins [1], [14], [32], [45]. We use the 2021 single-precision version with CUDA and threaded-MPI enabled features. We consider a simulation system containing a T4 lysozyme protein in a complex with a ligand. The equilibration was conducted under the NPT ensemble (constant number of particles, pressure and temperature) for 5,000 steps.
- *LAMMPS* is molecular dynamics simulation software focusing on material modeling including solid-state, soft-matter, coarse-grained and mesoscopic systems [7], [16], [56], [63]. We use the 2020 version of the code with single-precision and CUDA-enabled features. Two inputs were considered: (1) Rhodopsin, a solvated protein model with 32 K atoms for 3,000 steps, and (2) a Colloid model with 60 K atoms for 2,000 steps.

B. Graph Analytics

Graph analytics workloads have become a critical workload with the recent emergence of big data and social networks. Although prior benchmark suites have included graph processing workloads, such as breadth-first-search (BFS), the initial implementations are suboptimal. More recently, highly optimized libraries have been developed and increasingly widely used. One such example is *Gunrock*, which is a CUDA library for graph-processing designed specifically for the GPU. It uses a high-level, bulk-synchronous, data-centric abstraction focused on operations on a vertex or edge frontier [15], [62]. Analyses show that Gunrock is fast compared to other graph libraries for large graphs [46], [61], [65]. In this work, we use BFS graph traversal using the Gunrock library for two large graphs: (1) a social network graph with 21 M vertices and 265 M edges [53], and (2) a road network with 23 M vertices and 28 M edges [5]. As we will quantify in the evaluation section of this paper, the version of BFS included in *Cactus* invokes multiple kernels as opposed to the version included in Rodinia and Parboil.

C. Machine Learning

Machine learning, and deep learning in particular, is one of the important killer applications for GPU-compute systems. PyTorch is a Python library for deep learning on irregular input data, including Tensor computation with GPU support and deep neural networks built on a tape-based autograd system [43], [47]. With its simple and easy to learn programming interface, there are numerous projects that use the PyTorch framework for machine learning operations [22], [35], [48]. For our purpose, we have selected the training phase of a number of state-of-the-art learning models [59]. Because the training phase is a time-consuming repetitive task that lends itself well to GPU acceleration, we analyze the applications for one epoch and an arbitrary number of iterations. Based on our observations, achieved results are consistent across different number of epochs and iterations. We have selected the following applications:

- *DCGAN* or Deep Convolutional Generative Adversarial Network is a framework for teaching a deep learning model where it can generate new data after extracting information features using convolutional networks and leaning an input data set [50]. We select the training phase

of the model to generate new pictures from the Celeb-A data set [38].

- *Neural Style* is a deep neural network that takes an input image along with a style image and produces an artistic image with high perceptual quality [23]. The algorithm is based on Convolutional Neural Networks (CNNs) which are powerful for image processing. In our analysis we train the model to use one content image and one style image to produce the artistic image.
- *Reinforcement Learning* is a machine learning area that tries to instruct an artificial agent to take actions in an environment to maximize its reward. Proposed by DeepMind, Deep Q-Network (DQN) is a method for reinforcement learning [40]. In this work, we choose the training phase of the flappy bird game [26].
- *Spatial Transformer* aims to enhance geometric invariants of a model by allowing a neural network to learn how to perform spatial transformations on an input image. Such enhancements include correcting the orientation of an image or cropping part of an image [29]. For our work, we use an SGD algorithm for the training phase on the MNIST data set [27].
- *Language Translation* uses machine learning models to translate sentences from one language to another. For this purpose, a sequence-to-sequence model is trained to translate German sentences into English from the Spacy dataset [6].

D. Discussion

Table I briefly summarizes the selected benchmarks, their inputs, and the abbreviations that we will use throughout the paper. Additionally, some other basic execution statistics are shown in the table, including the total number of dynamically executed instructions per warp, which ranges from 1.6 B up to 621 B instructions. (Note that a warp instruction consists of 32 thread instructions.) The average number of warp instructions executed per kernel ranges between 40 K and 187 M. All applications consist of multiple (tens of) kernels ranging between 8 and up to 66. Some applications require up to 14 kernels to account for 70% of the total execution time.

We do not claim that the *Cactus* benchmark suite is representative of the entire domain of molecular simulation, graph analytics and/or machine learning, let alone the entire HPC space. Instead, we believe though that the *Cactus* benchmarks reflect modern-day GPU-compute applications. In particular, the *Cactus* benchmarks are more complex than current existing benchmarks, as we will quantify in the results section of this paper. We hence believe that *Cactus* is a useful complement to existing benchmark suites for driving GPU-compute architecture explorations.

IV. Experimental Setup

We analyze and contrast the *Cactus* benchmark suite against previously proposed benchmark suites using the following experimental setup.

TABLE II: System setup.

GPU	Nvidia RTX 3080, 68 multiprocessors and 128 CUDA cores per multiprocessor at 1.9 GHz, Ampere SM architecture, 10 GB DRAM with 320-bit memory bus, 760 GB/s DRAM bandwidth, 5 MB L2 cache, released in 2020
Libraries	CUDA Toolkit 11.2, CuDNN 8.1, Nsight Compute 2020.3
Software	Gromacs-2021, LAMMPS-2020, Gunrock-1.1, PyTorch-1.7.1, TorchText-0.8.1 and TorchVision-0.8.2

TABLE III: Benchmarks from Parboil, Rodinia and Tango.

Suite	Workload
Parboil	bfs (1M), cutcp, histo, lbm, mri-gridding, mri-q, sad, sgemm, spmv, stencil, tpacf
Rodinia	b+tree, backprop, bfs, cfd, dwt2d, gaussian (4K), heartwall, hotspot3d, huffman, kmeans, lavamd, leukocyte, lud, nn, nw, pathfinder, srad_v1, streamcluster
Tango	alexnet (AN), resnet (RN), sqeezenet (SN)

System Specification. We use a state-of-the-art high-end Nvidia RTX 3080 GPU with the latest Ampere architecture. We further use the Nvidia Nsight Compute 2020 software profiler and various up-to-date software libraries as summarized in Table II. Due to the large number of kernels and the repetitive behavior of molecular and machine learning applications, we limit the program execution time that we profile to reduce the amount of data generated by the profiler. Before gathering our final performance metrics, we have run the programs multiple times in a fast tracing mode to identify a steady-state region of execution. For the graph analytics workloads, we profile the entire execution time.

Workloads. We contrast the *Cactus* benchmarks described in Section III against several other workloads from Rodinia, Parboil and Tango, as listed in Table III. We use the large or reference inputs where possible or create sufficiently large inputs with the provided input generator scripts (e.g., 4K input for Gaussian from Rodinia). We did not include all benchmarks from these benchmark suites for a variety of reasons including too small or hard-coded inputs, or incompatibility with the profiler. We do not use recently released machine learning benchmarks, such as DeepBench [44] and Altis [28], because their code bases are incompatible with CUDA 11.2 supported by the Nvidia Ampere architecture.

Performance Model. We analyze *Cactus* using the roofline model [13]. This performance model plots *performance* or the number of Giga (warp) Instructions Per Second (GIPS) versus *instruction intensity* or the number of warp instructions per DRAM transaction. The theoretical maximum GIPS for the RTX 3080 is calculated as follows: 68 (no. SMs) $\times 4$ (no. warp schedulers) $\times 1$ (warp instructions per cycle) $\times 1.9$ (clock frequency in GHz), which amounts to a total of 516.8 GIPS. In this model, a kernel's performance is a function of the peak machine bandwidth, calculated by the number of Giga transactions per second, instruction intensity and the device's peak GIPS. With a GDDR6X bandwidth of 760.3 GB/s and transaction size of 32 bytes, the peak memory bandwidth

TABLE IV: Performance characteristics.

Metric	Description
Warp occupancy	Average no. of active warps across all SMs
SM efficiency	Fraction of time w/ at least one active warp per SM
L1/L2 hit rate	Fraction of accesses that hit in L1 or L2
DRAM read throughput	Total DRAM read bytes per second
LD/ST utilization	Average load/store functional unit utilization
SP utilization	Average FP32 pipeline utilization
Fraction branches	Fraction branch instructions
Fraction LD/ST insts	Fraction memory operations
Execution stall	Stall ratio due to execution dependencies
Pipe stall	Stall ratio due to busy pipeline
Sync stall	Stall ratio due to synchronization
Memory stall	Stall ratio due to memory accesses

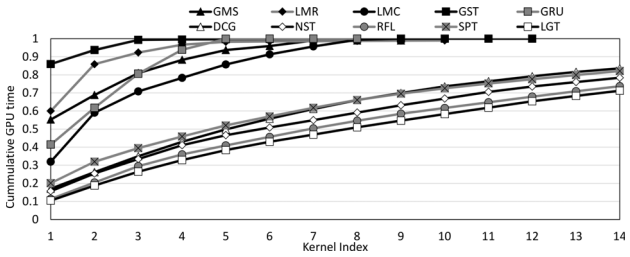


Fig. 3: Cumulative distribution of execution time spent in the most dominant kernels for the *Cactus* workloads. *Real-life applications tend to spend their execution time in many more kernels than traditional benchmarks. Different (number of) kernels are executed depending on the benchmark input.*

equals 23.75 GTXN/s (Giga Transactions per Second). Therefore, the elbow point where the memory roof reaches the compute roof is when instruction intensity equals 21.76 warp instructions per DRAM transaction.

To analyze a workload using the roofline model, we have to calculate two metrics: (1) *instruction intensity (II)* which is computed as the total number of warp instructions divided by the total number of memory transactions; and (2) *performance (GIPS)* which is computed as the total number of warp instructions divided by the kernel's runtime (in seconds). In addition to these key performance metrics, we also collect several other metrics for deeper analysis as provided by the profiler, see Table IV.

Dominant Kernels. Assume a program consists of N kernels with $N \geqslant 1$. Each kernel i is invoked r_i times and each invocation takes t_i time unit. This implies that the total execution time of the kernel equals $T_i = \sum_{i=1}^{N} r_i \times t_i$. The total GPU application execution time hence equals $T = \sum_{i=1}^{N} T_i$. The kernel that has the highest $r_i \times t_i$ rank is called the dominant kernel. This implies that we do not solely rely on the execution time per kernel invocation to determine the most dominant kernel. A kernel that has a relatively short per-invocation execution time may be invoked more frequently, and may hence become a more dominant kernel. As we will argue in the results section, we will primarily focus on the dominant kernels that collectively amount for at least 70% of the total GPU execution time.

V. Methodology

We now analyze and compare the *Cactus* workloads against the Parboil, Rodinia and Tango benchmarks. We first quantify the number of kernels executed, and then characterize the applications and their constituent kernels using the roofline model. We finally analyze the (dis)similarity among the benchmarks using data analysis techniques such as hierarchical clustering and factor analysis using mixed data.

A. GPU Time Distribution

Figure 3 reports the cumulative distribution of time spent in the different kernels for the *Cactus* workloads. The vertical axis shows the cumulative GPU time while the horizontal axis shows the number of kernels. (We limit the number of kernels to 14 to account for at least 70% of the total execution time, see also the data presented in Table I.) This result leads to the following key observations:

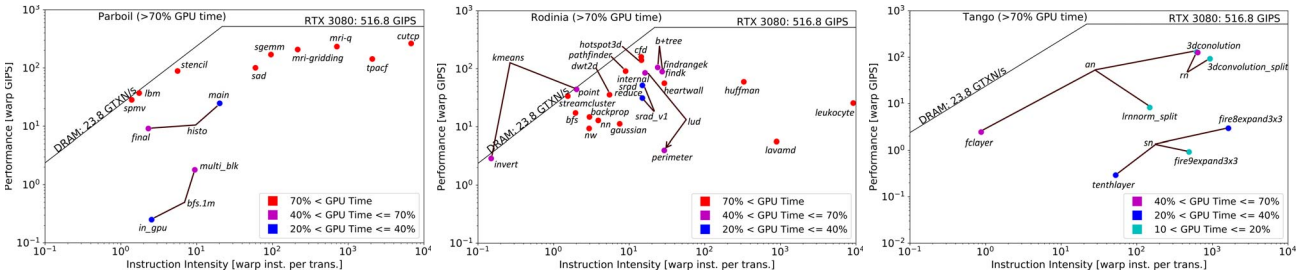
Observation #1: *Real-life applications tend to execute many more kernels than traditional benchmarks.* Whereas Rodinia, Parboil and Tango spend most of their time in one or just a few kernels, as previously reported, the *Cactus* applications spend their execution time in many more kernels. This is particularly the case for the ML applications for which a dozen of kernels account for approximately 70% of the execution time. For all molecular dynamics and graph analytics benchmarks (except *GST*), we note that at least three kernels are responsible for at least 90% of the total execution time.

Observation #2: *The total number of kernels executed rises up to multiple tens in real-life applications.* The total number of kernels executed rises up to multiple tens of kernels, as is the case for the ML workloads, e.g., 66 kernels account for 100% of the execution time for *LGT*, see Table I. Also, for the other workload categories we note that up to a couple dozen kernels are executed at least once: up to 15 kernels for the molecular dynamics *LMR* benchmark, and up to a dozen kernels for the graph analytics workload *GST*.

Observation #3: *Real-life applications tend to execute different kernels depending on the input.* It is worth noting that some workloads are input-sensitive in the sense that they execute different (number of) kernels depending on the input provided to the application. In particular, we note different execution profiles for the *Lammps* workload (see *LMR* versus *LMC*) as well as for the *Graph* workload (see *GST* versus *GRU*). These workloads execute the same code base, but different inputs trigger the execution of different kernels.

B. Roofline Analyses

1) Parboil, Rodinia and Tango: Figure 4 shows the roofline model for the Parboil, Rodinia and Tango workloads. As aforementioned, the roofline model reports performance (GIPS) against instruction intensity. The elbow point in the roofline is where the memory roof reaches the compute roof, and hence



(a) Parboil (b) Rodinia (c) Tango

Fig. 4: Roofline model for (a) Parboil, (b) Rodinia, and (c) Tango. *Most of the workloads exhibit unambiguous behavior: their constituent kernels are either compute-intensive or memory-intensive. There are few exceptions as indicated, including LUD (from Rodinia) and AN (from Tango).*

workloads situated on the left-hand side of the elbow point are memory-intensive whereas workloads on the right-hand side are compute-intensive. In these roofline plots, the colorization indicates the GPU time spent in each kernel. As noted before, most Parboil, Rodinia and Tango benchmarks consist of a single or just a few kernels that take up most (>70%) of the execution time.

Observation #4: *The Parboil, Rodinia and Tango benchmarks exhibit mostly unambiguous execution behavior, i.e., they are either memory-intensive or compute-intensive, but not mixed.* The Parboil, Rodinia and Tango benchmarks are located on the left-hand side or the right-hand side of the elbow point — this is unsurprising given that most benchmarks consist of a single dominant kernel. However, even the benchmarks that spend a significant amount of time in more than one kernel, have all of their kernels on one side of the elbow point, i.e., all kernels are either memory-intensive or they are all compute-intensive. Examples include *BFS* and *Histo* from Parboil for which all kernels are memory-intensive. For Rodinia, all kernels in *Kmeans* and *Srad_v1* are memory-intensive and all kernels in *B+tree* are compute-intensive; the only exception is *LUD* which consists of a memory-intensive kernel and a compute-intensive kernel. For Tango, we also note that all kernels in *SN* and *RN* are compute-intensive, whereas *AN* has three kernels out of which two are compute-intensive and one is memory-intensive. In short, out of the 31 workloads, only two workloads have kernels with mixed memory- and compute-intensive execution characteristics. This makes the performance optimization of these workloads relatively simple: by knowing the position of the dominant kernel in the roofline chart, architects are able to focus on the parameters that are likely to affect performance the most, i.e., improving performance of a memory-intensive workload is likely to be achieved by increasing effective memory bandwidth, whereas improving the performance of a compute-intensive workload is likely to be achieved by increasing the effective compute bandwidth.

2) Cactus: Aggregate Behavior: We now analyze the *Cactus* workloads using the roofline model, see also Figure 5 which reports the overall application execution characteristics

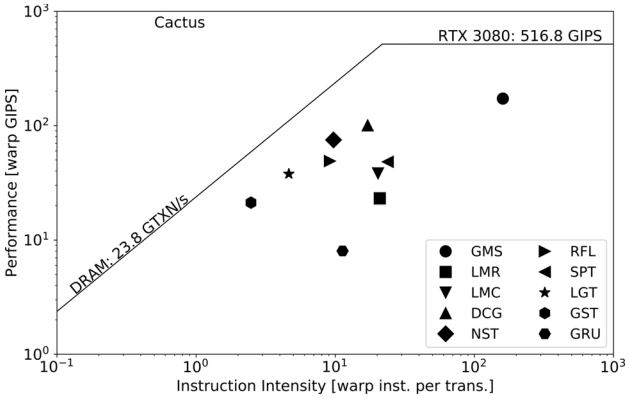
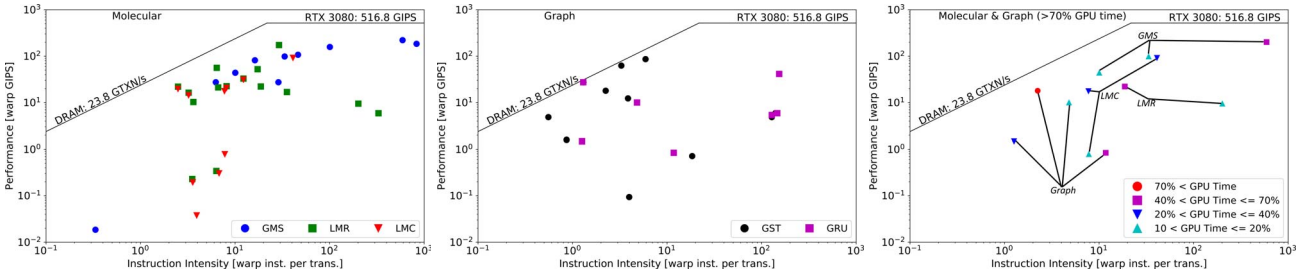


Fig. 5: Roofline model for the *Cactus* workloads: this includes all kernels and hence represents overall per-application performance. *Most of the Cactus workloads are primarily memory-intensive.*

in the roofline chart. Note that we include all kernels in this analysis, and we therefore characterize a workload's overall aggregate execution behavior across all kernels.

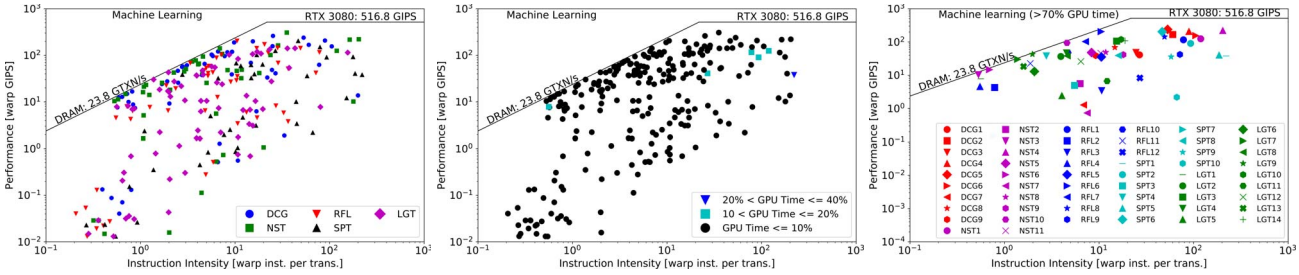
Observation #5: *The Cactus applications are primarily memory-intensive.* The roofline plot in Figure 5 clearly indicates that most of the *Cactus* applications are located on the left-hand side of the elbow point. The graph analytics workloads, *GST* and *GRU*, are clearly memory-intensive and achieve the lowest performance. The machine learning applications (*DCG*, *NST*, *RFL* and *LGT*) achieve higher performance and are memory-intensive, with *SPT* as the only exception although it is located close the memory/compute boundary. The molecular dynamics workloads are close to the boundary (*LMR* and *LMC*) while *GMS* is the only *Cactus* workload that is clearly on the compute-intensive side. Overall, we conclude that the *Cactus* applications are mostly memory-intensive.

3) Cactus: Per-Kernel Behavior: We now dive deeper and analyze per-kernel execution behavior. Figures 6a and 6b plot the roofline chart for all individual kernels for the *Cactus* molecular simulation and graph analytics applications, respec-



(a) Molecular simulation (b) Graph analytics (c) Most dominant kernels

Fig. 6: Roofline charts for the *Cactus* molecular simulation and graph analytics workloads: (a) all kernels from the molecular simulation applications, (b) all kernels from the graph analytics applications, and (c) the most dominant kernels from the molecular simulation and graph analytics workloads. *The Cactus applications feature both memory-intensive kernels as well as compute-intensive kernels.*



(a) All kernels color-coded by benchmark (b) All kernels color-coded by contribution (c) Dominant kernels per benchmark

Fig. 7: Roofline charts for the Cactus machine learning applications: (a) all kernels color-coded by benchmark; (b) all kernels color-coded by contribution to overall execution time; and (c) the dominant kernels color-coded by benchmark. *The Cactus machine learning applications feature many kernels with diverse execution characteristics, many of which are dominant and memory-bandwidth bound, i.e., they are close to the memory roof.*

tively; Figure 6c plots the kernels among these applications that contribute to at least 70% of the total execution time.

Observation #6: *The Cactus workloads include both memory-intensive kernels as well as compute-intensive kernels.* In contrast to the Parboil, Rodinia and Tango benchmarks, the *Cactus* applications feature a mix of memory-intensive and compute-intensive kernels. (We observe similar trends for the machine learning workloads in *Cactus* but we discuss those separately.) For the molecular simulation applications (Figure 6a), most of *GMS*' kernels are compute-intensive while some are memory-intensive; *LMR* and *LMC* mostly feature memory-intensive kernels although some are compute-intensive. For the graph workloads (Figure 6b), we also see that most of the kernels are memory-intensive, although there are compute-intensive kernels as well.

We note that the diversity in kernel characteristics is the case for even the dominant kernels, not just the non-dominant kernels, see Figure 6c. *GMS* features three dominant kernels of which two are compute-intensive and one is memory-intensive. One of *LMR*'s two dominant kernels is compute-intensive while the other one is memory-intensive. *LMC* features three dominant kernels out of which two are memory-intensive and one is compute-intensive. The graph analytics are an exception with all dominant kernels being memory-intensive.

The same analysis can be done for the *Cactus* machine learning applications, see Figure 7 in which we report all kernels (a) color-coded by benchmark, and (b) color-coded by contribution, as well as (c) the dominant kernels (>70% of total execution time) per benchmark.

Observation #7: *The Cactus machine learning applications feature many kernels with a wide diversity in compute- and memory-intensive behavior and performance.* We note from Figure 7a that all machine learning applications feature compute- and memory-intensive kernels with a wide diversity in performance. This is remarkably different from the Parboil, Rodinia and Tango benchmarks, as well as the Cactus molecular and graph workloads. We further observe from Figure 7b that a large fraction of the kernels contribute by less than 10% to the total execution time, however, this does not mean that they are not important, on the contrary. The four top-ranked kernels in *DCG*, *NST*, *RFL* and *SPT* are compute-intensive; only for *LGT* is the most dominant kernel memory-intensive. However, as we have observed before, the overall behavior of the machine learning applications is primarily memory-intensive, hence the lower-ranked kernels must be memory-intensive transferring the overall application behavior to the

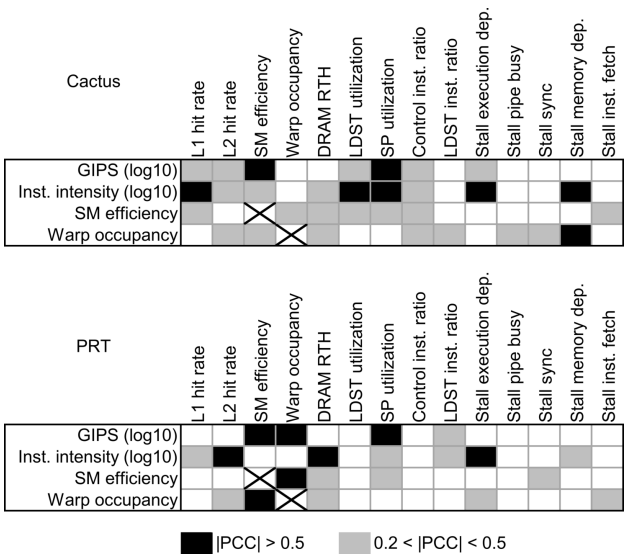


Fig. 8: Absolute values of the Pearson Correlation Coefficient between the primary performance metrics (rows) versus other performance metrics (columns) for (a) *Cactus* and (b) Parboil, Rodinia and Tango (PRT). *The execution behavior in Cactus is more complex.*

memory-intensive region.

Observation #8: *The Cactus machine learning applications feature dominant kernels that are memory-bandwidth-bound.* Figure 7c analyzes the dominant kernels that collectively contribute for more than 70% of the total application runtime. Generally speaking, all applications feature dominant kernels with low to high instruction intensity and low to high performance, spreading out across the memory-intensive and compute-intensive regions. It is specifically worth noting that a large number of the dominant kernels in the machine learning applications are limited by DRAM bandwidth as they are close to the memory roof. This suggests that machine learning performance can possibly be improved by providing higher memory bandwidth.

C. Correlation Analysis

We perform correlation analysis to better understand the behavioral characteristics of the *Cactus* workloads. We consider four primary performance metrics, namely GIPS, instruction intensity, SM efficiency and warp occupancy, and we correlate those to the performance characteristics listed in Table IV. The goal is to understand how the primary metrics are affected by various underlying performance phenomena. We use the Pearson Correlation Coefficient (PCC) to compute correlation. PCC varies between -1 and $+1$. A value close to zero means no correlation whereas a value close to -1 and $+1$ means strong negative and positive correlation, respectively. Figure 8 visualizes the absolute values of the correlation coefficient for *Cactus* versus Parboil, Rodinia and Tango. We use the following color code: black denotes strong correlation ($0.5 \leq |PCC| \leq 1$), gray means weak correlation ($0.2 \leq |PCC| < 0.5$), and white means no correlation ($0 \leq |PCC| < 0.2$)

Observation #9: *The execution behavior is more complex in Cactus compared to Rodinia, Parboil and Tango.* Figure 8 clearly illustrates that the correlation values are higher for *Cactus* than for Parboil, Rodinia and Tango. In particular, GIPS is correlated (strongly or weakly) with 7 performance metrics for *Cactus* versus only 4 for Parboil, Rodinia and Tango. The same applies to instruction intensity, SM efficiency and warp occupancy. More detailed analysis is warranted to truly understand the more complex behavior of the *Cactus* workloads. A couple of interesting observations can be made though. For example, we observe weak correlation for *Cactus* for all four primary metrics with the ratio of control instructions, in contrast to Parboil, Rodinia and Tango for which there is no correlation. A similar observation can be made for LD/ST utilization.

D. Hierarchical Clustering

Clustering is a data analysis technique that groups similar objects in a multi-dimensional space. In the context of this paper, the objects are kernels and the dimensions in the space are execution characteristics. The goal is to categorize kernels based on their (dis)similarity. More specifically, we use a combination of data analysis techniques: we first apply factor analysis to identify the most dominant dimensions in the data set, which then serves as input to hierarchical clustering. In contrast to prior work in workload characterization [2], [17], [24], [28], [49], [54] which uses principal component analysis (PCA) on quantitative data only, we use both quantitative and qualitative variables. We therefore rely on Factor Analysis of Mixed Data (FAMD) [34], [64] to act as a denoising method where the first few, most significant dimensions extract the essential information, and the last few, least significant dimensions are subject to noise. Applying hierarchical clustering on the most important dimensions provides a clustering outcome that is more stable than if we were to apply cluster analysis on the original execution characteristics.

The quantitative execution characteristics are the numerical variables from Table IV. The qualitative variables are derived from the roofline model. We consider two qualitative metrics: memory-intensive versus compute-intensive, and bandwidth-bound versus latency-bound. Whether a workload is classified as memory- versus compute-intensive is done based on its instruction intensity, i.e., applications on the left-hand side of the elbow point in the roofline model are memory-bound while the ones on the right-hand side are compute-bound. Classifying a workload as bandwidth- versus latency-bound is done based on its achieved performance. We choose a threshold of 1% of peak performance (5.16 GIPS) to label kernels, i.e., a kernel that achieves a performance number of less than 5.16 GIPS is labeled latency-bound whereas a kernel with performance above 5.16 GIPS is labeled bandwidth-bound.

Figure 9 shows the dendrogram as the final outcome of the hierarchical clustering considering the dominant kernels

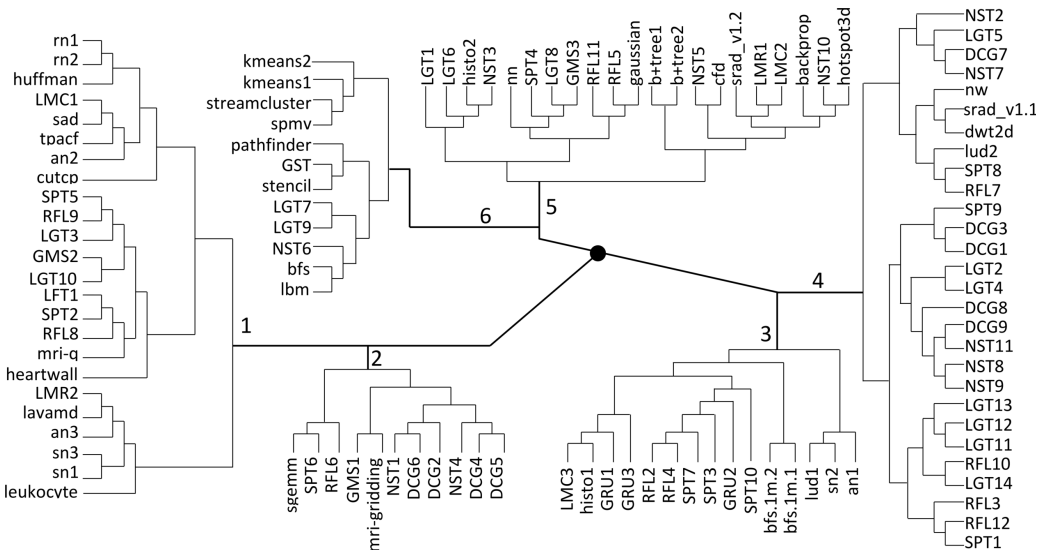


Fig. 9: Dendrogram visualizing the (dis)similarity among kernels from Cactus versus Rodinia, Parboil and Tango. *Kernels from the same Cactus application exhibit widely different execution behavior.*

from all benchmark suites. Benchmarks that are connected through short links are similar, whereas benchmarks that are far apart are dissimilar. We use a non-traditional visualization to optimize for space, in which the links from the central point to the six primary clusters are drawn to scale as a measure for their relative (dis)similarity. There are several interesting observations to be made here:

Observation #10: *The Parboil, Rodinia and Tango benchmarks exhibit unambiguous behavior.* The benchmarks consisting of a single kernel naturally belong to a single cluster. We observe that benchmarks that consist of a couple kernels belong to the same cluster as well, see for example *RN*, *Kmeans*, *B+tree*, *Bfs*. A couple of benchmarks have kernels that belong to at most two clusters. This is observed for benchmarks with two or three kernels, see for example *SN*, *AN*, *Histo*, *Lud* and *Srad_v1*. Overall, we conclude that benchmarks from existing suites exhibit limited execution diversity and kernels in such programs are limited to one or two clusters.

Observation #11: *Kernels from the same Cactus application are dissimilar from each other.* We note that different kernels from the same *Cactus* application exhibit widely different behavior, i.e., kernels are distributed across different clusters. This is particularly the case for the molecular simulation and machine learning workloads. More specifically, the *GMS* and *LMC* molecular simulation benchmarks have three dominant kernels with each kernel belonging to a different cluster. The machine learning workloads *LGT* and *NST* have dominant kernels in four different clusters; and *RFL* and *SPT* have kernels in five clusters. In other words, we find that the execution characteristics are very diverse across kernels from the same *Cactus* application (especially in molecular simulation and machine learning), in contrast to Rodinia, Parboil and Tango.

Observation #12: *While Rodinia, Parboil and Tango cover large parts of the workload space, Cactus covers an even larger part.* We observe Cactus, Rodinia, Parboil and Tango kernels in all six major clusters. However, clusters #2 and #4 are primarily dominated by *Cactus* kernels. This observation applies to some sub-clusters in cluster #3 as well. Note that the bottom sub-clusters of cluster #4 consist of *Cactus* kernels exclusively. This illustrates that *Cactus* covers a larger part of the entire workload space than Rodinia, Parboil and Tango collectively do. We hence conclude that *Cactus* exhibits more diverse execution behavior.

VI. Conclusion

This paper proposed the *Cactus* benchmark suite which is designed in a top-down fashion by selecting a number of representative GPU-compute applications from the HPC domain covering molecular dynamics simulation, graph analytics and machine learning. In contrast to the bottom-up benchmark suites prevalently used in the literature, we find that *Cactus* workloads execute many more kernels and exhibit execution behavior that is more diverse and more complex, while covering a broader range of the workload space. We hence believe that *Cactus* is a useful complement to the GPU-compute benchmarking toolbox by providing a focus on application-level GPU-compute performance representative of modern-day workloads. As part of our future work, we plan to extend *Cactus* by analyzing and including additional modern-day applications and by evaluating *Cactus* across a broader range of GPU platforms (both CUDA and OpenCL-based systems). In addition, we plan to create *Cactus* instruction traces that are compatible with state-of-the-art GPU simulators so that researchers can simulate *Cactus* workloads without requiring access to a real GPU device.

ACKNOWLEDGEMENTS

We thank the anonymous reviewers for their valuable feedback. This work is supported in part by the European Research Council (ERC) Advanced Grant agreement no. 741097, and the UGent-BOF GOA grant agreement BOF21-GOA-014.

APPENDIX

This artifact appendix summarizes how to conduct the experiments described in the paper using the Cactus benchmarks on an Nvidia RTX 3080 device. We have created a package that contains all program sources with their input data sets, installation scripts and run scripts. Additionally, the package contains the data files that were used to produce the results and figures reported in the paper.

- **Run-time environment:** Ubuntu 20.04.1, Kernel 5.4.0, CUDA 11.2 with driver 460.27, CuDNN 8.1.0, Nsight Compute 2020.3
- **Hardware:** Nvidia RTX 3080
- **Metrics:** Performance metrics available in Nvidia Nsight Compute
- **How much disk space required (approximately)?** 50 GB
- **How much time is needed to prepare workflow (approximately)?** 1 day (this includes downloading, setting up and installing the programs, as well as running the programs as a test and fixing any errors that may occur)
- **How much time is needed to complete experiments (approximately)?** 10–12 days (profiling is a time consuming task given the metrics that are being collected and the programs' execution times)
- **Publicly available?** Yes

A. How to access

Cactus package is publicly available at https://zenodo.org/record/5517569. The package contains all prerequisite software packages, including CUDA-11.2, CuDNN-8.1.0, program sources used in the work including Gromacs, Lammps, Gunrock and PyTorch as well as artifact-specific programs including Nsight Compute 2020.3, and the Rodinia, Parboil and Tango benchmark suites. The core package without prerequisite files is also available at Github and can be fetched via the command `git clone https://github.com/gpubench/cactus`.

B. Hardware dependencies

We have used Nvidia RTX 3080 as the physical device for the profiling experiments, although the experiments could be run on other Nvidia GPU devices as well. According to the manuals, a minimum version of CUDA-9 is sufficient for running the workloads. The benchmarks can also be configured to use OpenCL instead of CUDA according to the program manuals, but the workflow and current versions of the Cactus scripts focus on Nvidia devices.

C. Software dependencies

Cactus primarily needs GPU-related toolkits for developers. For Nvidia this is CUDA which contains both the driver and compiler. We used Ubuntu 20.04.1 with kernel 5.4, CUDA-11.2 with driver 460.72 and CuDNN-8.1.0. Other CUDA versions are also functional based on the documents.

D. Data sets

All programs and input data sets are publicly available and they are also available in the Cactus repository.

First, download the whole package from Zenodo or Github as described in A. Open `./scripts/common` with a text editor where there are four variables (with explanations) that you can change based on your needs. If you have a fresh Ubuntu 20.04.1 installation, only specify the architecture number of your Nvidia device and leave the other three variables with their default values. Then, execute `./setup.sh`. By default, the setup file will automatically install CUDA-11.2. More information about the setup script is available in `README.md`. The script will execute four scripts in `scripts/` to install Gromacs, Lammps, Gunrock and PyTorch. The programs will be compiled in order which means that if at any step a program fails to compile, the script exists with an error code. Since the package contains all files, the setup script skips download commands.

At this stage, we assume that all programs have been successfully installed with `./setup.sh`. The experiment itself consists of two main parts:

- Running the applications with the inputs to verify they are working properly on a GPU. To do that, navigate to `workloads/` where you can find the 10 workloads. Execute `./runme.sh` in each folder and verify they are working correctly.
- Profiling the applications with Nvidia profiler. Note that you can skip this part if you are not intended to profile the applications. To do that, navigate to `artifact/`. The `./profiler.sh` script contains profiling commands with the list of metrics used in the experiments. Please note that the script is ready to run, but it is recommended to execute the commands individually or splitting the metrics due to 1) the long runtime of profiling and 2) possible crashes due to using too many metrics at once. The script can be used as a template script if you want to collect other metrics. The folder also contains Rodinia, Parboil and Tango sources with installation and profiling scripts. The folder also contains Python and R scripts that use the data files for plotting and analyzing the results.

Depending on the specific hardware platform, profiling a large number of metrics is time consuming. For convenience, the final data files are present in `artifact/data/`. We also provide the Python and R scripts to parse the data files used to generate Figures 2 through 9 in the paper.

REFERENCES

[1] M. J. Abraham, T. Murtola, R. Schulz, S. Pall, J. C. Smith, B. Hess, and E. Lindahl. GROMACS: High performance molecular simulations through multi-level parallelism from laptops to supercomputers. *SoftwareX*, 1-2:19–25, 2015.

[2] V. Adhinarayanan and W. c. Feng. An automated framework for characterizing and subsetting GPGPU workloads. In *Proceedings of the International Symposium on Performance Analysis of Systems and Software*, pages 307–317, 2016.

[3] NVIDIA DGX A100 System Architecture. https://images.nvidia.com/aem-dam/en-zz/Solutions/data-center/dgx-a100/dgxa100-system-architecture-white-paper.pdf, 2020.

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

[5] D. A. Bader, H. Meyerhenke, P. Sanders, and D. Wagner. Graph partitioning and graph clustering. In *Proceedings of the 10th DIMACS Implementation Challenge Workshop*, 2012.

[6] D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. In *Proceedings of the International Conference on Learning Representations*, 2015.

[7] W. M. Brown, A. Kohlmeyer, S. J. Plimpton, and A. N. Tharrington. Implementing molecular dynamics on hybrid high performance computers – particle–particle particle-mesh. *Computer Physics Communications*, 183(3):449–459, 2012.

[8] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S.-H. Lee, and K. Skadron. Rodinia: A benchmark suite for heterogeneous computing. In *Proceedings of the International Symposium on Workload Characterization*, pages 44–54, 2009.

[9] cuBLAS: the CUDA Basic Linear Algebra Subroutine library. https://docs.nvidia.com/cuda/cublas/index.html, 2021.

[10] NVIDIA CUDA-X. https://www.nvidia.com/en-us/technologies/cuda-x/, 2021.

[11] H. Cui, H. Zhang, G. R. Ganger, P. B. Gibbons, and E. P. Xing. GeePS: scalable deep learning on distributed GPUs with a GPU-specialized parameter server. In *Proceedings of the European Conference on Computer Systems*, pages 1–16, 2016.

[12] A. Danalis, G. Marin, C. McCurdy, J. S. Meredith, P. C. Roth, K. Spafford, V. Tipparaju, and J. S. Vetter. The scalable heterogeneous computing (SHOC) benchmark suite. In *Proceedings of the 3rd Workshop on General-Purpose Computation on Graphics Processing Units*, pages 63–74, 2010.

[13] N. Ding and S. Williams. An instruction roofline model for GPUs. In *Proceedings of the International Conference on Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems*, pages 7–18, 2019.

[14] Gromacs download page. https://www.gromacs.org/Downloads, 2020.

[15] Gunrock download page. https://github.com/gunrock/gunrock, 2020.

[16] LAMMPS download page. https://www.lammps.org/download.html, 2020.

[17] L. Eeckhout, H. Vandierendonck, and K. De Bosschere. Workload design: selecting representative program-input pairs. In *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques*, pages 83–94, 2002.

[18] N. P. Jouppi et al. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the International Symposium on Computer Architecture*, pages 1–12, 2017.

[19] P. Mattson et al. MLPerf training benchmark. In *Proceedings of the International Conference on Machine Learning and Systems*, 2020.

[20] V. J. Reddi et al. MLPerf inference benchmark. In *Proceedings of the International Symposium on Computer Architecture*, pages 446–459, 2020.

[21] CUDA Templates for Linear Algebra Subroutines. https://github.com/NVIDIA/cutlass, 2021.

[22] Hydra: A framework for elegantly configuring complex applications. https://github.com/facebookresearch/hydra, 2019.

[23] L. A. Gatys, A. S. Ecker, and M. Bethge. Image style transfer using convolutional neural networks. In *Proceedings of the International Conference on Computer Vision and Pattern Recognition*, page 2414–2423, 2016.

[24] N. Goswami, R. Shankar, M. Joshi, and T. Li. Exploring GPGPU workloads: Characterization methodology, analysis and microarchitecture evaluation implications. In *Proceedings of the International Symposium on Workload Characterization*, pages 1–10, 2010.

[25] S. Grauer-Gray, L. Xu, R. Searles, S. Ayalasomayajula, and J. Cavazos. Auto-tuning a high-level language targeted to GPU codes. In *Proceedings of Innovative Parallel Computing*, pages 1–10, 2012.

[26] Flappy Bird hack using Deep Reinforcement Learning (Deep Q-learning). https://github.com/yenchenlin/DeepLearningFlappyBird, 2019.

[27] MNIST handwritten digit database. http://yann.lecun.com/exdb/mnist, 2010.

[28] B. Hu and C. J. Rossbach. Altis: Modernizing GPGPU benchmarks. In *Proceedings of the International Symposium on Performance Analysis of Systems and Software*, pages 1–11, 2019.

[29] M. Jaderberg, K. Simonyan, A. Zisserman, and K. Kavukcuoglu. Spatial transformer networks. In *Proceedings of the International Conference on Neural Information Processing Systems*, page 2017–2025, 2015.

[30] A. Karki, C. Palangotu Keshava, S. Mysore Shivakumar, J. Skow, G. Madhukeshwar Hegde, and H. Jeon. Tango: A deep neural network benchmark suite for various accelerators. *CoRR abs/1901.04987*, 2013.

[31] M. Kulkarni, M. Burtscher, C. Casçaval, and K. Pingali. Lonestar: A suite of parallel irregular programs. In *Proceedings of the International Symposium on Performance Analysis of Systems and Software*, pages 65–76, 2009.

[32] C. Kutzner, S. Páll, M. Fechner, A. Esztermann, B. L. de Groot, and H. Grubmüller. Best bang for your buck: GPU nodes for GROMACS biomolecular simulations. *Journal of Computational Chemistry*, 36(26):1990–2008, 2015.

[33] A. Lavin and S. Gray. Fast algorithms for convolutional neural networks. In *Proceedings of the International Conference on Computer Vision and Pattern Recognition*, pages 4013–4021, 2016.

[34] S. Le, J. Josse, and F. Husson. Factominer: An r package for multivariate analysis. *Journal of Statistical Software*, 25(1):1–18, 2008.

[35] Ignite: High level library to help with training, evaluating neural networks in PyTorch flexibly, and transparently. https://github.com/pytorch/ignite, 2019.

[36] P. Li, Y. Luo, N. Zhang, and Y. Cao. Heterospark: A heterogeneous CPU/GPU spark platform for machine learning algorithms. In *Proceedings of the Conference on Networking, Architecture and Storage*, pages 347–348, 2015.

[37] E. Lindholm, J. Nickolls, S. Oberman, and J. Montrym. NVIDIA Tesla: A unified graphics and computing architecture. *IEEE Micro*, 28(2):39–55, 2008.

[38] Z. Liu, P. Luo, X. Wang, and X. Tang. Deep learning face attributes in the wild. In *Proceedings of the International Conference on Computer Vision*, pages 3730–3738, 2015.

[39] D. Merrill, M. Garland, and A. Grimshaw. Lonestar: A suite of parallel irregular programs. In *Proceedings of the ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, page 117–128, 2012.

[40] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing atari with deep reinforcement learning. In *Proceedings of the International Conference on Neural Information Processing Systems, Workshop on Deep Learning*, 2013.

[41] A. Munshi. The OpenCL specification. In *Proceedings of the IEEE Hot Chips 21 Symposium*, 2009.

[42] J. Nickolls, I. Buck, M. Garland, and K. Skadron. Scalable parallel programming with cuda. *ACM Queue*, 6(2):40–53, 2008.

[43] PyTorch: An open source machine learning framework. https://pytorch.org, 2020.

[44] Benchmarking Deep Learning operations on different hardware. https://github.com/baidu-research/DeepBench, 2018.

[45] S. Pall, M. Ja. Abraham, C. Kutzner, B. Hess, and E. Lindahl. Tackling exascale software challenges in molecular dynamics simulations with GROMACS. In *Proceedings of the International Conference on Exascale Applications and Software*, pages 3–27, 2015.

[46] Y. Pan, R. Pearce, , and J. D. Owens. Scalable breadth-first search on a GPU cluster. In *Proceedings of the International Parallel and Distributed Processing Symposium*, pages 1090–1101, 2018.

[47] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. Automatic differentiation in PyTorch. In *NIPS Autodiff Workshop*, 2017.

[48] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. PyTorch: An imperative style, high-performance deep learning library. In *Proceedings of the International Conference on Neural Information Processing Systems*, pages 8024–8035, 2019.

[49] A. Phansalkar, A. Joshi, and L. K John. Analysis of redundancy and application balance in the SPEC CPU2006 benchmark suite. In *Proceedings of the International Symposium on Computer Architecture*, pages 412–423, 2007.

[50] A. Radford, L. Metz, and S. Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. In *Proceedings of the International Conference on Learning Representations*, 2016.

[51] V. T. Ravi, M. Becchi, G. Agrawal, and S. Chakradhar. Supporting GPU sharing in cloud environments with a transparent runtime consolidation framework. In *Proceedings of the Symposium on High Performance Distributed Computing*, pages 217–228, 2011.

[52] J. Redmon. Darknet: Open source neural networks in C. http://pjreddie.com/darknet/, 2013–2016.

[53] R. A. Rossi and N. K. Ahmed. The network data repository with interactive graph analytics and visualization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 4292–4293, 2015.

[54] J. H. Ryoo, S. J. Quirem, M. Lebeane, R. Panda, S. Song, and L. K. John. GPGPU benchmark suites: How well do they sample the performance spectrum? In *Proceeding of the International Conference on Parallel Processing*, pages 320–329, 2015.

[55] NVIDIA CUDA SDK Code Samples. https://developer.nvidia.com/cuda-code-samples, 2020.

[56] A. Shkurti, M. Orsi, E. Macii, E. Ficarra, and A. Acquaviva. Acceleration of coarse grain molecular dynamics on GPU architectures. *Journal of Computational Chemistry*, 34(10):803–818, 2013.

[57] A. Snell and L. Segervall. HPC applications support for GPU computing. Technical report, Intersect365 Research, 2017.

[58] J. A. Stratton, C. Rodrigues, I. Sung, N. Obeid, L. Chang, N. Anssari, G. D. Liu, and W. W Hwu. Parboil: A revised benchmark suite for scientific and commercial throughput computing. Technical Report IMPACT-12-01, Center for Reliable and High-Performance Computing, University of Illinois at Urbana-Champaign, 2012.

[59] PyTorch Tutorials. https://pytorch.org/tutorials, 2019.

[60] V. Volkov and J. W. Demmel. Benchmarking GPUs to tune dense linear algebra. In *Proceedings of the International Conference on Supercomputing*, pages 1–11, 2008.

[61] Y. Wang, A. Davidson, Y. Pan, Y. Wu, A. Riffel, and J. D. Owens. Gunrock: A high-performance graph processing library on the GPU. In *Proceedings of the ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pages 1–12, 2016.

[62] Y. Wang, P. Yangzihao, A. Davidson, Y. Wu, C. Yang, L. Wang, M. Osama, C. Yuan, W. Liua, A. T. Riffel, and J. D. Owens. Gunrock: GPU graph analytics. *ACM Transactions on Parallel Computing*, 4(2):1–49, 2017.

[63] B. Welton and B. Miller. Exposing hidden performance opportunities in high performance GPU applications. In *Proceedings of the International Symposium on Cluster, Cloud and Grid Computing*, pages 301–310, 2018.

[64] FactoMineR: Exploratory Multivariate Data Analysis with R. http://factominer.free.fr, 2021.

[65] Y. Wu, Y. Wang, Y. Pan, C. Yang, and J. D. Owens. Performance characterization for high-level programming models for GPU graph analytics. In *Proceedings of the International Symposium on Workload Characterization*, pages 66–75, 2015.

[66] Y. Xu, R. Wang, T. Li, M. Song, L. Gao, Z. Luan, and D. Qian. Scheduling tasks with mixed timing constraints in GPU-powered real-time systems. In *Proceedings of the International Conference on Supercomputing*, pages 1–13, 2016.

[67] P. Yalamanchili, U. Arshad, Z. Mohammed, P. Garigipati, P. Entschev, B. Kloppenborg, J. Malcolm, and J. Melonakos. ArrayFire: A high performance software library for parallel computing with an easy-to-use API. https://github.com/arrayfire/arrayfire, 2015.