Dwarfs on Accelerators: Enhancing OpenCL Benchmarking for Heterogeneous Computing Architectures

Beau Johnston
Research School of Computer Science
Australian National University
Canberra, Australia
beau.johnston@anu.edu.au

Josh Milthorpe
Research School of Computer Science
Australian National University
Canberra, Australia
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. 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 – three Intel CPUs, five Nvidia GPUs, six AMD 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.[10] 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 [7]. The Summit supercomputer at Oak Ridge National Laboratory is based on the IBM Power9 CPU, which includes both NVLINK [22], a high bandwidth interconnect between Nvidia GPUs; and CAPI, an interconnect to support FPGAs and other accelerators. [23] Promising next generation architectures include Fujitsu's Post-K [21], and Cray's CS-400, which forms the platform for the Isambard supercomputer [9]. Both architectures use ARM cores alongside other conventional accelerators, with several Intel Xeon Phi and Nvidia P100 GPUs per node.

Given this heterogeneity of hardware and the wide diversity of scientific application codes, workload characterization, performance prediction and scheduling are all becoming more challenging. To evaluate different approaches requires a representative benchmark suite which is portable to a wide variety of devices. We focus on the OpenCL programming model as it 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. [20] 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.

In this paper, we present an extended version of the OpenDwarfs benchmark suite, a set of OpenCL benchmarks for heterogeneous computing platforms.[15] We added new benchmarks to improve the diversity of the suite, and made a number of modifications aimed at improving the reproducibility and interpretability of results, portability between devices, and flexibility of configuration including problem sizes. We report preliminary results for a subset of the enhanced OpenDwarfs suite on a range of platforms including CPU, GPU and MIC devices.

2 ENHANCING THE OPENDWARFS BENCHMARK SUITE

The OpenDwarfs benchmark suite comprises a variety of OpenCL codes, classified according to patterns of computation and communication known as the 13 Berkeley Dwarfs.[1]

Beau Johnston and Josh Milthorpe

The original suite focused on collecting representative benchmarks for scientific applications, 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.

Marjanović et al. [18] 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. To improve reproducibility of results, we also modified each benchmark to execute in a loop for a minimum of two seconds, to ensure that sampling of execution time and performance counters was not significantly affected by operating system noise.

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 [3], which worked correctly in all our tests. We have also added a 2-D discrete wavelet transform from the Rodinia suite [5] (with modifications to improve portability), and we plan to add a continuous wavelet transform code.

To understand benchmark performance, it is useful to be able to collect hardware performance counters associated with each timing segment. LibSciBench is a performance measurement tool which allows high precision timing events to be collected for statistical analysis [12]. 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 [24] 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. Through PAPI modules such as Intel's Running Average Power Limit (RAPL) and Nvidia Management Library (NVML), LibSciBench also supports energy measurements, for which we report preliminary results in this paper.

3 RELATED WORK

The NAS parallel benchmarks [2] 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. [19] 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 (SHOC) [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. Sun et al. [28] propose Hetero-Mark, a Benchmark Suite for CPU-GPU Collaborative Computing, which has 5 benchmark applications each implemented in HCC – which compiles to OpenCL, HIP – for a CUDA and Radeon Open Compute back-end, and a CUDA version. Meanwhile, Chai by Gómez-Luna et al. [11], offers 15 applications in 7 different implementations with the focus on supporting integrated architectures. These benchmark suites centre on a comparison between languages, environments and the corresponding support on heterogeneous architectures; whereas our work focuses on benchmarking for device specific performance limitations by examining the problem sizes where these limitations occur – this is largely ignored by benchmarking suites with fixed problem sizes.

Additionally, our enhanced OpenDwarfs benchmark suite aims to cover a wider range of application patterns by focusing exclusively on OpenCL using higher-level benchmarks.

Barnes et al. [4] 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 14 hardware platforms and are presented in Table 1. Three of these systems are Intel CPU architectures, five are current Nvidia GPUs, five are AMD 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 L2 cache per SM multiplied by the number of SMs. 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.

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
i5-3550	Intel	CPU	Ivy Bridge	4∗	1600/3380/3700	32/256/6144	77	Q2 2012
Titan X	Nvidia	GPU	Pascal	3584†	1417/1531/–	48/2048/–	250	Q3 2016
GTX 1080	Nvidia	GPU	Pascal	2560†	1607/1733/–	48/2048/–	180	Q2 2016
GTX 1080 Ti	Nvidia	GPU	Pascal	3584†	1480/1582/–	48/2048/–	250	Q1 2017
K20m	Nvidia	GPU	Kepler	2496†	706/–/–	64/1536/–	225	Q4 2012
K40m	Nvidia	GPU	Kepler	2880†	745/875/–	64/1536/–	235	Q4 2013
HD 7970	AMD	GPU	Tahiti	2048‖	925/1010/–	16/768/–	250	Q4 2011
R9 290X	AMD	GPU	Hawaii	2816‖	1000/–/–	16/1024/–	250	Q3 2014
R9 295x2	AMD	GPU	Hawaii	5632‖	1018/–/–	16/1024/–	500	Q2 2014
FirePro S9150	AMD	GPU	Hawaii	2816‖	900/–/–	16/1024/–	235	Q3 2014
R9 Fury X	AMD	GPU	Fuji	4096‖	1050/–/–	16/2048/–	273	Q2 2015
RX 480	AMD	GPU	Polaris	4096‖	1120/1266/–	16/2048/–	150	Q2 2016
Xeon Phi 7210	Intel	MIC	KNL	256‡	1300/1500/–	32/1024/–	215	Q2 2016

∗ HyperThreaded cores
† CUDA cores
‖ Stream processors
‡ Each physical core has 4 hardware threads per core, thus 64 cores

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 26 Feb 2016. [25] LibSciBench version 0.2.2 was used for all performance measurements.

4.3   Measurements

We measured execution time and energy for individual OpenCL kernels within each benchmark. Each benchmark was run 50 times for each problem size (see §4.4) for both execution time and energy measurements.

A sample size of 50 per group – for each combination of benchmark and problem size – was used to ensure that sufficient statistical power $\beta = 0.8$ would be available to detect a significant difference in means on the scale of half standard deviation of separation. This sample size was computed using the t-test power calculation over a normal distribution.

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.

For each benchmark we also measured memory transfer times between host and device, however, only the kernel execution times and energies are presented here.

Energy measurements were taken on Intel platforms using the RAPL PAPI module, and on Nvidia GPUs using the NVML PAPI module.

IWOCL '18, 14-16 May, 2018, St Catherine's College, Oxford. UK

Beau Johnston and Josh Milthorpe

4.4 Setting Sizes

For each benchmark, four different problem sizes were se-
lected, namely **tiny**, **small**, **medium** and **large**. These prob-
lem 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.

The memory footprint was verified for each benchmark by
printing the sum of the size of all memory allocated on the
device.

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 **large** is at least 4× larger than L3 cache,
we are guaranteed to have last-level 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 pa-
rameters is demonstrated on the k-means benchmark.

4.4.1 kmeans.
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 bench-
mark 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. For all problem sizes, the number of clusters is
fixed at 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 three large one-dimensional
arrays passed to the device, namely **feature**, **cluster** and
membership. **feature** is the array which stores the un-
clustered feature space, it is of size $P_n \times F_n \times \text{sizeof (float)}$.
Each feature is represented by a 32-bit floating point number.

cluster is the working and output array to store the interme-
diately clustered points, it is of size $C_n \times F_n \times \text{sizeof (float)}$,
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 $P_n \times \text{sizeof (int)}$,
where sizeof (int) is the number of bytes to represent an in-
teger value. Thereby the working kernel memory, in KiB,
is:

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

Using this equation, we can determine the largest problem
size that will fit in each level of cache. 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 **tiny**,
small and **medium** problem sizes in the first row of Table 2
correspond to L1, L2 and L3 cache respectively. 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 – to ensure
that data are transferred between main memory and cache.

For brevity, cache miss results are not presented in this
paper but were used to verify the selection of suitable problem
sizes for each benchmark. The procedure to select problem
size parameters is specific to each benchmark, but follows a
similar approach to k-means.

4.4.2 lud, fft, srad, crc.
The LU-Decomposition `lud`, Fast
Fourier Transform `fft`, Speckle Reducing Anisotropic Diffu-
sion `srad`, and Cyclic Redundancy Check `crc` benchmarks
did not require additional data sets. Where necessary these
benchmarks were modified to generate the correct solution
and run on modern architectures. Correctness was exam-
ined either by directly comparing outputs against a serial
implementation of the codes (where one was available), or by
adding utilities to compare norms between the experimental
outputs.

4.4.3 dwt.
Two-Dimensional Discrete Wavelet Transform
is commonly used in image compression. It has been ex-
tended 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.

4.4.4 gem.
Gemnoui is an n-body-method based bench-
mark 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) [17]
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 [29] 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 [27] also consisting of 1 protein molecule, with an associated memory footprint of 252KiB. **medium** used the nucleosome dataset originally provided in the OpenDwarfs benchmark suite, using 7498 KiB of device-side memory. **large** used an X-Ray Structure of a Nucleosome Core Particle [6], 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` [8] tool. Generation of the solvent excluded molecular surface used the tool `msms` [26]. The datasets used for `gem` and all other benchmarks can be found in this paper's associated GitHub repository [13].

4.4.5 Summary of Benchmark Settings. The problem size parameters for all benchmarks are presented in Table 2.

Table 2: OpenDwarfs 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

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 [13]. 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.

5 RESULTS

5.1 Time

We first present execution time measurements for each benchmark, starting with the Cyclic Redundancy Check `crc` benchmark which represents the Combinational Logic dwarf.

Table 3: Program Arguments

Benchmark	Arguments
kmeans	`-g -f 26 -p` Φ
lud	`-s` Φ
csr†	`-i` Ψ
	$\Psi = $ `createcsr -n` Φ `-d 5000` $\triangle$
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 `createcsr` according to the workload size parameter Φ; this file is represented by Ψ.

Figure 1 shows the execution times for the `crc` benchmark over 50 iterations on each of the target architectures, including the KNL. Execution times are lowest on CPU-type architectures, probably due to the low floating-point intensity of the CRC computation [14, Ch. 6]. Excluding `crc`, all the other benchmarks perform best on GPU type accelerators; furthermore, the performance on the KNL is poor due to the lack of support for wide vector registers in Intel's OpenCL SDK. We therefore omit results for KNL for the remaining benchmarks.

Figures 2 and 3 shows the distribution of kernel execution times for the remaining benchmarks. Each benchmark corresponds to a particular dwarf: Figure 2a (`kmeans`) represents the MapReduce dwarf, Figure 2b (`lud`) represents the Dense Linear Algebra dwarf, Figure 2c (`csr`) represents Sparse Linear Algebra, Figure 2d (`dwt`) and Figure 2e (`fft`) represent Spectral Methods, Figure 3a (`gem`) represents N-Body Methods, and Figure 3b (`srad`) represents the Structured Grid dwarf.

For most benchmarks, the variance in execution times is much greater on the CPU platforms than on GPUs. While execution time increases with problem size for all benchmarks and platforms, the modern GPUs (Titan X, GTX1080 and GTX1080i) performed relatively better for large problem sizes, possibly due to their greater second-level cache size compared to the other platforms. A notable exception is `k-means` for which CPU execution times were comparable to GPU, which reflects the relatively low ratio of floating point to memory operations in the benchmark.

5.2 Energy

In addition to execution time, we are interested in differences in energy consumption between devices and applications. We measured the energy consumption of benchmark kernel
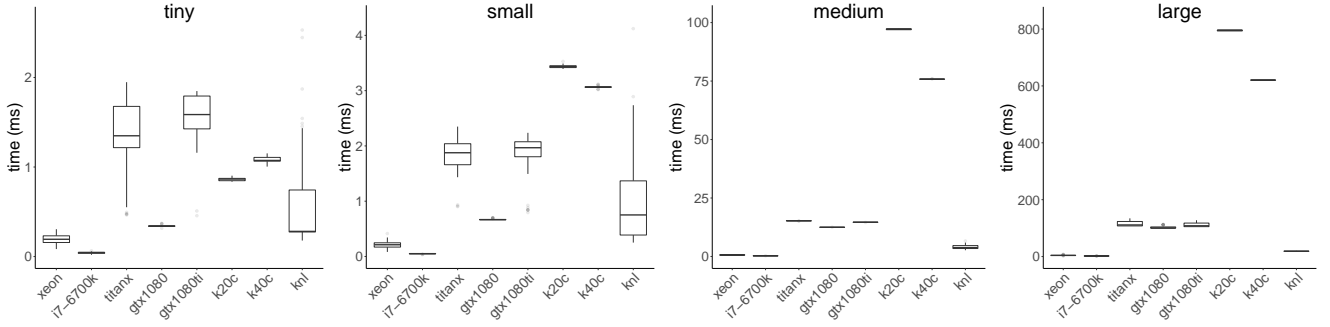
Figure 1: Kernel execution times for the crc benchmark on different hardware platforms, including KNL

execution on the Intel Skylake i7-6700k CPU and the Nvidia GTX1080 GPU, using PAPI modules for RAPL and NVML. These were the only devices examined since collection of PAPI energy measurements (with LibSciBench) requires superuser access, and these devices were the only accelerators available with this permission. The distributions were collected by measuring solely the kernel execution over a distribution of 50 runs. RAPL CPU energy measurements were collected over all cores in package 0 `rapl:::PP0_ENERGY:PACKAGE0`. NVML GPU energy was collected using the power usage readings `nvml:::GeForce_GTX_1080:power` for the device and presents the total power draw ($+/-5$ watts) for the entire card – memory and chip. Measurements results converted to energy J from their original resolution nJ and mW on the CPU and GPU respectively.

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 we expect that the **large** problem size will also show the largest difference in energy.

Figures 4a and 4b show the kernel execution energy for several benchmarks for the **large** size. All results are presented in joules. The box plots are coloured according to device: red for the Intel Skylake i7-6700k CPU and blue for the Nvidia GTX1080 GPU. The logarithmic transformation has been applied to Figure 4b to emphasise the variation at smaller energy scales ($< 1\,$J), which was necessary due to small execution times for some benchmarks. In future this will be addressed by balancing the amount of computation required for each benchmark, to standardize the magnitude of results.

All the benchmarks use more energy on the CPU, with the exception of `crc` which as previously mentioned has low floating-point intensity and so is not able to make use of the GPU's greater floating-point capability. Variance with respect to energy usage is larger on the CPU, which is consistent with the execution time results.

6 CONCLUSIONS

We have performed essential curation of the OpenDwarfs 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 [18]. 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 [15] 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. 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. Additional architectures such as FPGA, DSP and Radeon Open Compute based APUs – which further breaks down the walls between the CPU and GPU – will be considered.
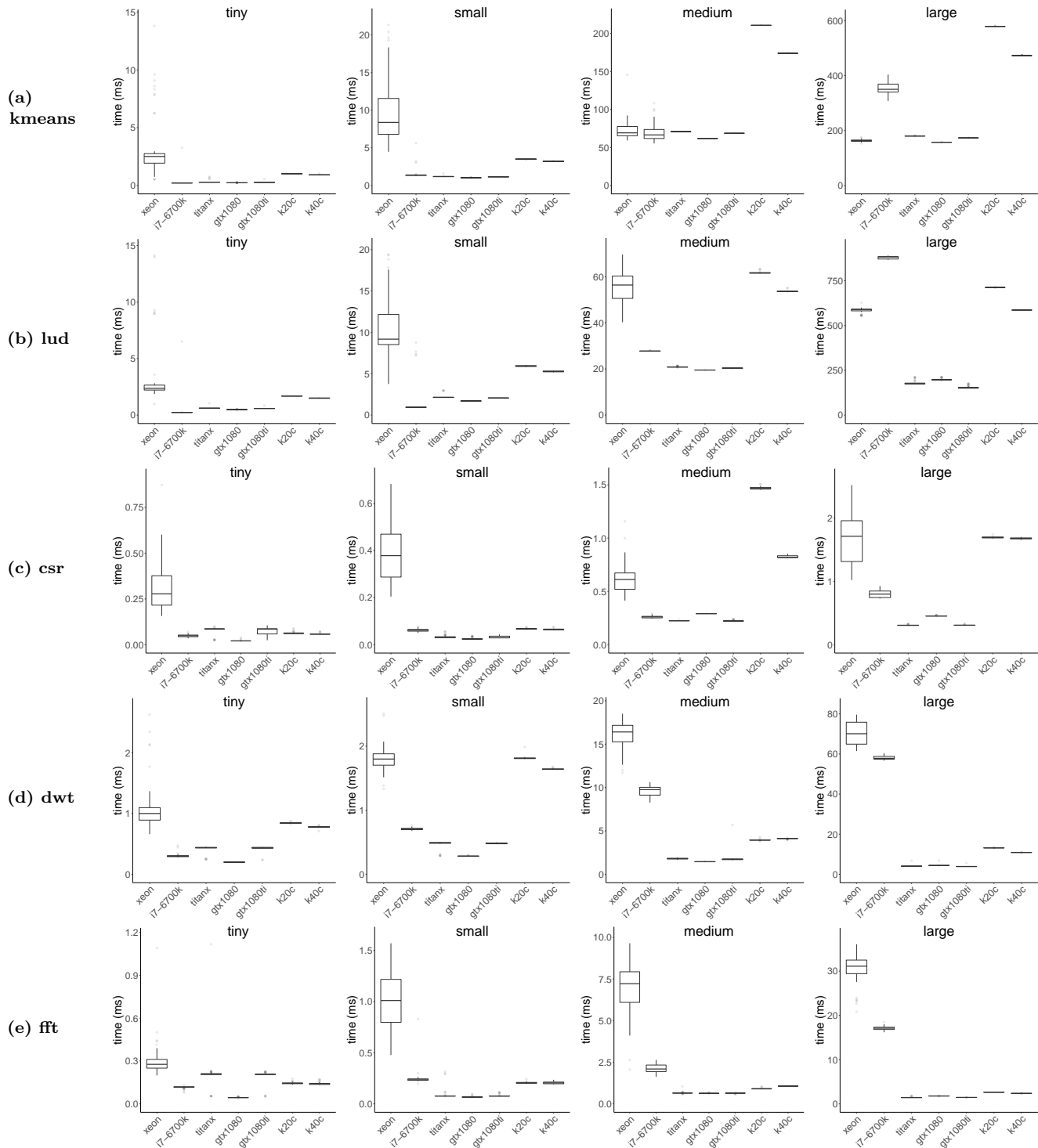
Figure 2: Benchmark kernel execution times on different hardware platforms
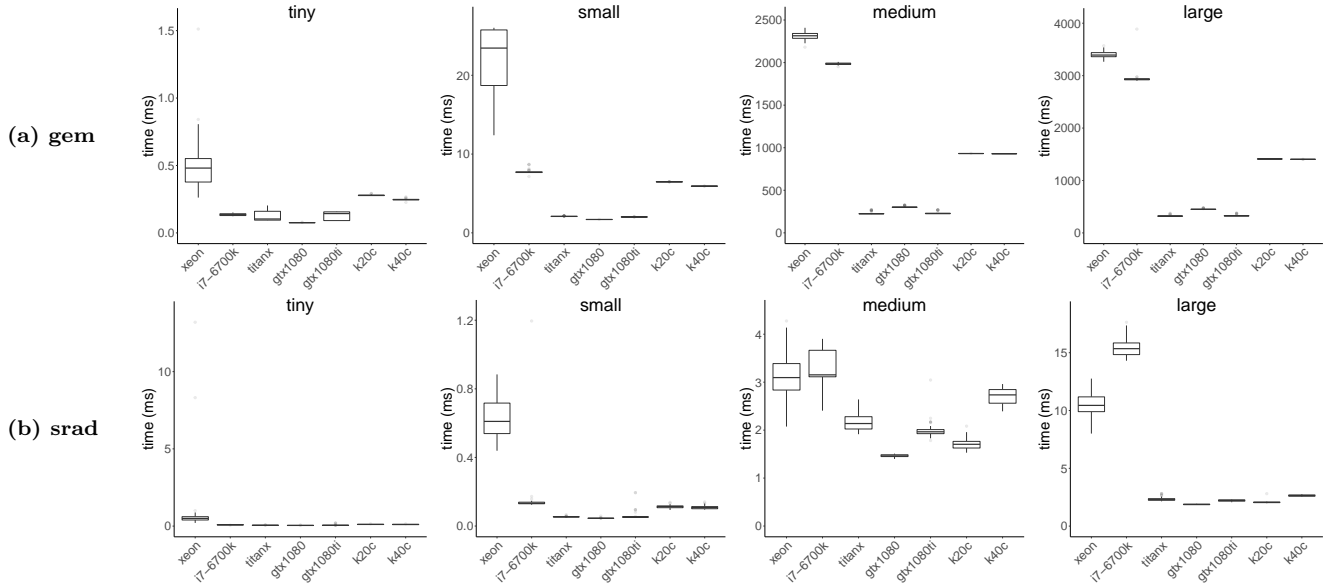
Figure 3: Benchmark kernel execution times on different hardware platforms (continued)

(a) Kernel execution energy
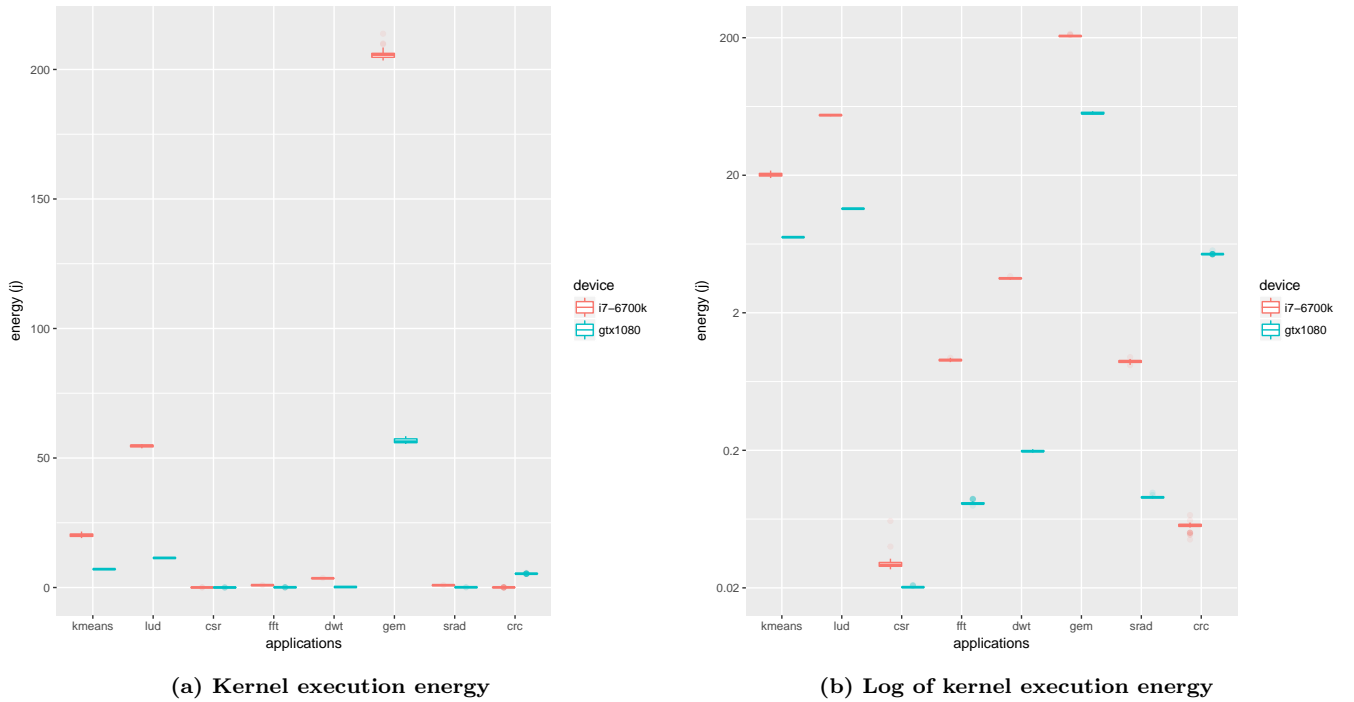
(b) Log of kernel execution energy

Figure 4: Benchmark kernel execution energy (large problem size) on Core i7-6700K and Nvidia GTX1080

Each OpenCL kernel presented in this paper has been inspected using the Architecture Independent Workload Characterization (AIWC). Analysis using AIWC helps understand how the structure of kernels contributes to the varying runtime characteristics between devices that are presented in this work, and will be published in the future.

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.

ACKNOWLEDGEMENTS

REFERENCES

[1] Krste Asanović, Ras Bodik, Bryan Christopher Catanzaro, Joseph James Gebis, Parry Husbands, Kurt Keutzer, David A Patterson, William Lester Plishker, John Shalf, Samuel Webb Williams, and Kathy Yelick. 2006. *The landscape of parallel computing research: A view from Berkeley*. Technical Report UCB/EECS-2006-183. EECS Department, University of California, Berkeley.

[2] 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. 1991. The NAS parallel benchmarks. *Internat. J. Supercomputing Applic.* 5, 3 (1991), 63–73.

[3] Eric Bainville. 2010. OpenCL Fast Fourier Transform. (2010). http://www.bealto.com/gpu-fft.html

[4] Taylor Barnes, Brandon Cook, Jack Deslippe, Douglas Doerfler, Brian Friesen, Yun He, Thorsten Kurth, Tuomas Koskela, Mathieu Lobet, Tareq Malas, et al. 2016. Evaluating and optimizing the NERSC workload on Knights Landing. In *International Workshop on Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS)*. IEEE, 43–53.

[5] Shuai Che, Michael Boyer, Jiayuan Meng, David Tarjan, Jeremy W Sheaffer, Sang-Ha Lee, and Kevin Skadron. 2009. Rodinia: A benchmark suite for heterogeneous computing. In *IEEE International Symposium on Workload Characterization (IISWC)*. IEEE, 44–54.

[6] Curt A Davey, David F Sargent, Karolin Luger, Armin W Maeder, and Timothy J Richmond. 2002. Solvent mediated interactions in the structure of the nucleosome core particle at 1.9Å resolution. *Journal of Molecular Biology* 319, 5 (2002), 1097–1113.

[7] Tina Declerck, Katie Antypas, Deborah Bard, Wahid Bhimji, Shane Canon, Shreyas Cholia, Helen Yun He, Douglas Jacobsen, and Nicholas J Wright Prabhat. 2016. Cori - A system to support data-intensive computing. *Proceedings of the Cray User Group* (2016), 8.

[8] Todd J Dolinsky, Jens E Nielsen, J Andrew McCammon, and Nathan A Baker. 2004. PDB2PQR: an automated pipeline for the setup of Poisson–Boltzmann electrostatics calculations. *Nucleic Acids Research* 32, suppl_2 (2004), W665–W667.

[9] Michael Feldman. 2017. Cray to Deliver ARM-Powered Supercomputer to UK Consortium. *TOP500 Supercomputer Sites* (Jan 2017). https://www.top500.org/news/cray-to-deliver-arm-powered-supercomputer-to-uk-consortium/

[10] Michael Feldman. 2017. TOP500 Meanderings: Supercomputers Take Big Green Leap in 2017. *TOP500 Supercomputer Sites* (Sep 2017). https://www.top500.org/news/top500-meanderings-supercomputers-take-big-green-leap-in-2017/

[11] Juan Gómez-Luna, Izzat El Hajj, Victor Chang, Li-Wen Garcia-Flores, Simon Garcia de Gonzalo, Thomas Jablin, Antonio J Pena, and Wen-mei Hwu. 2017. Chai: Collaborative Heterogeneous Applications for Integrated-architectures. In *Performance Analysis of Systems and Software (ISPASS), 2017 IEEE International Symposium on*. IEEE.

[12] Torsten Hoefler and Roberto Belli. 2015. 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*. ACM, 73.

[13] Beau Johnston. 2017. OpenDwarfs. https://github.com/BeauJoh/OpenDwarfs. (2017).

[14] Anagha S Joshi. 2016. *A performance focused, development friendly and model aided parallelization strategy for scientific applications*. Master's thesis. Clemson University.

[15] Konstantinos Krommydas, Wu-Chun Feng, Christos D Antonopoulos, and Nikolaos Bellas. 2016. OpenDwarfs: Characterization of dwarf-based benchmarks on fixed and reconfigurable architectures. *Journal of Signal Processing Systems* 85, 3 (2016), 373–392.

[16] M Graham Lopez, Jeffrey Young, Jeremy S Meredith, Philip C Roth, Mitchel Horton, and Jeffrey S Vetter. 2015. 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)*. ACM, 3.

[17] Thomas Madej, Christopher J Lanczycki, Dachuan Zhang, Paul A Thiessen, Renata C Geer, Aron Marchler-Bauer, and Stephen H Bryant. 2013. MMDB and VAST+: tracking structural similarities between macromolecular complexes. *Nucleic Acids Research* 42, D1 (2013), D297–D303.

[18] Vladimir Marjanović, José Gracia, and Colin W Glass. 2016. HPC benchmarking: problem size matters. In *International Workshop on Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS)*. IEEE, 1–10.

[19] 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. 2016. 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)*. IEEE, 54–64.

[20] Gaurav Mitra, Eric Stotzer, Ajay Jayaraj, and Alistair P Rendell. 2014. Implementation and optimization of the OpenMP accelerator model for the TI Keystone II architecture. In *International Workshop on OpenMP*. Springer, 202–214.

[21] Timothy Morgan. 2016. Inside Japan's Future Exascale ARM Supercomputer. (Jun 2016). https://www.nextplatform.com/2016/06/23/inside-japans-future-exaflops-arm-supercomputer/

[22] Timothy Morgan. 2016. NVLink Takes GPU Acceleration To The Next Level. *The Next Platform* (May 2016). https://www.nextplatform.com/2016/05/04/nvlink-takes-gpu-acceleration-next-level/

[23] Timothy Morgan. 2017. The Power9 Rollout Begins With Summit And Sierra Supercomputers. *The Next Platform* (Sep 2017). https://www.nextplatform.com/2017/09/19/power9-rollout-begins-summit-sierra/

[24] Philip J Mucci, Shirley Browne, Christine Deane, and George Ho. 1999. PAPI: A portable interface to hardware performance counters. In *Proceedings of the Department of Defense HPCMP users group conference*, Vol. 710.

[25] OpenDwarfs. 2017. OpenDwarfs (base version). https://github.com/vtsynergy/OpenDwarfs/commit/31c099aff5343e93ba9e8c3cd42bee5ec536aa93. (26 Feb 2017).

[26] Michel F Sanner, Arthur J Olson, and Jean-Claude Spehner. 1996. Reduced surface: an efficient way to compute molecular surfaces. *Biopolymers* 38, 3 (1996), 305–320.

[27] Mitsunori Shiroishi, Mizuho Kajikawa, Kimiko Kuroki, Toyoyuki Ose, Daisuke Kohda, and Katsumi Maenaka. 2006. Crystal structure of the human monocyte-activating receptor, "Group 2" leukocyte Ig-like receptor A5 (LILRA5/LIR9/ILT11). *Journal of Biological Chemistry* 281, 28 (2006), 19536–19544.

[28] Yifan Sun, Xiang Gong, Kavyan Amir, Amir Kavyan Ziabari, Leiming Yu, Xiangyu Li, Saoni Mukherjee, Carter McCardwell, Alejandro Villegas, and David Kaeli. 2016. Hetero-Mark, A Benchmark Suite for CPU-GPU Collaborative Computing. (09 2016).

[29] Lu Yu, Seung-Joo Lee, and Vivien C Yee. 2015. Crystal structures of polymorphic prion protein $\beta1$ peptides reveal variable steric zipper conformations. *Biochemistry* 54, 23 (2015), 3640–3648.