Evaluation framework for GPU performance based on OpenCL standard

Martin Jurečko*, Jana Kočišová*,‡, Ján Buša Jr.*,§, Tomáš Kasanický*,∥, Marek Domiter†,§, Marián Zvada*,‡,*
{jurecko, kocisova, jbusa, kasanicky, domiter, zvada}@sors.com

*FURT Solutions, s.r.o., Strojárenská 3, Košice, Slovakia
†SORŠ Holding, Strojárenská 3, Košice, Slovakia
‡Pavol Jozef Šafárik University in Košice, Šrobárova 3, Košice, Slovakia
§Technical University of Košice, Letná 9, Košice, Slovakia
∥Karlsruhe Institute of Technology, Karlsruhe, Germany
∥Institute of Informatics, Slovak Academy of Sciences, Bratislava, Slovakia

Abstract—There are many projects focused on performance measurements of GPUs but there is no unifying test framework that could be used for evaluating generic floating point intensive applications. This work describes the testing suite for evaluating GPUs that measures raw performance and numerical precision of a subset of OpenCL operations, and analyzes results obtained from several commonly available high-end GPUs.

Keywords—GPU, OpenCL, NVIDIA, ATI, performance, cluster, scalability

I. INTRODUCTION

GPUs (Graphics Processing Units [1]–[3]) are more commonly used not only for graphics applications, but also for non-graphics applications, so-called GPU computing or GPGPU (General-Purpose computation on GPUs). Thanks to their high floating-point operations capability and memory bandwidths [4], GPUs can be successfully used in various science and engineering applications [5]. There are several programming frameworks written directly for GPUs of various vendors [6] (NVIDIA, ATI), but as of today, the OpenCL [7] is the only royalty-free standard for parallel programming of heterogeneous systems. OpenCL specification describes a C-like language for programming kernels to run on an OpenCL-capable device, and an API for transferring data to such devices and running kernels on them.

OpenCL provides an abstraction over vendor-specific details that improves portability over GPUs from various vendors [8]. OpenCL kernels remain close enough to the metal to be nearly as efficient as the kernels written using vendor specific framework.

This work was supported by project FACADE (Financial Analysis Computing Architecture in Distributed Environment) in which we are creating cluster environment based on GPUs [9]. Our main goal is to create a testing suite for easy and automated evaluation of GPUs available on the market. We tested NVIDIA Tesla C1060, NVIDIA GeForce GTX285, and ATI Radeon HD5870 in our experiments.

Different GPU vendors and generation gaps between our units were the main reasons to choose the OpenCL.

The vast majority of papers [10]–[12] do test performance and accuracy but they test it on the basis of typical algorithm s.a. matrix multiplication, matrix transpose, binary search or on some custom, application specific algorithm. The use of such algorithms for evaluating GPUs often leads to specific results that are bound to the domain of an algorithm.

Our approach evaluates GPUs by testing the accuracy and performance of most common OpenCL arithmetic operations, namely addition, subtraction, division, multiplication, sin, cbrt, exp, log, (f)abs, floor, fmod, pow, fmax, and mad.

II. GLOSSARY

Authors prefer simplified NVIDIA CUDA terminology over the terminology introduced by OpenCL. Please refer to NVIDIA CUDA documentation [13] for details.

Coalesced/Uncoalesced access: Read or write access to memory by multiple threads at the same time that can/can not be organized into single read or write operation, either because the memory range requested is/is not continuous or because it does/does not cover range small enough to be transferred through memory bus within single operation.

Core: CUDA multiprocessor, ATI computing unit.

Device: Tested graphics card (GPU).

Golden results: Reference results, used as a basis for comparison of the results computed on GPUs.

Global memory: Memory accessible by all threads, with lifetime spanning multiple kernel launches.

Grid: Working space of threads, organized as a vector, matrix or cube.

Host (Platform): Hardware and the operating system hosting slave device.

Kernel: Code executed by GPU cores in blocks of threads within the grid, possibly concurrently.

Local size: Number of threads within one thread block. Could vary from few dozen to hundreds.
### III. Testing Suite

Test results were produced by our in-house written OpenCL testing suite. Tests aimed to exercise subset of typical arithmetic operations used, or expected to be used in our applications.

**Global size:** Number of elements in the grid and consequently, number of unique threads within grid.

**Shared memory:** Low latency cache memory, private to all threads within one thread block. Used as a grounds for threads communication within thread block, or as a programmer controlled L1 data cache for global memory.

**Shared memory bank:** Portion of the shared memory with limited parallel access – concurrent conflicting access by two or more threads must be serialized.

**Thread block:** Group of threads executed by a GPU core, seemingly in parallel.

**Unit in last place (ulp):** The measure of numerical precision. According to [14], ulp is defined to be the absolute value of the difference between the two numbers in a given finite numerical representation which are closest to a given number. For more elaborate definition of ulp see [7, Sec 7.4].

### A. Tested OpenCL Operations

Current tests groups cover following arithmetic operations:

- unary — sin, cbrt, exp, log, (f)abs, floor;
- binary — addition, subtraction, multipication, division, fmod, pow, fmax;
- ternary — multiply and add.

Some unary tests on HD5870 were skipped due to differences between drivers implementation and official OpenCL specification. We received internal error: no implementation for longX, abs(ulongX) and there was no implicit conversion for longX variable = ulongX(value).

Other tests were skipped due to GPU hardware limitations. HD5870 didn’t support local sizes of 512 threads. All GPUs had shared shared memory constraints manifesting by kernel compile failures, mostly in tests with highest possible local sizes.

### B. Tests Inputs

Kernels operated on randomly generated input data with maximum possible length, limited only by the amount of allocable GPU’s global memory. At the time of testing, the allocation limit of OpenCL implementations was 1/4 of the total available amount of GPU’s global memory.
To eliminate data corruption caused by faulty hardware or by possibly incorrect algorithm design, kernels ran multiple times on the same inputs and their outputs were compared to each other.

C. Measuring Numerical Precision

Test outcome was either pass or fail, depending on the number and on the extent of differences between all elements in kernel output and in golden result, computed on host CPU.

Elements were considered mismatching if their difference was greater than some kernel specific threshold. Threshold values depend on the exercised operation, used element types, the number of repetitions of operation within the kernel, and also on host CPU compliance to IEEE 754.

Different CPUs (Intel, AMD) manifested differences in golden results, usually in magnitude of ulp tolerances of corresponding OpenCL operations. To account for precision deviations between various CPU FPUs, thresholds values were chosen empirically using OpenCL specification [7, Sec. 7.4] as a starting point.

Results within run that had more than average count of mismatches, and results with high average or high maximal differences between the elements were investigated, marked as failed, and they were excluded from performance analysis.

D. Measuring Performance

Tests benchmarked kernel operations only, excluding timing or checking of any memory transfers between the host and the device. All kernels were fully convergent i.e. every thread executed the same code path. Test performance was measured as a throughput of its kernel in GB/s, equal to the number of processed input and output elements times element’s size in bytes, divided by the average of kernel run times.

IV. Results Analysis

A. Aggregation of Tests Results

Cards were tested multiple times on multiple platforms yielding moderate collection of results. To simplify the analysis and to perform basic measurements validity checking, results were divided into run groups containing results of all runs of the same kernel on the same GPU. Performance and precision measurements of results from the same run group should not vary much, not even between different platforms hosting the same GPU.

In the processes of statistical analysis we focused on statistical dispersion in the sense of variability results of tests. The sum of squared deviations can be related to moments calculated from the data

\[
s^2(X) = E(X^2) - (E(X))^2, \tag{1}
\]

where \( E \) denotes a mean value of the data set \( X \) [15].

Large dispersions in measured throughputs could be caused by small kernel run times or by small work size (corresponding to input elements count); dispersions in mismatches (see Figure 1) indicate faulty algorithm or its sensitivity to input data, or it could be caused by overly small differences threshold. All results with large dispersions were excluded from later evaluation. Run groups with majority of excluded results were considered unreliable and were also excluded.

Run groups with small dispersions were aggregated into one representative result, simplifying further analysis.

Boundary values were chosen empirically after analyzing failed tests to be 1% for mismatches and 0.5% for throughput dispersions. Higher value of mismatches dispersion reflects the fact that the runs were obtained from various platforms differing in implementations of their FPUs used to compute golden results.

Figure 2 shows dispersions in throughput of aggregated results. It could be seen that the throughput dispersion is generally low.

After the aforementioned filtering, the results from all run groups were aggregated by picking one representative result per run group. Unlike averaging, aggregation preserves the
relation between test throughput and mismatch count. We aggregated the results by selecting results with the smallest measured mismatch count; the only other option would be to prefer results with highest throughput.

B. Mismatches in Aggregated Results

Aggregation method described in Section IV-A selects the results with highest numerical precision and detects major result affecting errors, but it does not guarantee that the precision of selected results will be sufficient to be acceptable for practical use. Run group consisting solely of results with high but close mismatch counts would not be detected in aggregation, yet the validity of such results is dubious at least. Such cases were investigated and excluded from further analysis.

Maximal acceptable mismatch value is largely application dependent. We chose boundary mismatch values per card, so that at least 80% of results would end up having their mismatch count smaller or equal to the boundary value.

Figure 3 shows cumulative distribution function from tests counts to mismatches percent, for every tested card.

Tests with mismatch counts higher than boundary mismatch values were discarded. Table I shows count of passed and failed (and therefore discarded) tests.

C. Aggregating Results by Local Size

Optimal local size for certain kernel code with regard to performance depends mostly on architecture of respective GPU, and is a matter of choice limited by occupancy constraints imposed by kernel requirements. Programmers usually have the freedom to pick suitable local size, favoring the one giving largest throughput and hardly, if ever, preferring less efficient local sizes. Due to similar reasons, the results of tests differing only in local size were aggregated to single result with best throughput.

Precision measurements on kernels differing only in local size should not have large mismatch count dispersions. Irregularities are usually caused by faulty algorithm, or they could be rarely caused by a bug in OpenCL compiler. Figure 5 shows mismatches dispersions of aggregated results and it can be seen that the dispersions values are very low compared to dispersions obtained by aggregating results from multiple runs on the same GPU (see Figure 1).

Figure 4 shows counts of aggregated results with best throughput by local sizes. It seems that HD5870, and possibly whole Cypress platform prefer large local sizes available, while NVIDIA CUDA based cards prefer small local sizes.

D. Measuring Repeated Arithmetic Operations

Operations were tested by multiple kernels differing only in number of the repetitions of respective operation (see Section III-A for explanation). Figure 6 gives an overview of average throughputs of results aggregated by differing repeat counts. Nonlinear scaling behavior and performance drop-downs at 8 for NVIDIA GPUs and at 4 repeats in HD5870 results could probably be attributed to occupancy effects. To simplify further analysis, we filtered out all results but those with repeat count equal to 1.

E. Comparison of Performance by Element Types

Generated kernels combinations also exercise their operations on various input and output element types, namely:

- scalar types – char, short, int, long, float, and double;
- vector types – char\text{n}, short\text{n}, int\text{n}, long\text{n}, float\text{n}, and double\text{n} for \( n = 2, 4, \text{and } 8 \).

Table II, III, and IV show average throughputs of operations on selected scalar and vector types.

V. TESTED GPU HARDWARE AND HOST PLATFORMS

We used two host platforms: Zodiac – 2\times Intel Xeon E5530, 47GB RAM and Hydra – AMD Phenom II X2 545, 4GB RAM.
Figure 6. Average throughput as a function of repeat count of arithmetic operations.

Three cards were tested: two from NVIDIA – Tesla C1060 (Zodiac), GeForce GTX285 (Zodiac, Hydra) both on platform OpenCL 1.0 CUDA 3.0.1 with NVIDIA driver version 195.36.15 and one from ATI – Radeon HD5870 (Hydra) on platform OpenCL 1.0 ATI Stream 2.0.1 with ATI driver version CAL 1.4.519.

VI. FAULTS AND OMISSIONS

The HD5870 we tested was available to us for relatively short time and only in the early stages of the development of the testing suite. Furthermore, used OpenCL driver did not support all tested operations and data types and even manifested deviations from OpenCL standard (e.g., function abs() on signed vector types returned their unsigned counterpart).

Also, the tests were progressively expanded – some earlier measurements do not cover every test group, or some of their combinations, leading to fewer measurements of tests that were added later. Again, this mostly affects HD5870. Consequently, compared to statistical analyses of other cards, the analysis for HD5870 is less complete and its results contain fewer tests runs.
VII. Future Work

Test more arithmetic operations and combinations of operations. We tested basic arithmetic operations that should be supported natively by the underlying platform and therefore should reflect raw HW performance. Testing of derived operations could reveal the optimality of their library implementations, e.g., we tested sin but we could test asin, asinpi. Test wider span of repeat counts. It would be interesting to see results of combinations covering high repeat counts, ideally up until the cases manifesting $\text{throughput}(n) > \text{throughput}(1) \times n$. This could give us some insight into efficient loop unrolling vs. kernel partitioning strategies.

Cover remaining element types as half, wider vectors, etc. Ensure the repeatability of tests by freezing inputs – use a generator, keep the seed.

VIII. Conclusions

Our framework covered nearly every combination of operations, element types and local sizes we selected for study. Combinations were compiled into standalone kernels and each of them was tested multiple times, on different host platforms. Additionally, tests for arithmetic operations were expanded by varying number of repetitions, aimed to create kernels with big enough execution times to hide kernel setup time for new threads. Tested element types include most of scalar and vector integers and floating points with the exception of half types. Double precision floating points were tested only on NVIDIA cards as at the time of testing, ATI drivers did not support required OpenCL extensions.

Described testing suite is capable of validating most used portions of OpenCL host API and it also provides a means to validate and evaluate representative selection of typically used GPU arithmetic operations. The tests results we obtained and analyzed in this work showed to be essential for planning our cluster hardware strategy.

References


