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

IWOCL/SYCLcon 2020

Technische Universität Berlin

Sohan Lal
Nicolai Stawinoga

Universität Heidelberg

Aksel Alpay (Speaker)
Vincent Heuveline

universität innsbruck

Philip Salzmann
Peter Thoman
Thomas Fahringer

Università degli Studi di Salerno

Biagio Cosenza

SYCL implementation ecosystem

- Growing SYCL implementation ecosystem

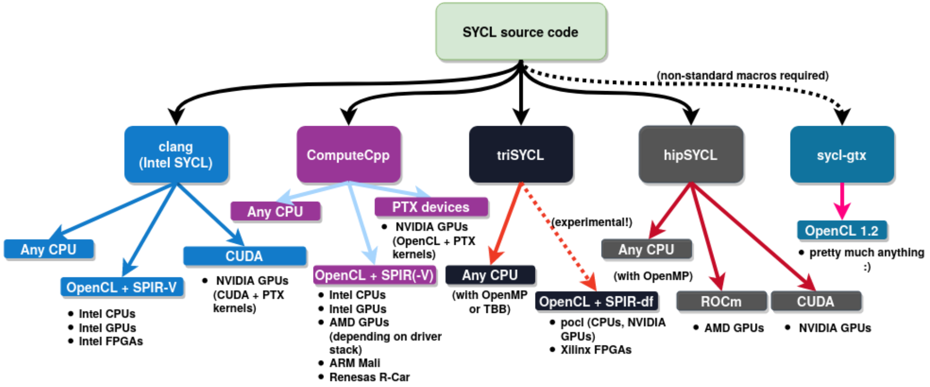


Figure: SYCL implementations. (Figure is part of the hipSYCL project: `https://github.com/illuhad/hipSYCL`)

- 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 $\rightarrow$ interesting for hardware characterization

$\Rightarrow$ 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`

[1] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, Sang-Ha Lee and K. Skadron. "Rodinia: A Benchmark Suite for Heterogeneous Computing". IEEE International Symposium on Workload Characterization, Oct 2009.

[2] L.-N. Pouchet, U. Bondhugula, et al. The polybench benchmarks. 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

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

Benchmarks

Benchmark Name	Short	Domain
fdtd2d(from PolyBench)	FTD2D	Stencils
2DConvolution(from PolyBench)	2DCON	Image Processing
3DConvolution(from PolyBench)	3DCON	Image Processing
sobel3/5/7	SOBEL3/5/7	Image Processing
median	MEDIAN	Image Processing
correlation(from PolyBench)	CORR	Data Mining
covariance (from PolyBench)	COV	Data Mining
lin_reg_coeff	LRC	Data Analytics
lin_reg_error	LRE	Data Analytics
blocked_transform	BT	SYCL runtime benchmarking
dag_task_throughput_independent	DTI	SYCL runtime benchmarking
dag_task_throughput_sequential	DTS	SYCL runtime benchmarking
reduction	RD	Parallel pattern
segmentedreduction	SRD	Parallel pattern

- ...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

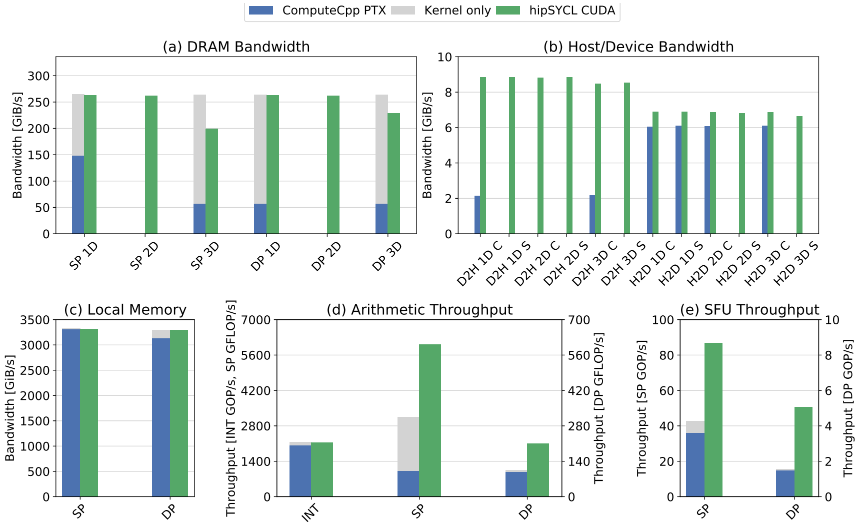
- Measure bandwidth by continuously swapping SP/DP values in local memory

arith / sf

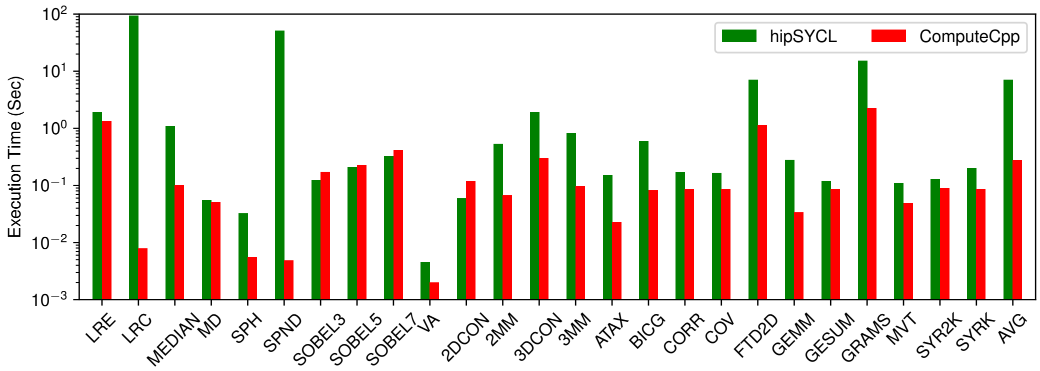
- Measure arithmetic / special function throughput
- FMA / sin(tan(cos(x))) in a loop

Microbenchmarks on NVIDIA Titan X

UNIVERSITÄT
HEIDELBERG
ZUKUNFT
SEIT 1386

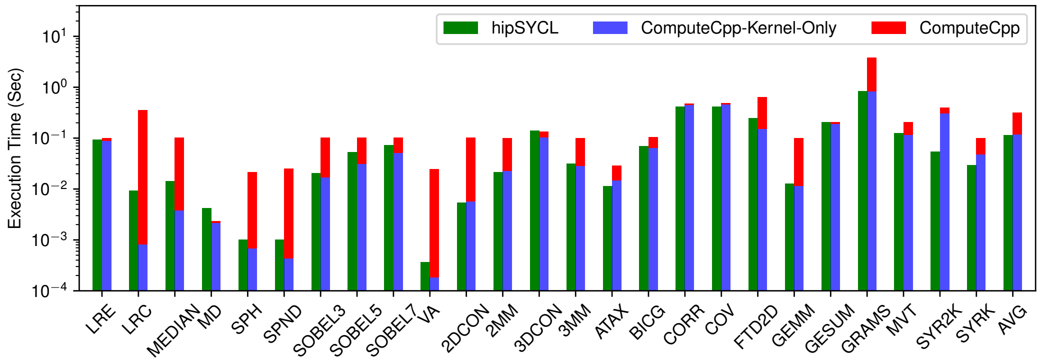


hipSYCL and ComputeCpp on Xeon E5-2699 v3



- Cases where hipSYCL is much slower than ComputeCpp (LRC, SPND) are caused by ndrange parallel for invocations
- $\Rightarrow$ 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

SYCL runtime benchmarking

Let's look at task throughput of SYCL implementations!

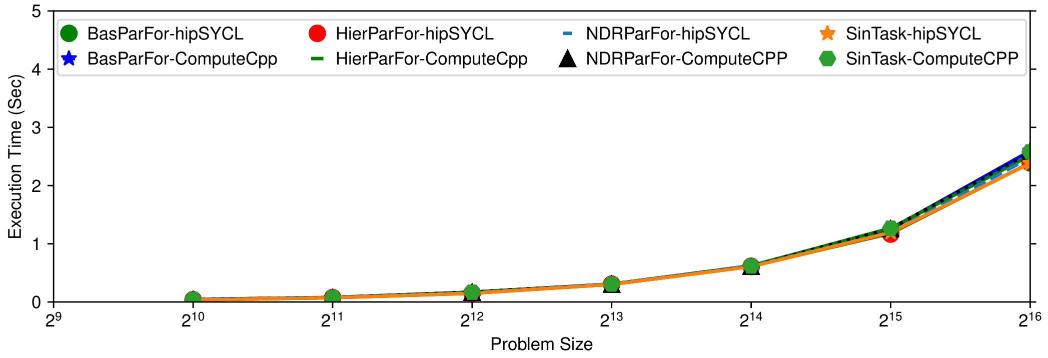
sequential task throughput

- Submits many kernels which all require r/w access to the same buffer $\rightarrow$ 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

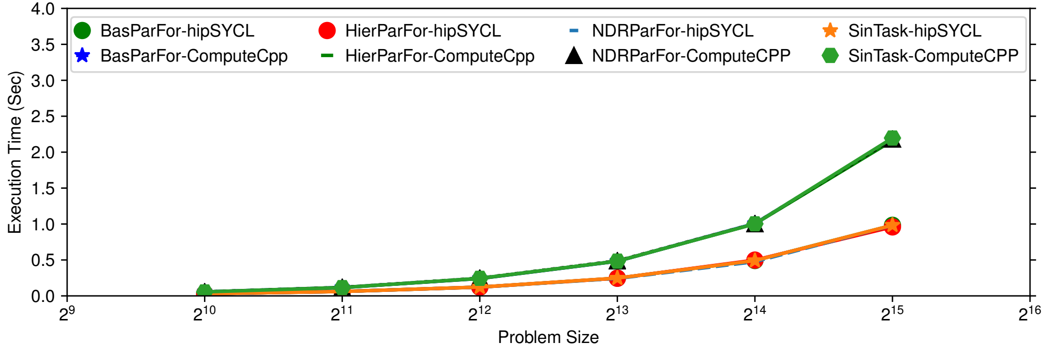
- Submits many kernels which all require r/w access to different buffers $\rightarrow$ SYCL implementation can execute multiple kernels simultaneously
- kernels are trivial: Single work group, work item 0 sets buffer content for validation
- Problem size corresponds to number of kernels submitted

Throughput for sequential tasks on NVIDIA Titan X



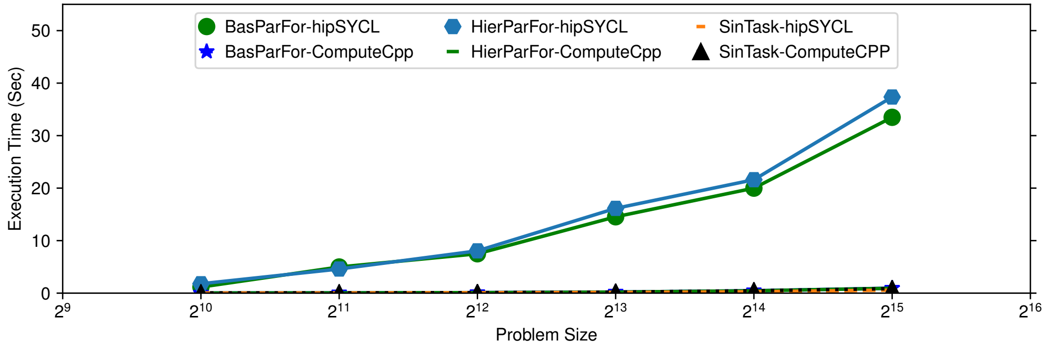
- For sequential tasks, both implementations show very similar throughput
- Performance likely “as good as it gets”

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
- $\rightarrow$ 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!)

The Last Slide

- 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