SYCL-Bench
A Versatile Single-Source Benchmark Suite for Heterogeneous Computing

IWOCL/SYCLcon 2020

Sohan Lal
Nicolai Stawinoga

Aksel Alpay (Speaker)
Vincent Heuveline

Philip Salzmann
Peter Thoman
Thomas Fahringer

Biagio Cosenza
SYCL implementation ecosystem

▶ Growing SYCL implementation ecosystem

Figure: SYCL implementations. (Figure is part of the hipSYCL project: https://github.com/illuhad/hipSYCL)

▶ How do SYCL implementations compare in terms of performance for given hardware and code?
Motivation

- SYCL relies heavily on implicit behavior
- SYCL implementations use different optimizations
  ▶ Patterns efficient in one implementation may be inefficient in another
- Performance implications are not consistently documented across implementations
  ▶ Will my SYCL implementation overlap compute/data transfers?
  ▶ Will my SYCL implementation avoid unnecessary data transfers?
  ▶ What is the runtime overhead of different SYCL implementations?
- SYCL allows testing a wide variety of hardware → interesting for hardware characterization
  ⇒ A benchmark suite dedicated to characterizing SYCL implementations and hardware is needed
SYCL-Bench

- SYCL-Bench main goals
  - Hardware characterization
  - SYCL implementation characterization
  - SYCL-specific benchmarks to evaluate SYCL-runtime

- SYCL-Bench contains three categories
  - Microbenchmarks
  - Applications/Single kernels
  - SYCL runtime benchmarks

- First benchmark suite focused entirely on SYCL
- Composed of original benchmarks and SYCL ports from Rodinia\(^1\) and PolyBench\(^2\)
- Open source: https://github.com/bcosenza/sycl-bench

---


http://www.cse.ohio-state.edu/~pouchet/software/polybench
SYCL-Bench Features

- Benchmarking framework (e.g. common command line arguments) for easy extension
- Verification layer for (almost) all benchmarks
- Focus on reproducibility using test profiles and run-suite script
- Automated execution of the entire benchmark suite
- All results can be automatically saved to a single csv output file
- Mechanisms to make sure that no undesired data transfers are measured
- Measure total execution time and kernel time if SYCL implementation supports queue profiling
- Where appropriate, provides benchmarks in many variants with different template types or SYCL kernel submission mechanisms
SYCL implementation support

- Goal: Support all SYCL implementations
- Tested with ComputeCpp (PTX, SPIR backends), hipSYCL (CPU, CUDA, ROCm backends)
- Experimental/partial support for LLVM SYCL and LLVM SYCL CUDA backend
- triSYCL WIP
Common arguments

- `--size=<problem-size>` – to scale problem size (usually, translates to global range of work items)
- `--local=<local-size>` – work group size (not utilized by all benchmarks)
- `--num-runs=<N>` – number of runs for runtime average/median calculation
- `--device=<d>` – select cpu or gpu
- `--no-verification` – disable verification
- `--no-ndrange-kernels` – skip kernels using ndrange parallel for
## Benchmarks

<table>
<thead>
<tr>
<th>Benchmark Name</th>
<th>Short</th>
<th>Domain</th>
</tr>
</thead>
<tbody>
<tr>
<td>DRAM, arith, sf, local_mem, host_device_bandwidth</td>
<td>-</td>
<td>Microbenchmarking</td>
</tr>
<tr>
<td>mol_dyn</td>
<td>MD</td>
<td>Physics Simulation</td>
</tr>
<tr>
<td>nbody</td>
<td>NBODY</td>
<td>Physics Simulation</td>
</tr>
<tr>
<td>scalar_prod</td>
<td>SP</td>
<td>Linear Algebra</td>
</tr>
<tr>
<td>vec_add</td>
<td>VA</td>
<td>Linear Algebra</td>
</tr>
<tr>
<td>2mm(from PolyBench)</td>
<td>2MM</td>
<td>Linear Algebra</td>
</tr>
<tr>
<td>3mm(from PolyBench)</td>
<td>3MM</td>
<td>Linear Algebra</td>
</tr>
<tr>
<td>atax(from PolyBench)</td>
<td>ATAX</td>
<td>Linear Algebra</td>
</tr>
<tr>
<td>bicg(from PolyBench)</td>
<td>BICG</td>
<td>Linear Algebra</td>
</tr>
<tr>
<td>gemm(from PolyBench)</td>
<td>GEMM</td>
<td>Linear Algebra</td>
</tr>
<tr>
<td>gesummv(from PolyBench)</td>
<td>GESUM</td>
<td>Linear Algebra</td>
</tr>
<tr>
<td>gramschmidt(from PolyBench)</td>
<td>GRAMS</td>
<td>Linear Algebra</td>
</tr>
<tr>
<td>mvt(from PolyBench)</td>
<td>MVT</td>
<td>Linear Algebra</td>
</tr>
<tr>
<td>syr2k(from PolyBench)</td>
<td>SYR2K</td>
<td>Linear Algebra</td>
</tr>
<tr>
<td>syrk(from PolyBench)</td>
<td>SYRK</td>
<td>Linear Algebra</td>
</tr>
</tbody>
</table>
### Benchmarks

<table>
<thead>
<tr>
<th>Benchmark Name</th>
<th>Short</th>
<th>Domain</th>
</tr>
</thead>
<tbody>
<tr>
<td>ftd2d (from PolyBench)</td>
<td>FTD2D</td>
<td>Stencils</td>
</tr>
<tr>
<td>2DConvolution (from PolyBench)</td>
<td>2DCON</td>
<td>Image Processing</td>
</tr>
<tr>
<td>3DConvolution (from PolyBench)</td>
<td>3DCON</td>
<td>Image Processing</td>
</tr>
<tr>
<td>sobel3/5/7</td>
<td>SOBEL3/5/7</td>
<td>Image Processing</td>
</tr>
<tr>
<td>median</td>
<td>MEDIAN</td>
<td>Image Processing</td>
</tr>
<tr>
<td>correlation (from PolyBench)</td>
<td>CORR</td>
<td>Data Mining</td>
</tr>
<tr>
<td>covariance (from PolyBench)</td>
<td>COV</td>
<td>Data Mining</td>
</tr>
<tr>
<td>lin_reg_coeff</td>
<td>LRC</td>
<td>Data Analytics</td>
</tr>
<tr>
<td>lin_reg_error</td>
<td>LRE</td>
<td>Data Analytics</td>
</tr>
<tr>
<td>blocked_transform</td>
<td>BT</td>
<td>SYCL runtime benchmarking</td>
</tr>
<tr>
<td>dag_task_throughput_independent</td>
<td>DTI</td>
<td>SYCL runtime benchmarking</td>
</tr>
<tr>
<td>dag_task_throughput_sequential</td>
<td>DTS</td>
<td>SYCL runtime benchmarking</td>
</tr>
<tr>
<td>reduction</td>
<td>RD</td>
<td>Parallel pattern</td>
</tr>
<tr>
<td>segmentedreduction</td>
<td>SRD</td>
<td>Parallel pattern</td>
</tr>
</tbody>
</table>

► ...actual number of obtained results much larger! (templated kernels and different SYCL kernel invocation mechanisms)
Experimental Evaluation

Hardware:
- Intel Xeon CPU E5-2699 v3
- NVIDIA Titan X

Software:
- Ubuntu 16.04
- hipSYCL master(12406c8c) + clang 9 + LLVM 9 OpenMP + CUDA 10.1
- ComputeCpp 1.3 + Intel OpenCL 18.1.0.013 + CUDA 10.1 OpenCL
- SYCL-bench test profile from sohan-dev branch
Microbenchmarks

SYCL-Bench includes **five microbenchmarks** for device characterization

**DRAM**
- 1D: Measure memory bandwidth by copying SP/DP values between two buffers
- 2D/3D: Additionally quantify the quality of mapping of work items to hardware threads

**host_device_bandwidth**
- Measure host ⇔ device copy bandwidth for 1D/2D/3D contiguous and strided buffers

**local_mem**
- Measure bandwidth by continuously swapping SP/DP values in local memory

**arith / sf**
- Measure arithmetic / special function throughput
- FMA / sin(tan(cos(x))) in a loop
Microbenchmarks on NVIDIA Titan X

(a) DRAM Bandwidth

(b) Host/Device Bandwidth

(c) Local Memory

(d) Arithmetic Throughput

(e) SFU Throughput
Cases where hipSYCL is much slower than ComputeCpp (LRC, SPND) are caused by `ndrange` parallel for invocations

⇒ `ndrange` parallel for is not performance portable!
hipSYCL and ComputeCpp on NVIDIA Titan X

- **hipSYCL results are overall times: kernel + overheads**
- **ComputeCpp PTX backend mainly limited by runtime overheads**
Let's look at task throughput of SYCL implementations!

**sequential task throughput**
- Submits many kernels which all require r/w access to the same buffer → SYCL implementation needs to order them sequentially
- Kernels are trivial: single work group, atomic add to counter for validation.
- Problem size corresponds to number of kernels submitted

**independent task throughput**
- Submits many kernels which all require r/w access to different buffers → SYCL implementation can execute multiple kernels simultaneously
- Kernels are trivial: Single work group, work item 0 sets buffer content for validation
- Problem size corresponds to number of kernels submitted
For sequential tasks, both implementations show very similar throughput
Performance likely “as good as it gets”
Throughput for independent tasks on NVIDIA Titan X

- hipSYCL showed higher independent task throughput on GPU vs ComputeCpp
- This problem stresses the scheduler of the SYCL runtime
- → probably lower scheduling overhead in hipSYCL
Throughput for independent tasks on Xeon E5-2699 v3

- On CPU, ComputeCpp shows consistently higher task throughput
- Different CPU kernel execution mechanisms: hipSYCL (OpenMP) vs Intel OpenCL (ComputeCpp)
- (Note: hipSYCL single task does not go through OpenMP!)
SYCL-Bench: benchmark suite dedicated to SYCL benchmarking

https://github.com/bcosenza/sycl-bench

Focus on characterization of hardware and SYCL implementations

Takes into account SYCL specifics (e.g. multiple kernel submission mechanisms)

ndrange parallel for is not performance portable

ComputeCpp PTX backend is mainly limited by runtime overheads

Future: Add more benchmarks (e.g. more parallel patterns), expand supported SYCL implementations

Acknowledgments: DFG project CELERITY CO 1544/1-1 and EPSRC fellowship EP/N018869/1