Performance Evaluation of OpenCL Standard Support (and Beyond)

Tyler Sorensen
Princeton University, USA
ts20@princeton.edu

Sreepathi Pai
University of Rochester, USA
sree@cs.rochester.edu

Alastair F. Donaldson
Imperial College London, UK
alastair.donaldson@imperial.ac.uk

ABSTRACT

In this talk, we will discuss how support (or lack of it) for various OpenCL (OCL) features affects performance of graph applications executing on GPU platforms. Given that adoption of OCL features varies widely across vendors, our results can help quantify the performance benefits and potentially motivate the timely adoption of these OCL features.

Our findings are drawn from the experience of developing an OCL backend for a state-of-the-art graph application DSL, IrGL, originally developed with a CUDA backend [1]. IrGL allows competitive algorithms for applications such as breadth-first-search, page-rank, and single-source-shortest-path to be written at a high level. A series of optimisations can then be applied by the compiler to generate OCL code. These user-selectable optimisations exercise various features of OCL: on one end of the spectrum, applications compiled without optimisations require only core OCL version 1.1 features; on the other end, a certain optimisation requires inter-workgroup forward progress guarantees, which are yet to be officially supported by OCL, but have been empirically validated and are relied upon e.g. to achieve global device-wide synchronisation [3]. Other optimisations require OCL features such as: fine-grained memory consistency guarantees (added in OCL 2.0) and subgroup primitives (added to core in OCL 2.1).

Our compiler can apply 6 independent optimisations (Table 1), each of which requires an associated minimum version of OCL to be supported. Increased OCL support enables more and more optimisations: 2 optimisations are supported with OCL 1.x; 1 additional optimization with OCL 2.0; and a further 2 with OCL 2.1. Using OCL FP to denote v2.1 extended with *forward progress* guarantees (not officially supported at present), the last optimisation is enabled. We will discuss the OCL features required for each optimisation and the idioms in which the features are used. Use-case discussions of these features (e.g. memory consistency and subgroup primitives) are valuable as there appear to be very few open-source examples: a GitHub search yields only a small number of results.

Our compiler enables us to carry out a large and controlled study, in which the performance benefit of various levels of OCL support can be evaluated. We gather runtime data exhaustively on all combinations across: all optimisations, 17 applications, 3 graph inputs, 6 different GPUs, spanning 4 vendors: Nvidia, AMD, Intel and ARM (Table 2).

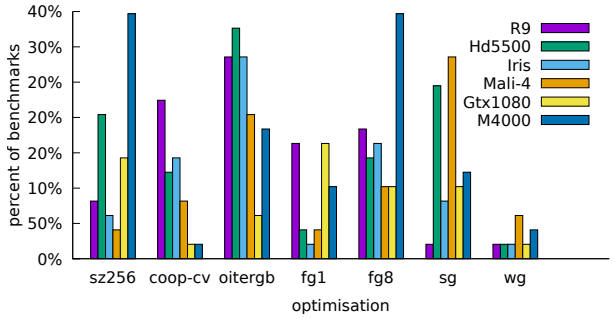

Figure 1: Summary of optimisations used per-chip to obtain the top speedups in our empirical study. An optimisation is counted when it was required for an application/input tuple to reach the highest speedup.

We show two notable results in this abstract: our first result, summarised in Figure 1, shows that all optimizations can be beneficial across a range of GPUs, despite significant architectural differences (e.g. subgroup size as seen in Table 2). This provides motivation that previous vendor specific approaches (e.g. for Nvidia) can be ported to OCL and achieve speedups on range of devices.

Our second result, summarised in Figure 2, shows that if feature support is limited to OCL 2.0 (or below), the available optimisations (fg wg sz256) fail to achieve any speedups in over 70% of the chip/application/input benchmarks. If support for OCL 2.1 (adding the optimizations: sg coop-cv) is considered, this number drops to 60% but observed speedups are modest, rarely exceeding 2×. Finally, if forward progress guarantees are assumed (adding the oitergb optimization), speedups are observed in over half of the cases, including impressive speedups of over 14× for AMD and Intel GPUs. This provides compelling evidence for forward progress properties to be considered for adoption for a future OCL version.

An extended version of this material can be found in [2, ch. 5].

ACM Reference Format:
Tyler Sorensen, Sreepathi Pai, and Alastair F. Donaldson. 2019. Performance Evaluation of OpenCL Standard Support (and Beyond). In *International Workshop on OpenCL (IWOCL'19), May 13–15, 2019, Boston, MA, USA.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3318170.3318177

IWOCL'19, May 13–15, 2019, Boston, MA, USA

ACM ISBN 978-1-4503-6230-6/19/05.
https://doi.org/10.1145/3318170.3318177

REFERENCES

[1] Sreepathi Pai and Keshav Pingali. 2016. A compiler for throughput optimization of graph algorithms on GPUs. In *OOPSLA*. 1–19.
[2] Tyler Sorensen. 2018. *Inter-workgroup Barrier Synchronisation on Graphics Processing Units.* Ph.D. Dissertation. Imperial College London. http://www.cs.princeton.edu/~ts20/files/phdthesis.pdf.

Table 1: List of optimisations, the OpenCL features they exploit and the architectural parameters that influence performance. Support class refers to the OpenCL version support described in the text.

Optimisation	OCL features	Support class	Architecture parameters
cooperative conversion (coop-cv)	Local memory, `sub_group_any`, `sub_group_reduce`, `barrier`, `atomic_fetch_and_add`, popcount	OCL 2.1	workgroup size, subgroup size, atomic read–modify–write throughput, subgroup collectives throughput
fine-grained nested parallelism (fg)	Local memory, `barrier`	OCL 1.x	local memory size, barrier throughput
subgroup nested parallelism (sg)	Local memory, `sub_group_barrier`, `sub_group_any`, `atomic_store`, `atomic_load`	OCL 2.1	subgroup size, subgroup barrier throughput, local memory size
workgroup nested parallelism (wg)	Local memory, `atomic_store`, `atomic_load`, `barrier`	OCL 2.0	workgroup size, local memory size, barrier throughput, atomic load/store throughput
iteration outlining (oitergb)	`atomic_load`, `atomic_store`	OCL FP	overheads for kernel launch and memory transfers, global memory fence throughput, workgroup scheduler behaviour
workgroup size of 256 (sz256)	`clEnqueueNDRangeKernel`	OCL 1.x	occupancy, resource limits

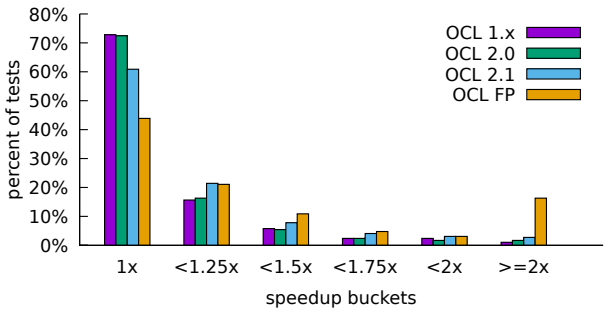

Figure 2: Summary of top speedups over all tests (chip/application/input tuples), grouped by OCL support class. As higher levels of support are provided, a larger number of tests achieve higher speedups, with the experimental support class OCL FP providing a significant proportion of speedups over 2×.

Table 2: The GPUs considered in this work: a short name used in figures, the number of compute units (#CUs), the subgroup size (SG size), and the supported OpenCL version (OCL).

Vendor	Chip	Short name	#CUs	SG size	OCL
Nvidia	Quadro M4000	M4000	13	32	1.2
	GTX 1080	Gtx1080	20	32	1.2
Intel	HD5500	Hd5500	27	8,16	2.0
	Iris 6100	Iris	47	8,16	2.0
AMD	Radeon R9 Fury	R9	28	64	2.0
ARM	Mali-T628	Mali-4	4	1	1.2

[3] Tyler Sorensen, Alastair F. Donaldson, Mark Batty, Ganesh Gopalakrishnan, and Zvonimir Rakamaric. 2016. Portable inter-workgroup barrier synchronisation for GPUs. In *OOPSLA*. 39–58.