OpenCL and the 13 Dwarfs: A Work in Progress[*]

W. Feng, H. Lin, T. Scogland, and J. Zhang
Department of Computer Science
Virginia Tech
{feng, hlin2, njustn, zjing14}@cs.vt.edu

ABSTRACT

In the past, evaluating the architectural innovation of parallel computing devices relied on a benchmark suite based on existing programs, e.g., EEMBC or SPEC. However, with the growing ubiquity of parallel computing devices, we argue that it is unclear how best to express parallel computation, and hence, a need exists to identify a higher level of abstraction for reasoning about parallel application requirements. Therefore, the goal of this combination "Work-in-Progress and Vision" paper is to delineate application requirements in a manner that is not overly specific to individual applications or the optimizations used for certain hardware platforms, so that we can draw broader conclusions about hardware requirements. Our initial effort, dubbed "OpenCL and the 13 Dwarfs" or OCD for short, realizes Berkeley's 13 computational dwarfs of scientific computing in OpenCL, where each dwarf captures a pattern of computation and communication that is common to a class of important applications.

Categories and Subject Descriptors: *D.0 [General]; I.6.3 [Simulation & Modeling]: Applications; J.0 [General]*

General Terms: Algorithms, Benchmarking, Measurement, Experimentation.

Keywords: computational dwarfs, OpenCL, GPU, heterogeneous computing, portability.

1. INTRODUCTION

The increasing proliferation of heterogeneous computing platforms presents the parallel computing community with the challenge of evaluating the efficacy of such parallel architectures, particularly given the diversity of hardware architectures and their associated (non-interoperable) programming environments such as Cilk+ and CUDA. For instance, the graphics processing unit (GPU), which has become an increasingly popular processor, differs substantially from traditional CPU architectures. The GPU offers simpler SIMD-like processing elements to deliver extraordinary performance for data-parallel and task-parallel jobs. Its ability to support massive multi-threaded parallelism indicates its capability as a high-throughput processor versus the low-latency CPU, which is optimized for single-threaded performance.

Performance benchmark suites have been playing an important role in evaluating hardware design. However, traditional parallel benchmark suites have serious shortcomings when trying to evaluate evolving heterogeneous computing systems. First, such suites are written in programming models designed for CPUs and thus cannot be run directly on the emergent heterogeneous architectures. Second, such suites typically focus on concrete implementations of specific applications. Those applications are not necessarily sufficient for capturing future trends in parallel computing or for comprehensively exercising new heterogeneous architectures.

To address the above issues, we present *OpenCL and 13 Dwarfs* or OCD for short, a benchmark suite that aims to provide a "future-proofed" software methodology to enable the evaluation of hardware innovation across a gamut of architectures. We choose to use OpenCL because it is a standardized industry effort addressing the lack of interoperability in heterogeneous programming models. While it began as a programming model for programming GPUs, and optionally falling back on CPUs, the major processor vendors — including AMD, ARM, IBM, Intel, and NVIDIA — have either released or are developing OpenCL compilers and runtime systems. Using OpenCL as our programming model of choice will enable our benchmark suite to work well across a wide range of platforms today and into the future.

In addition, we seek to enable a fundamental re-thinking of both hardware and programming models by capturing application design via high-level computation and communication patterns. To this end, we select application kernels following computation and communication patterns from the *Berkeley 13 Dwarfs* [2]. We focus on these because they offer a diverse set of patterns, each of which is relevant across a variety of domains. For example, the n-body method is relevant across physics, chemistry, and a variety of other domains.

We are populating each dwarf with an application as a starting point. However, because no single application completely captures the breadth of a dwarf, our longer-term intent is to include multiple applications that present different aspects of a given dwarf as well as a synthetic benchmark that represents the dwarf alone (without other patterns included) to form a full application. In this way, we hope to create a set of implementations which may be used to make

[*]This work was supported in part by NSF I/UCRC IIP-0804155 (via NSF CHREC) and a DoD National Defense Science & Engineering Graduate Fellowship (NDSEG).

generalizations about the higher-level patterns and the effectiveness of a given platform for executing a given pattern.

The initial release of OCD is meant to provide functionally portable benchmarks in OpenCL and allow users to draw conclusions based on the performance of portable code. To accomplish this, we have made an effort to avoid optimizing any given benchmark specifically for a given underlying platform, and instead, focus on writing to the programming model.[1] The result is that more reasonable performance comparisons across different architectures are possible.

2. THE 13 DWARFS

Below is a brief description of each of the 13 Berkeley dwarfs, along with a description of our initial instantiation of the dwarf in OCD, if applicable.

Dense linear algebra consists of dense matrix and vector operations. It has a high ratio of math-to-load operations and a high degree of data interdependency between threads. We are finalizing a benchmark for this dwarf based on LU factorization, but for the time being, we include the k-means clustering algorithm, denoted as kmeans in OCD.

Sparse linear algebra solves the same problem as dense linear algebra but has matrices with few non-zero entries. To reduce space and computation, such algorithms store and operate on a list of values and indices rather than proper matrices, resulting in more indirect memory accesses. For OCD, we implement a pattern for matrix-vector multiplication that uses a *compressed spare row* format to store sparse matrices. As such, the implemented dwarf is denoted as csr.

Spectral methods transform data from/to either a spatial or temporal domain. The execution profile is typically characterized by multiple stages of processing, where dependencies within a stage form a "butterfly" pattern of computation. We capture this pattern via a FFT, i.e., clfft in OCD.

N-body methods calculate interactions between many discrete points and are characterized by large numbers of independent calculations within a timestep, followed by all-to-all communication between timesteps. Our GEM code [1], denoted as gemnoui in OCD, calculates the effect that all atoms have on the charge at each point along the surface of a molecule, leading to $O(M * N)$ complexity where N is atoms and M is points along the surface.

Structured grids organize data in a regular multidimensional grid, where computation proceeds as a series of grid updates. For each grid update, all points are updated using values from a small neighborhood around each point. The neighborhood is normally implicit in the data and determined by the algorithm. For OCD, we include srad, short for *speckle-reducing anisotropic diffusion*, a stencil-based pattern of computation and communication that reduces noise and enhances feature clarity in 2D images.

Unstructured grids possess data structures, e.g., linked list of pointers, that keep track of the location and 'neighborhood' of points which are used to update the location. Like sparse linear algebra, updates typically involve multiple levels of memory reference indirection, as an update to any point requires first determining a list of neighboring points, and then loading values from those neighboring points. For OCD, we include a pattern of computation and communication that is representative of an unstructured grid code for computational fluid dynamics, denoted as cfd in OCD.

[1] Similarly, other benchmark suites write to MPI or a general CPU rather than to Intel SSE4 instructions, for example.

MapReduce captures the repeated independent execution of a "map" function and results are aggregated at the end via a "reduce" function. No communication is required between processes in the map phase, but the reduce phase requires global communication. For OCD, we have a prototype dwarf that we dub StreamMR ("streamer").

Combinational logic exploits bit-level parallelism in order to achieve high throughput. Such a workload involves performing simple operations on very large amounts of data. For OCD, we include crc, short for cyclic redundancy check, which is used to generates hashes or signatures of files to verify their correct transfer over a network.

Graph traversal visits and evaluates a number of objects in a graph. Such algorithms typically involve a significant amount of random memory access for indirect lookups. The bottleneck is generally due to access latency rather than access bandwidth. For OCD, we include breadth-first search (bfs) and bitonic sort (bsort).

Dynamic programming solves a complex problem by solving a series of simpler subproblems. For OCD, we adopt the Needleman-Wunsch algorithm, i.e., needle in OCD. This algorithmic pattern calculates the optimal alignment of two strings by calculating scores based on all possible alignments in a matrix and backtracking along the highest scoring path.

Backtrack & branch-and-bound approaches generally search a very large search space to find a globally optimal solution. Because the search space is so large, an implicit method is needed to prune the search space to make this approach computationally tractable. For OCD, we capture the computation and communication pattern of the A* search algorithm (astar in OCD).

Graphical models map variables into nodes and conditional probabilities into edges, e.g., Bayesian networks. For OCD, we have captured this pattern of computation and communication via a hidden Markov model.

Finite state machines capture a system whose behavior is defined by states, transitions defined by inputs and the current state, and events associated with transitions or states. These dwarf algorithms are highly dependent on conditional operations and interdependent data, which are also commonly found in graph traversal. For OCD, we provide a "temporal data mining" algorithm, which discovers temporal correlations between EEG events from the brain.

3. EXPERIENCES WITH OCD

This section presents our experiences with *OpenCL and the 13 Dwarfs (OCD)* across a myriad of CPU and GPU computing platforms.

3.1 Experimental Setup

For all of our experiments, we ran OCD on a single test box consisting of two quad-core Intel Xeon E5405 CPUs, 4GB of DDR3 memory, and at any one time, a single GPU. We physically swapped between an AMD HD5450, AMD HD5870, NVIDIA GT520, and NVIDIA C2050 to ensure comparable results between the different GPU platforms. The software environment is x86_64 Ubuntu 10.04 with Linux kernel 2.6.32, using GCC 4.4.3 along with NVIDIA SDK 4.0, AMD APP SDK 2.5 RC2 and Intel OpenCL SDK 1.5. The drivers used are AMD Catalyst 11.11 and NVIDIA 290.10.

3.2 Runtime Diversity

In theory, OpenCL performance on a single piece of hardware ought to be consistent regardless of the runtime sys-

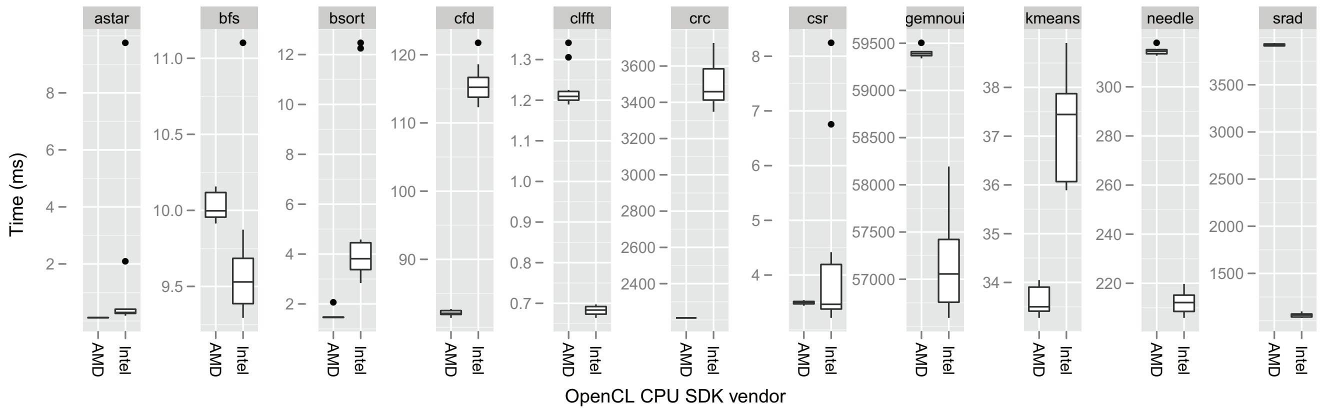

Figure 1: Time for each benchmark on two quad-core Intel Xeon CPUs across OpenCL CPU runtimes from AMD and Intel for all implemented dwarfs. Each box represents the interquartile range with a line at the median, whiskers reach to the data point closest to 1.5x the interquartile range outside of the middle 50 percent without going over, and dots are those points which fall past that mark.

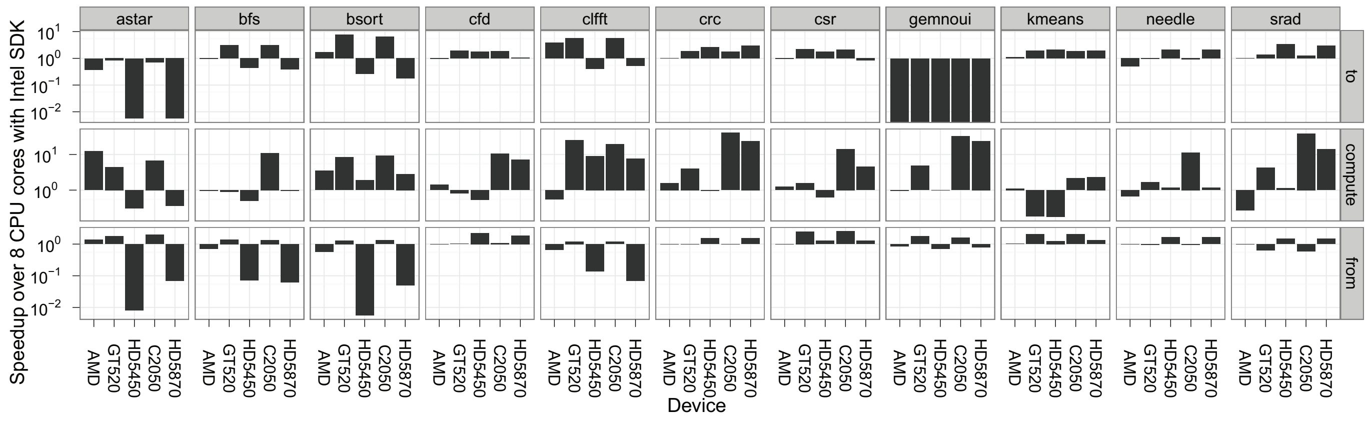

Figure 2: Speedup of the three segments of each application, i.e., data transfer to the device, computation on the device and data transfer back from it, over the Intel OpenCL SDK on eight Intel Xeon cores.

tem; but in practice, this is rarely the case. To illustrate this point, we compared the performance of our OCD across two different SDKs, namely the AMD OpenCL SDK and the new Intel OpenCL SDK, while running on *identical* hardware. Figure 1 shows the empirical results.

Since our experimental platform consisted of a pair of Intel Xeon CPUs, the initial expectation was that Intel's SDK would either match AMD's or outperform it for each case, but that was not the case. For example, the *combinational logic* dwarf performed approximately 50% faster on the AMD SDK than on the Intel SDK. In addition, the performance of the Intel SDK was much more erratic than with the AMD SDK. In some cases, like astar and csr, the performance range spanned an order of magnitude more than the range of the results using the AMD SDK. In general, the results show that the more compute-intensive applications performed better with the Intel SDK while the data-transfer-heavy applications perform better with AMD. Overall, what the above tells us is that the compiler and runtime of a system can have a significant effect on realized performance.

3.3 Architecture Diversity

For an ecosystem like OpenCL, which works across multiple architectures, there exists a wide diversity in the capabilities of the underlying architectures. To analyze the behavior of the dwarfs across such diverse platforms, we collected performance results across all the OCD benchmarks running on the aforementioned devices. Figure 2 presents the results as speedup over the Intel SDK CPU results, on a log_{10} scale. The devices in these plots are grouped by type — CPU, low-power GPUs, and high-power GPUs. The GT520 and C2050 are low-power and high-power NVIDIA GPUs, respectively, while the HD5450 and HD5870 are low-power and high-power AMD GPUs, respectively.

The results show that the performance profiles of OCD are quite diverse, not only with respect to the compute time, but also transfer times to and from the device. In fact, we were surprised by the spread of the results for data transfer times. For virtually all of the dwarfs, the data transfer time from the CPU to the GPU was actually shorter than the transfer from the CPU "to" the CPU; the same held true for pulling the data back. The notable exception to this was the Intel SDK version of GEM (i.e., gemnoui), the n-body dwarf from OCD. The code for data transfer in GEM uses the hinting available in OpenCL to specify that the host buffer supplied by the user should be used directly by the OpenCL runtime system, if possible, rather than treating it as a source for a copy. Only the Intel SDK, however, actually honors that

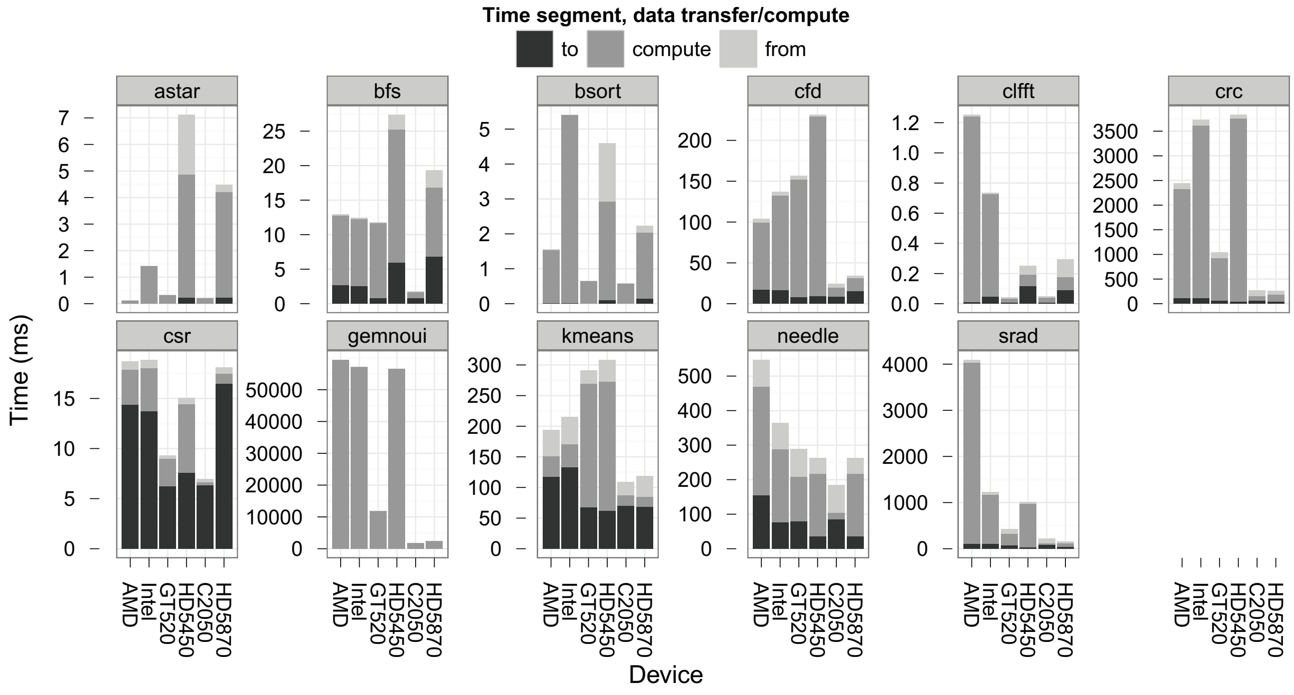

Figure 3: Absolute runtime for all implemented dwarfs across all platforms.

request; the end result is that the transfer time "to" the CPU is extraordinarily small. So, the question remains as to why the data transfer from CPU to GPU across PCIe is generally faster than from one location in CPU memory to another location in CPU memory, a subject of future work.

Relative to both compute time and transfer time, the GPUs from each vendor exhibit similar behavior for certain features, e.g., the performance of data transfer from AMD GPUs for astar, bfs, and bsort. Perhaps more surprising are the anomalies. For instance, the clfft dwarf is the only one for which using low-power GPUs is faster than the high-power GPUs. This particular implementation of FFT is written in such a way that it is highly dependent on the performance of a single processing unit on the GPU rather than the aggregated performance of all the processing units in a GPU. In turn, this seems to favor the smaller GPUs.

In addition to the normalized results, we also present the absolute results as a stack, as shown in Figure 3. While this represents the same data as in Figure 2, it encompasses more devices and allows one to more easily compare the distribution of the actual time spent on data movement and computation for each of the dwarfs. In this case, the times can range from well over 50% data transfer (csr) to over 99% computation (gemnoui). Also interesting is how the performance profile changes from platform to platform. For example, astar spends almost no time to transfer data to or from NVIDIA GPUs but spends significantly more time doing so on the AMD GPUs.

4. CONCLUSION AND FUTURE WORK

In writing and testing the OCD benchmarks, we have found the Berkeley dwarfs to be an effective way to select and classify benchmarks. Using OCD, we have found significant diversity in the applications, architectures, and runtime environments.

Overall, we believe that OpenCL and the 13 Dwarfs will provide a useful baseline for the evaluation of platforms and runtime systems across application domains. In the future, we will continue populating each dwarf with representative applications as well as investigate architecture-aware optimization techniques for the included benchmarks. We are also investigating the possibility of packaging a subset of the OCD to the SPEC High Performance Group (SPEC HPG).

5. AVAILABILITY

The initial release of OCD, currently being beta-tested by selected members of the NSF Center for High-Performance Reconfigurable Computing (CHREC), includes benchmarks representing 11 of the 13 patterns, with the rest to follow in the near future. It has been tested across a multitude of parallel computing architectures including multicore CPUs, graphics processing units (GPUs), accelerated processing units (APUs), i.e., AMD's fused CPU+GPU on a die, and soon, field-programmable gate arrays (FPGAs). It is slated for open-source deployment to the community in April 2012.

6. REFERENCES

[1] R. Anandakrishnan, T. Scogland, A. Fenley, J. Gordon, W. Feng, and A. Onufriev. Accelerating Electrostatic Surface Potential Calculation with Multi-Scale Approximation on Graphics Processing Units. *J. Molecular Graphics and Modelling*, June 2010.

[2] 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, Dec. 2006.