Dwarfs on Accelerators: Enhancing OpenCL Benchmarking for Heterogeneous Computing Architectures

Beau Johnston and Josh Milthorpe

Research School of Computer Science, The Australian National University, Canberra, Australia
{beau.johnston, josh.milthorpe}@anu.edu.au

Abstract. For reasons of both performance and energy efficiency, high performance computing (HPC) hardware is becoming increasingly heterogeneous. The OpenCL framework supports portable programming across a wide range of computing devices and is gaining influence in programming next-generation accelerators, in particular, FPGAs. To characterize the performance of these devices across a range of applications requires a diverse, portable and configurable benchmark suite, and OpenCL is an attractive programming model for this purpose.
We present an extended and enhanced version of the OpenDwarfs OpenCL benchmark suite, with a strong focus placed on robustness of applications, curation of additional benchmarks with an increased emphasis on correctness of results and an increased problem size diversity within each application. Preliminary results and analysis are reported for eight benchmark codes on a representative set of current architectures – two Intel CPUs, five Nvidia GPUs and a Xeon Phi.

1 Introduction

High performance computing (HPC) hardware is becoming increasingly heterogeneous. A major motivation for this is to reduce energy use; indeed, without significant improvements in energy efficiency, the cost of exascale computing will be prohibitive. From June 2016 to June 2017, the average energy efficiency of the top 10 of the Green500 supercomputers rose by 2.3x, from 4.8 to 11.1 gigaflops per watt.[1] For many systems, this was made possible by highly energy-efficient Nvidia Tesla P100 GPUs. In addition to GPUs, future HPC architectures are also likely to include nodes with FPGA, DSP, ASIC and MIC components. A single node may be heterogeneous, containing multiple different computing devices; moreover, a HPC system may offer nodes of different types. For example, the Cori system at Lawrence Berkeley National Laboratory comprises 2,388 Cray XC40 nodes with Intel Haswell CPUs, and 9,688 Intel Xeon Phi nodes [2]. The Summit supercomputer at Oak Ridge National Laboratory is based on the IBM Power9 CPU, which includes both NVLINK [3], a high bandwidth interconnect between Nvidia GPUs; and CAPI, an interconnect to support FPGAs and

other accelerators. [4] Promising next generation architectures include Fujitsu's Post-K [5], and Cray's CS-400, which forms the platform for the Isambard supercomputer [6]. Both architectures use ARM cores alongside other conventional accelerators, with several Intel Xeon Phi and Nvidia P100 GPUs per node.

We seek to answer the question: for a given architecture with a variety of computing devices and a given set of computational tasks, which is the best choice of device for each task? Furthermore, does the best choice of device vary depending on whether the primary concern is time to solution or energy efficiency? While we do not yet have a satisfactory answers to these questions, our early investigations have yielded performance insights and valuable benchmarks, which we share in this paper.

As we are interested in comparing a diverse set of computing devices for HPC, we focus on the OpenCL programming model. OpenCL is supported on a wide range of systems including CPU, GPU and FPGA devices. While it is possible to write application code directly in OpenCL, it may also be used as a base to implement higher-level programming models. This technique was shown by Mitra et al., [7] where an OpenMP runtime was implemented over an OpenCL framework for Texas Instruments Keystone II DSP architecture. Having a common back-end in the form of OpenCL allows a direct comparison of identical code across diverse architectures, including FPGAs.

Our starting point is the OpenDwarfs benchmark suite, a set of OpenCL benchmarks for heterogeneous computing platforms.[8] OpenDwarfs characterizes benchmarks according to patterns of computation and communication known as the 13 Berkeley Dwarfs.[9] We extended OpenDwarfs with additional benchmarks taken from the Rodinia suite [10] (with modifications to improve portability), and with new benchmarks such as the two-dimensional discrete wavelet transform and a new FFT benchmark.

Marjanović et al. [11] argue that the selection of problem size for HPC benchmarking critically affects which hardware properties are relevant. We have observed this to be true across a wide range of accelerators, therefore we have enhanced the OpenDwarfs benchmark suite to support running different problem sizes for each benchmark. In this paper we report preliminary results for our enhanced version of OpenDwarfs on a range of platforms including CPU, GPU and MIC devices.

2 Enhancing the OpenDwarfs Benchmark Suite

Both Rodinia and the original OpenDwarfs benchmark suite focused on collecting a representative benchmarks for scientific applications, classified according to dwarfs, with a thorough diversity analysis to justify the addition of each benchmark to the corresponding suite.

We aim to extend these efforts to achieve a full representation of each dwarf, both by integrating other benchmark suites and adding custom kernels. At the same time, we hope to improve portability between devices, interpretability and flexibility of configuration including problem sizes.

For the Spectral Methods dwarf, the original OpenDwarfs version of the FFT benchmark was complex, with several code paths that were not executed for the default problem size, and returned incorrect results or failures on some combinations of platforms and problem sizes we tested. We replaced it with a simpler high-performance FFT benchmark created by Eric Bainville [12], which worked correctly in all our tests. We have also added a 2-D discrete wavelet transform from Rodinia, and we plan to add a continuous wavelet transform code.

As problem size can profoundly affect performance on accelerator systems [11], we enhanced configurability so that each benchmark could be run for a wide range of problem sizes.

Previous work [13] has shown that energy consumption of accelerator devices, at least in the embedded space and on compute-bound applications, is strongly correlated with execution time. Extending this investigation of energy usage to communication-bound and memory-bound problems is essential, since many scientific codes targeted for HPC and supercomputing systems have these characteristics. Additionally, given the candidate accelerators are proposed as future components on such systems, an evaluation of energy consumption on a wide range of accelerators is essential. To this end, LibSciBench (a performance measurement tool which allows high precision timing events to be collected for statistical analysis), complete with PAPI counters has been added to all of the applications in the OpenDwarfs Benchmark Suite. Through PAPI modules such as RAPL and NVML, LibSciBench also supports energy measurements, for which we report preliminary results in this paper.

3 Related Work

The NAS parallel benchmarks [14] follow a 'pencil-and-paper' approach, specifying the computational problem but leaving implementation choices such as language, data structures and algorithms to the user. The benchmarks include varied kernels and applications which allow a nuanced evaluation of a complete HPC system, however, the unconstrained approach does not readily support direct performance comparison between different hardware accelerators using a single set of codes.

Martineau et al. [15] collected a suite of benchmarks and three mini-apps to evaluate Clang OpenMP 4.5 support for Nvidia GPUs. Their focus was on comparison with CUDA; OpenCL was not considered.

The Scalable Heterogeneous Computing benchmark suite [16], unlike OpenDwarfs and Rodinia, supports multiple nodes using MPI for distributed parallelism. SHOC supports multiple programming models including OpenCL, CUDA and OpenACC, with benchmarks ranging from targeted tests of particular low-level hardware features to a handful of application kernels. By focusing on application kernels written exclusively in OpenCL, our enhanced OpenDwarfs benchmark suite is able to cover a wider range of application patterns.

Table 1: Hardware

Name	Vendor	Type	Series	Core Count	Clock Frequency (MHz) (min/max/turbo)	Cache (KiB) (L1/L2/L3)	TDP (W)	Launch Date
Xeon E5-2697 v2	Intel	CPU	Ivy Bridge	24*	1200/2700/3500	32/256/30720	130	Q3 2013
i7-6700K	Intel	CPU	Skylake	8*	800/4000/4300	32/256/8192	91	Q3 2015
Titan X	Nvidia	GPU	Pascal	3584†	1417/1531/NA	48/2048/NA	250	Q3 2016
GTX1080	Nvidia	GPU	Pascal	2560†	1607/1733/NA	48/2048/NA	180	Q2 2016
GTX1080Ti	Nvidia	GPU	Pascal	3584†	1480/1582/NA	48/2048/NA	250	Q1 2017
K20m	Nvidia	GPU	Kepler	2496†	706/?/NA	64/1536/NA	225	Q4 2012
K40m	Nvidia	GPU	Kepler	2880†	745/875/NA	64/1536/NA	235	Q4 2013
Xeon Phi 7210	Intel	MIC	KNL	256‡	1300/1500/NA	32/1024/NA	215	Q2 2016

* HyperThreaded cores
† CUDA cores
‡ Each physical core has 4 hardware threads per core, thus 64 cores

Barnes et al. [17] collected a representative set of applications from the current NERSC workload to guide optimization for Knights Landing in the Cori supercomputer. As it is not always feasible to perform such a detailed performance study of the capabilities of different computational devices for particular applications, the benchmarks described in this paper may give a rough understanding of device performance and limitations.

4 Experimental Setup

4.1 Hardware

The experiments were conducted on eight hardware platforms and are presented in Table 1. Two of these systems are Intel CPU architectures, five are current Nvidia GPUs and one is a MIC (Intel Knights Landing Xeon Phi). The L1 cache size should be read as having both an instruction size cache and a data cache size of equal values as those displayed. For Nvidia GPUs, the L2 cache size reported is the size per SM. For the Intel CPUs, hyper-threading was enabled and the frequency governor was set to 'performance'.

4.2 Software

OpenCL version 1.2 was used for all experiments. On the CPUs we used the Intel OpenCL driver version 6.3, provided in 16.1.1 and the 2016-R3 opencl-sdk release. On the GPU we used the Nvidia OpenCL driver version 375.66, provided as part of CUDA 8.0.61.

The Knights Landing (KNL) architecture used the same OpenCL driver as the Intel CPU platforms, however, the 2018-R1 release of the Intel compiler was used. This was required to compile for the architecture natively on the host. Additionally, due to Intel removing support for OpenCL on the KNL architecture, some additional compiler flags were required. Unfortunately, as Intel has removed support for AVX2 vectorization (using the '`-xMIC-AVX512`' flag), vector

instructions use only 256-bit registers instead of the wider 512-bit registers available on KNL. This means that floating-point performance on KNL is limited to half the theoretical peak.

GCC version 5.4.0 with GLIBC 2.23 was used for the Skylake i7 and GTX 1080, GCC version 4.8.5 with GLIBC 2.23 was used on the remaining platforms. OS Ubuntu Linux 16.04.4 with kernel version 4.4.0 was used for the Skylake CPU and GTX 1080 GPU, Red Hat 4.8.5-11 with kernel version 3.10.0 was used on the other platform.

As OpenDwarfs has no stable release version it was extended from the last commit by the maintainer on the Feb 26, 2016. [18] LibSciBench version 0.2.2 was used for all performance measurements.

4.3 Measurements

LibSciBench is a performance measurement tool which allows high precision timing events to be collected for statistical analysis [19]. It offers a high resolution timer in order to measure short running kernel codes, reported with one cycle resolution and roughly 6 ns of overhead. We used LibSciBench to record timings in conjunction with hardware events, which it collects via PAPI [20] counters.

We modified the applications in the OpenDwarfs benchmark suite to insert library calls to LibSciBench to record timings and PAPI events for the three main components of application time: kernel execution, host setup and memory transfer operations. To help understand the timings, the following hardware counters were also collected:

- total instructions and IPC (Instructions Per Cycle);
- L1 and L2 data cache misses;
- total L3 cache events in the form of request rate (requests/instructions) miss rate (misses/instructions) and miss ratio (misses/requests)
- data TLB (Translation Look-aside Buffer) miss rate (misses/instructions); and
- branch instructions and branch mispredictions.

A sample of 50 iterations was collected for each measurement. Only the kernel execution times/energy are presented in the final results.

Energy measurements were taken on Intel platforms using the Running Average Power Limit (RAPL) PAPI module. Similarly, a PAPI module was used in conjunction with the Nvidia Management Library (NVML) to measure the energy usage on Nvidia GPUs.

4.4 Setting Sizes

For each application, 4 different problem sizes were selected, namely **tiny**, **small**, **medium** and **large**. These problem sizes are based on the memory hierarchy of the Skylake CPU. Specifically, **tiny** should just fit within L1 cache, on the Skylake this corresponds to 32 KiB of data cache, **small** should fit within the

256 KiB L2 data cache, **medium** should fit within 8192 KiB of the L3 cache, and **large** must be much larger than 8192 KiB to avoid caching and operate out of main memory.

For this study, problem sizes were not customized to the memory hierarchy of each platform, since the CPU is the most sensitive to cache performance. Also, note for these CPU systems the L1 and L2 cache sizes are identical, and since we ensure that L3 is at least 4× larger than L3 for the largest spill over event, we are guaranteed to have L3 cache misses for the **large** problem size.

Caching performance was measured using PAPI counters. On the Skylake L1 and L2 data cache miss rates were counted using the `PAPI_L1_DCM` and `PAPI_L2_DCM` counters. For L3 miss events, only the total cache counter event (`PAPI_L3_TCM`) was available. The final values presented as miss results are presented as a percentage, and were determined using the number of misses counted divided by the total instructions (`PAPI_TOT_INS`).

The methodology to determine the appropriate size parameters is demonstrated on the k-means benchmark. K-means is an iterative algorithm which groups a set of points into clusters, such that each point is closer to the centroid of its assigned cluster than to the centroid of any other cluster. Each step of the algorithm assigns each point to the cluster with the closest centroid, then relocates each cluster centroid to the mean of all points within the cluster. Execution terminates when no clusters change size between iterations. Starting positions for the centroids are determined randomly. The OpenDwarfs benchmark previously required the object features to be read from a previously generated file. We extended the benchmark to support generation of a random distribution of points. This was done to more fairly evaluate cache performance, since repeated runs of clustering on the same feature space (loaded from file) would deterministically generate similar caching performance. The same number of clusters is fixed, with the minimum and maximum cluster locations being returned both set to 5.

Given a fixed number of clusters, the parameters that may be used to select a problem size are the number of points P_n, and the dimensionality or number of features per point F_n. In the kernel for k-means there are 3 large one dimensional arrays passed to the device, namely **feature**, **cluster** and **membership**. **feature** is the array which stores the unclustered feature space, it is of size $P_n \times F_n \times \text{sizeof}(\text{float})$. Each feature is represented by a 32-bit floating point number. **cluster** is the working and output array to store the intermediately clustered points, it is of size $C_n \times F_n \times \text{sizeof}(\text{float})$, where C_n is the number of clusters. **membership** is an array indicating whether each point has changed to a new cluster in each iteration of the algorithm, it is of size $P_n \times \text{sizeof}(\text{int})$, where sizeof(int) is the number of bytes to represent an integer value. Thereby the working kernel memory, in KiB, is:

$$\frac{\text{size}(\textbf{feature}) + \text{size}(\textbf{membership}) + \text{size}(\textbf{cluster})}{1024} \tag{1}$$

Since it is known for the target Skylake CPU that the L1 cache is of size 32 KiB, and using the theoretical size of all memory buffers determined in Equa-

Table 2: OpenDwarf workload scale parameters Φ

Benchmark	**tiny**	small	medium	large
`kmeans`	256	2048	65600	131072
`lud`	80	240	1440	4096
`csr`	736	2416	14336	16384
`fft`	2048	16384	524288	2097152
`dwt`	72x54	200x150	1152x864	3648x2736
`gem`	4TUT	2D3V	nucleosome	1KX5
`srad`	80,16	128,80	1024,336	2048,1024
`crc`	2000	16000	524000	4194304

tion 1, a careful experiment can be conducted around each benchmark and the appropriate amount of memory required. The tiny problem size is defined to have 256 points and 30 features; from Equation 1 the total size of the main arrays is 31.5 KiB, slightly smaller than the 32 KiB L1 cache. The number of points is increased for each larger problem size to ensure that the main arrays fit within the lower levels of the cache hierarchy, measuring the total execution time and respective caching events. The first row of table 2 shows the number of points for each problem size for k-means.

Being constrained in such a way indicates that 30 points/objects should give the best L1, L2 and L3 in the **tiny**, **small** and **medium** problem sizes respectively. Spilling over at each level of cache should be at 31 points/objects for each problem size. **large** operates solely on main memory. The large problem size is at least four times the size of the last-level cache - in the case of the Skylake, at least 32 MiB.

For brevity, cache miss results are not presented in this paper but were used to verify the suitable selection of all workload scales across all applications. The procedure to generate the remaining scale arguments is benchmark specific but was similar same as those presented above for k-means, until a suitably sized domain for each cache size was found. The final selected parameters are presented in Table 2.

The LU-Decomposition `lud`, Fast Fourier Transform `fft`, Speckle Reducing Anisotropic Diffusion `srad` and Cyclic Redundancy Check `crc` benchmarks do not require additional data sets, and so they have been extended and verified that setting the numerical arguments both generate the correct answer to each serial implementation and run on modern architectures.

2-Dimensional Discrete Wavelet Transform – `dwt` is commonly used in image compression. It has been extended to support loading of Portable PixMap (.ppm) and Portable GrayMap (.pgm) image format, and storing Portable GrayMap images of the resulting DWT coefficients in a visual tiled fashion. The input image dataset for various problem sizes was generated by using the resize capabilities of the ImageMagick application. The original gum leaf image is the large sample size has the ratio of 3648×2736 pixels and was down-sampled to 80×60 Gemnoui – `gem` is an n-body-method based benchmark which computes electrostatic

potential of biomolecular structures. Determining suitable problem sizes was performed by initially browsing the National Center for Biotechnology Information's Molecular Modeling Database (MMDB) [21] and inspecting the corresponding Protein Data Bank format (pdb) files. Molecules were then selected based on complexity, since the greater the complexity the greater the number of atoms required for the benchmark and thus the larger the memory footprint. **tiny** used the Prion Peptide 4TUT [22] and was the simplest structure, consisting of a single protein (1 molecule), it had the device side memory usage of 31.3 KiB which should fit in the L1 cache (32 KiB) on the Skylake processor. **small** used a Leukocyte Receptor 2D3V [23] also consisting of 1 protein molecule, with an associated memory footprint of 252KiB. **medium** used the nucleosome dataset originally provided in the OpenDwarf benchmark suite, using 7498 KiB of device-side memory. **large** used an X-Ray Structure of a Nucleosome Core Particle [24], consisting of 8 protein, 2 nucleotide, and 18 chemical molecules, and requiring 10 970.2 KiB of memory when executed by `gem`. Each `pdb` file was converted to the `pqr` atomic particle charge and radius format using the `pdb2pqr` [25] tool. Generation of the solvent excluded molecular surface used the tool `msms` [26]. The final datasets used for `gem` and all other benchmarks can be found in this papers associated GitHub repository [27].

Table 3: Program Arguments

Benchmark	Arguments
`kmeans`	`-g -f 26 -p` Φ
`lud`	`-s` Φ
Ψ	`createcsr -n` Φ `-d 5000`$^{\triangle}$
`csr`†	`-i` Ψ
`fft`	Φ
`dwt`	`-l 3` Φ`-gum.ppm`
`gem`	Φ 80 1 0
`srad`	Φ_1 Φ_2 `0 127 0 127 0.5 1`
`crc`	`-i 1000_`Φ`.txt`

$^{\triangle}$ The `-d 5000` indicates density of the matrix in this instance 0.5% dense (or 99.5% sparse).

† The `csr` benchmark loads a file generated by each respective workload of Φ, this file is generated by the `createcsr` and is represented by the variable Ψ.

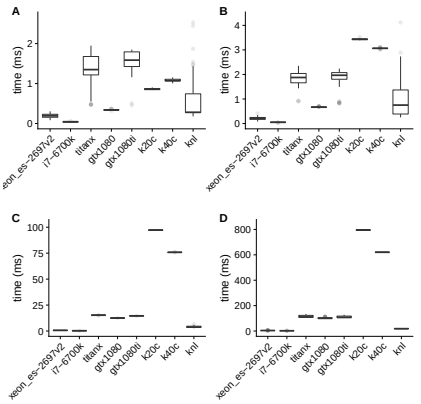


Fig. 1: Kernel Execution times for **crc**, the only application favouring the CPU architectures.

Where Φ is substituted as the argument for each benchmark, it is taken as the respective scale from Table 2 and is inserted into Table 3.

Each **Device** can be selected in a uniform way between applications using the same notation, on this system **Device** comprises of `-p 1 -d 0 -t 0` for the Intel Skylake CPU, where `p` and `d` are the integer identifier of the platform and

device to respectively use, and `-p 1 -d 0 -t 1` for the Nvidia Geforce GTX 1080 GPU. Each application is run as **Benchmark Device -- Arguments**, where **Arguments** is taken from Table 3 at the selected scale of Φ. For reproducibility the entire set of python scripts with all problem sizes is available in a Github repository [27].

5 Results

5.1 Time

Sample execution times from running 50 iterations per benchmark application. LibSciBench has a high resolution timer with one cycle resolution and roughly 6 ns of overhead. The top-left corner captions in all figures correspond to the 4 different workload sizes, such that **A** corresponds to **tiny**, **B** corresponds to **small**, **C** to **medium** and **D** to **large**.

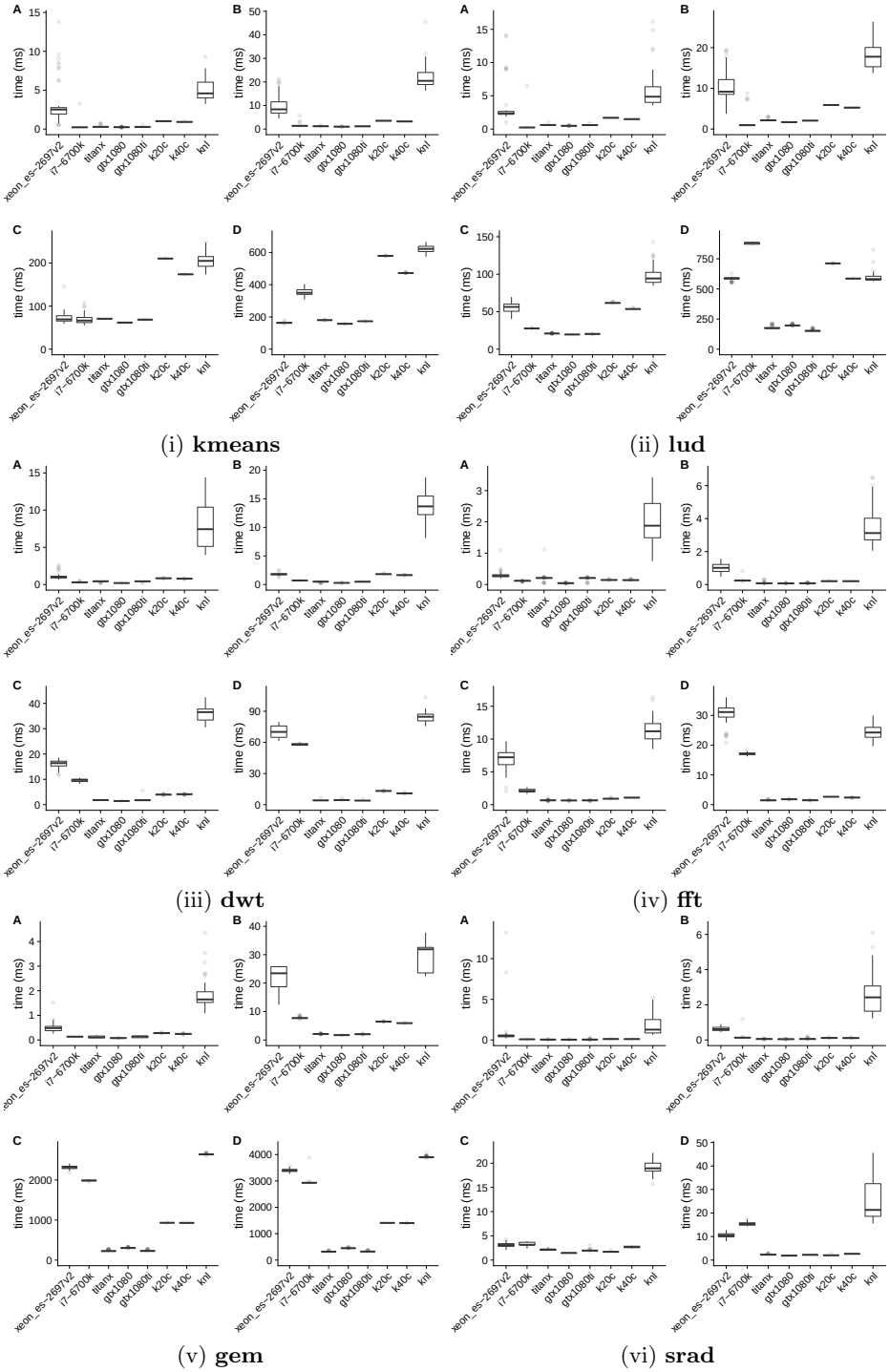
The first two sets of figures in a row have applications which both correspond to a particular dwarf, for instance Figure 2i and Figure 2ii are both representative of the Dense Linear Algebra dwarf, whereas Figure 2iii and Figure 2iv are both applications within Spectral Methods.

Figure 2v represents N-Body Methods, Figure 2vi encompasses a Structured Grid Method of Dwarf, and finally Figure 1 is a problem from Combinational Logic. All but the **crc** benchmark perform best on GPU type accelerators.

5.2 Energy

From the time results presented in Section 5.1 we see the largest difference occurs between CPU and GPU type accelerators at the **large** problem size. Thus, expected problem size which will show the largest difference in energy to also be **large**. This can be explored in Figures 3a and 3b where several applications are compared over the **large** size. All results are presented in joules. Each pair of box plots has been grouped by colour according to device, red for the Intel Skylake i7-6700k CPU and blue for the Nvidia GTX1080 GPU. These were the only devices examined since collection of RAPL and NVML PAPI performance measurements (with LibSciBench) requires super user access, and both these devices were the only accelerators available with this permission. Nonetheless, certain trends become apparent when directly comparing joules required over a range of problems. The logarithmic transformation has been applied to Figure 3b to emphasise the variation at the smaller energy scales (¡ 1 J), this will be remedied in the future by avoiding the small execution times to perform some of the applications, in particular all algorithms will be rebalanced with respect to the **gem** benchmark – which has the longest running time – to other **large** application sizes by adding an additional compute loop which will add redundancy but will standardise the magnitude of results.

The distributions were collected by measuring solely the kernel execution over a distribution of 50 runs. Variance with respect to energy usage is larger on the CPU, which is consistent to the execution time results. When comparing

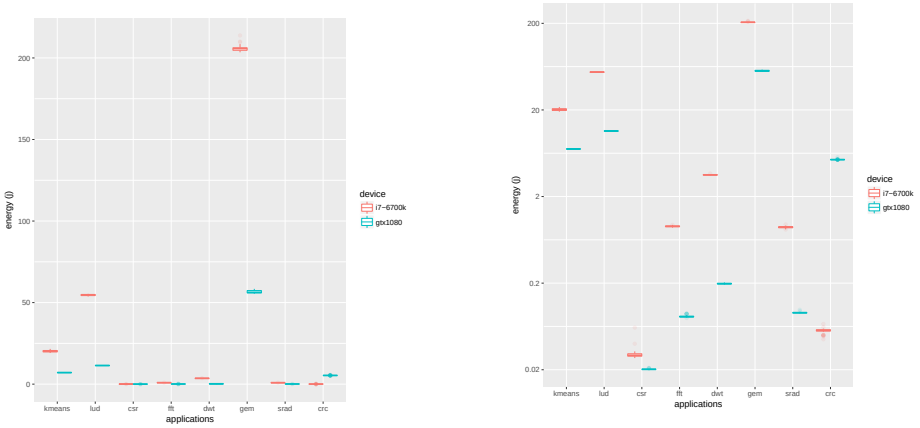


(i) **kmeans** (ii) **lud**

(iii) **dwt** (iv) **fft**

(v) **gem** (vi) **srad**

Fig. 2: Kernel execution times for applications where the GPU architectures are optimal



(a) Kernel Execution energy

(b) Logarithmic Kernel Execution energy

Fig. 3: Joules required to perform the kernel execution of all benchmark applications.

6 Conclusions

We have performed essential curation of the OpenDwarf benchmark suite. We improved coverage of spectral methods by adding a new Discrete Wavelet Transform benchmark, and replacing the previous inadequate `fft` benchmark. All benchmarks were enhanced to allow multiple problem sizes; in this paper we report results for four different problem sizes, selected according to the memory hierarchy of CPU systems as motivated by Marjanović's findings [11]. It is believed that these can now be quickly adjusted for next generation accelerator systems were each applications working memory will affect performance on these systems, this methodology was outlined in Section 4.4.

We ran many of the original benchmarks presented in the original OpenDwarfs [8] paper but on current hardware. This was done for two reasons, firstly to investigate the original findings to the state-of-the-art systems and secondly to extend the usefulness of the benchmark suite. Re-examining the original codes on range of modern hardware showed limitations, such as the fixed problem sizes along with many platform-specific optimizations (such as local work-group size). In the best case, such optimizations resulted in sub-optimal performance for newer systems (many problem sizes favored the original GPUs on which they were originally run). In the worst case, they resulted in failures when running on untested platforms or changed execution arguments.

Finally a major contribution of this work was to integrate LibSciBench into the benchmark suite, which adds a high precision timing library and support for statistical analysis and visualization. This has allowed collection of PAPI, energy and high resolution (sub-microsecond) time measurements at all stages of each application, which has added value to the analysis of OpenCL program flow on

each system, for example identifying overheads in kernel construction and buffer enqueuing.

7 Future Work

We plan to complete analysis of the remaining benchmarks in the suite for multiple problem sizes, and to add a benchmark for one of the 13 dwarfs that is not currently represented in the suite (MapReduce). In addition to comparing performance between devices, we would also like to develop some notion of 'ideal' performance for each combination of benchmark and device, which would guide efforts to improve performance portability.

Certain configuration parameters for the benchmarks, e.g. local workgroup size, are amenable to auto-tuning. We plan to integrate auto-tuning into the benchmarking framework to provide confidence that the optimal parameters are used for each combination of code and accelerator.

The original goal of this research was to discover methods for choosing the best device for a particular computational task, for example to support scheduling decisions under time and/or energy constraints. Until now, we found the available OpenCL benchmark suites were not rich enough to adequately characterize performance across the diverse range of applications and computational devices of interest. Now that a flexible benchmark suite is in place and results can be generated quickly and reliably on a range of accelerators, we plan to use these benchmarks to evaluate scheduling approaches.

References

1. Michael Feldman. Top500 meanderings: Supercomputers take big green leap in 2017. *TOP500 Supercomputer Sites*, Sep 2017.
2. Tina Declerck, Katie Antypas, Deborah Bard, Wahid Bhimji, Shane Canon, Shreyas Cholia, Helen Yun He, Douglas Jacobsen, and Nicholas J Wright Prabhat. Cori - a system to support data-intensive computing. *Proceedings of the Cray User Group*, page 8, 2016.
3. Timothy Morgan. NVLink takes GPU acceleration to the next level. *The Next Platform*, May 2016.
4. Timothy Morgan. The Power9 rollout begins with Summit and Sierra supercomputers. *The Next Platform*, Sep 2017.
5. Timothy Morgan. Inside Japan's future exascale ARM supercomputer, Jun 2016.
6. Michael Feldman. Cray to deliver ARM-powered supercomputer to UK consortium. *TOP500 Supercomputer Sites*, Jan 2017.
7. Gaurav Mitra, Eric Stotzer, Ajay Jayaraj, and Alistair P Rendell. Implementation and optimization of the OpenMP accelerator model for the TI Keystone II architecture. In *International Workshop on OpenMP*, pages 202–214. Springer, 2014.
8. Konstantinos Krommydas, Wu-chun Feng, Christos D Antonopoulos, and Nikolaos Bellas. OpenDwarfs: Characterization of dwarf-based benchmarks on fixed and reconfigurable architectures. *Journal of Signal Processing Systems*, 85(3):373–392, 2016.

9. Krste Asanovic, Ras Bodik, Bryan Christopher Catanzaro, Joseph James Gebis, Parry Husbands, Kurt Keutzer, David A Patterson, William Lester Plishker, John Shalf, Samuel Webb Williams, et al. The landscape of parallel computing research: A view from Berkeley. Technical report, Technical Report UCB/EECS-2006-183, EECS Department, University of California, Berkeley, 2006.
10. Shuai Che, Michael Boyer, Jiayuan Meng, David Tarjan, Jeremy W Sheaffer, Sang-Ha Lee, and Kevin Skadron. Rodinia: A benchmark suite for heterogeneous computing. In *Workload Characterization, 2009. IISWC 2009. IEEE International Symposium on*, pages 44–54. Ieee, 2009.
11. Vladimir Marjanović, José Gracia, and Colin W Glass. HPC benchmarking: problem size matters. In *International Workshop on Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS)*, pages 1–10. IEEE, 2016.
12. Eric Bainville. OpenCL fast Fourier transform, 2010.
13. Beau Johnston, Brian Lee, Luke Angove, and Alistair Rendell. Embedded accelerators for scientific high-performance computing: An energy study of OpenCL Gaussian elimination workloads. In *International Conference on Parallel Processing Workshops (ICPPW)*, pages 59–68. IEEE, 2017.
14. David H Bailey, Eric Barszcz, John T Barton, David S Browning, Robert L Carter, Leonardo Dagum, Rod A Fatoohi, Paul O Frederickson, Thomas A Lasinski, Rob S Schreiber, et al. The NAS parallel benchmarks. *International Journal of Supercomputing Applications*, 5(3):63–73, 1991.
15. Matt Martineau, Simon McIntosh-Smith, Carlo Bertolli, Arpith C Jacob, Samuel F Antao, Alexandre Eichenberger, Gheorghe-Teodor Bercea, Tong Chen, Tian Jin, Kevin O'Brien, et al. Performance analysis and optimization of Clang's OpenMP 4.5 GPU support. In *International Workshop on Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS)*, pages 54–64. IEEE, 2016.
16. M Graham Lopez, Jeffrey Young, Jeremy S Meredith, Philip C Roth, Mitchel Horton, and Jeffrey S Vetter. Examining recent many-core architectures and programming models using SHOC. In *International Workshop on Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS*, page 3. ACM, 2015.
17. Taylor Barnes, Brandon Cook, Jack Deslippe, Douglas Doerfler, Brian Friesen, Yun He, Thorsten Kurth, Tuomas Koskela, Mathieu Lobet, Tareq Malas, et al. Evaluating and optimizing the NERSC workload on Knights Landing. In *International Workshop on Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS*, pages 43–53. IEEE, 2016.
18. OpenDwarfs (base version). `https://github.com/vtsynergy/OpenDwarfs/commit/31c099aff5343e93ba9e8c3cd42bee5ec536aa93`, Feb 2017.
19. Torsten Hoefler and Roberto Belli. Scientific benchmarking of parallel computing systems: Twelve ways to tell the masses when reporting performance results. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, page 73. ACM, 2015.
20. Philip J Mucci, Shirley Browne, Christine Deane, and George Ho. PAPI: A portable interface to hardware performance counters. In *Proceedings of the Department of Defense HPCMP users group conference*, volume 710, 1999.
21. Thomas Madej, Christopher J Lanczycki, Dachuan Zhang, Paul A Thiessen, Renata C Geer, Aron Marchler-Bauer, and Stephen H Bryant. MMDB and VAST+: tracking structural similarities between macromolecular complexes. *Nucleic Acids Research*, 42(D1):D297–D303, 2013.

22. Lu Yu, Seung-Joo Lee, and Vivien C Yee. Crystal structures of polymorphic prion protein β1 peptides reveal variable steric zipper conformations. *Biochemistry*, 54(23):3640–3648, 2015.
23. Mitsunori Shiroishi, Mizuho Kajikawa, Kimiko Kuroki, Toyoyuki Ose, Daisuke Kohda, and Katsumi Maenaka. Crystal structure of the human monocyte-activating receptor, "Group 2" leukocyte Ig-like receptor A5 (LILRA5/LIR9/ILT11). *Journal of Biological Chemistry*, 281(28):19536–19544, 2006.
24. Curt A Davey, David F Sargent, Karolin Luger, Armin W Maeder, and Timothy J Richmond. Solvent mediated interactions in the structure of the nucleosome core particle at 1.9Å resolution. *Journal of Molecular Biology*, 319(5):1097–1113, 2002.
25. Todd J Dolinsky, Jens E Nielsen, J Andrew McCammon, and Nathan A Baker. PDB2PQR: an automated pipeline for the setup of Poisson–Boltzmann electrostatics calculations. *Nucleic Acids Research*, 32(suppl_2):W665–W667, 2004.
26. Michel F Sanner, Arthur J Olson, and Jean-Claude Spehner. Reduced surface: an efficient way to compute molecular surfaces. *Biopolymers*, 38(3):305–320, 1996.
27. Beau Johnston. OpenDwarfs. `https://github.com/BeauJoh/OpenDwarfs`, 2017.