A Comparison of OpenCL, CUDA, and HIP as Compilation Targets for a Functional Array Language

Troels Henriksen

Troels Henriksen
University of Copenhagen
athas@sigkill.dk
gopher://sigkill.dk

FOSDEM 2025

What I do

I work on **Futhark**, a data parallel functional programming language.

```
def add_two [n]     (a:  [n]i32):     [n]i32 = map (+2) a
def     sum [n]     (a:  [n]i32):        i32 = reduce (+) 0 a
def sumrows [n][m] (as: [n][m]i32): [n]i32 = map sum as
```

Main claim to fame is **compilation to GPU code**, meaning Futhark programs are (hopefully) pretty fast.

Three production-quality GPU backends

OpenCL

- Open standard, actively maintained.
- Implemented to various degrees by NVIDIA, AMD, Intel, etc...

Three production-quality GPU backends

OpenCL

- Open standard, actively maintained.
- Implemented to various degrees by NVIDIA, AMD, Intel, etc...

CUDA

- Proprietary NVIDIA standard.
- Dominant position within GPGPU.

Three production-quality GPU backends

OpenCL

- Open standard, actively maintained.
- Implemented to various degrees by NVIDIA, AMD, Intel, etc...

CUDA

- Proprietary NVIDIA standard.
- Dominant position within GPGPU.

HIP

- Reimplementation of CUDA API by AMD.

Three production-quality GPU backends

OpenCL

- Open standard, actively maintained.
- Implemented to various degrees by NVIDIA, AMD, Intel, etc...

CUDA

- Proprietary NVIDIA standard.
- Dominant position within GPGPU.

HIP

- Reimplementation of CUDA API by AMD.

Very similar—so what are the performance differences for equivalent code, and why?

Experimental performance investigation

Basic idea: compile Futhark benchmark programs with various backends and run on various GPUs and see how fast they are.

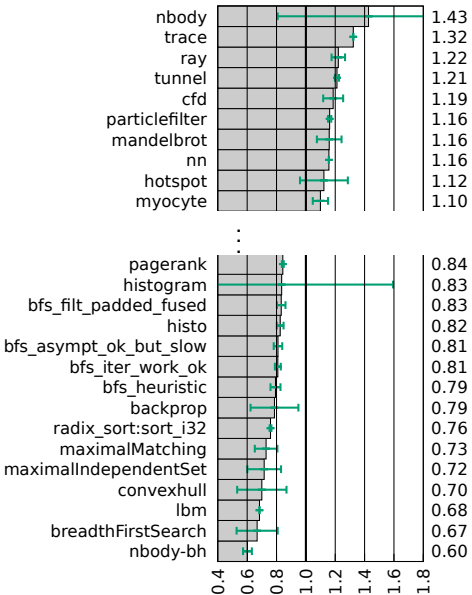
On NVIDIA A100

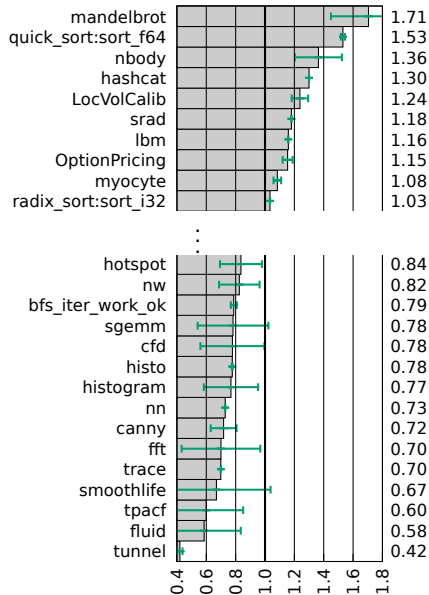
Compare OpenCL and CUDA backends.

On AMD MI100

Compare OpenCL and HIP backends.

48 benchmarks from published benchmark suites (Accelerate, PBBS, Rodinia, Parboil).


A100 - OpenCL vs CUDA
nbody 1.43
trace 1.32
ray 1.22
tunnel 1.21
cfd 1.19
particlefilter 1.16
mandelbrot 1.16
nn 1.16
hotspot 1.12
myocyte 1.10
pagerank 0.84
histogram 0.83
bfs_filt_padded_fused 0.83
histo 0.82
bfs_asympt_ok_but_slow 0.81
bfs_iter_work_ok 0.81
bfs_heuristic 0.79
backprop 0.79
radix_sort:sort_i32 0.76
maximalMatching 0.73
maximalIndependentSet 0.72
convexhull 0.70
lbm 0.68
breadthFirstSearch 0.67
nbody-bh 0.60
0.4 0.6 0.8 1.0 1.2 1.4 1.6 1.8

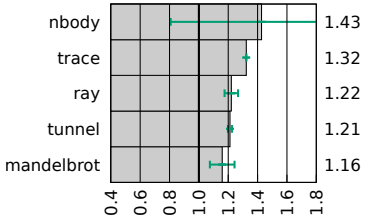

MI100 – OpenCL vs HIP
mandelbrot 1.71
quick_sort:sort_f64 1.53
nbody 1.36
hashcat 1.30
LocVolCalib 1.24
srad 1.18
lbm 1.16
OptionPricing 1.15
myocyte 1.08
radix_sort:sort_i32 1.03
hotspot 0.84
nw 0.82
bfs_iter_work_ok 0.79
sgemm 0.78
cfd 0.78
histo 0.78
histogram 0.77
nn 0.73
canny 0.72
fft 0.70
trace 0.70
smoothlife 0.67
tpacf 0.60
fluid 0.58
tunnel 0.42
0.4 0.6 0.8 1.0 1.2 1.4 1.6 1.8

Cause: Numerical defaults for single-precision floating point

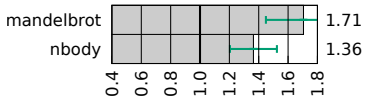
OpenCL allows fast-but-wrong division and square roots by default.

- Disabled by `-cl-fp32-correctly-rounded-divide-sqrt`.

A100 - OpenCL/CUDA



MI100 – OpenCL/HIP

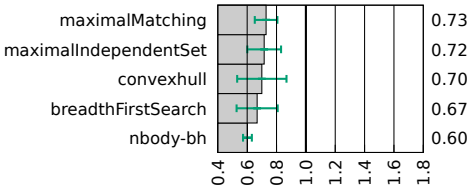


Cause: Different scan implementations

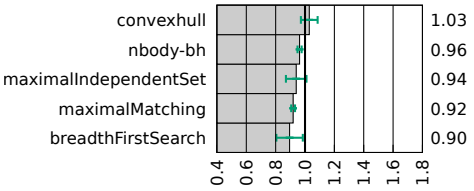
OpenCL does not allow the implementation of the decoupled lookback scan algorithm.

- Depends on subtle memory model and progress guarantees.[1]
  - Not provided by OpenCL spec.
- *Might* be possible to implement, but I have not seen anyone make it work.

A100 - OpenCL/CUDA



MI100 – OpenCL/HIP



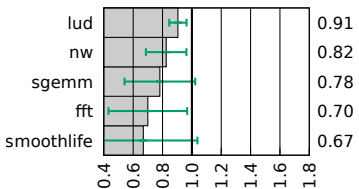
[1] *Single-pass Parallel Prefix Scan with Decoupled Look-back*, NVIDIA Technical Report

Cause: Smaller thread blocks

On AMD GPUs, OpenCL thread blocks are limited to 256 threads.

- May originally have been hardware limit.
- Limit does not exist with HIP, so modern hardware is clearly capable.
  - Fine print applies.
- Affects workloads that benefit from significant amount of intra-group/block communication.

MI100 – OpenCL/HIP

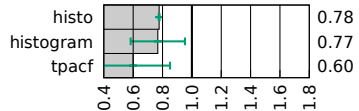


Cause: Imprecise cache information

No predictable way to query L2 cache size on OpenCL.

- `CL_DEVICE_GLOBAL_MEM_CACHE_SIZE`
  - L2 on NVIDIA, L1 on AMD.
- Affects generalised histograms, which uses the cache size in a formula that balances locality and redundant work.[2]

MI100 – OpenCL/HIP



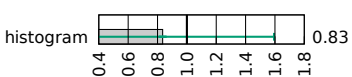
[2] *Compiling Generalized Histograms for GPU*, SC'20

Cause: Imprecise thread information

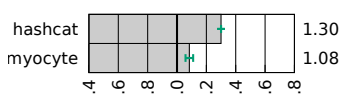
OpenCL does not provide information on how many threads fit on the GPU.

- Futhark makes a heuristic guess instead (1024 per compute unit).
  - Generally *smaller* than the CUDA/HIP-queried number.
- No guarantee that the "correct" number is better than heuristic.

A100 - OpenCL/CUDA



MI100 – OpenCL/HIP

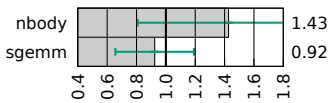


Cause: API overhead

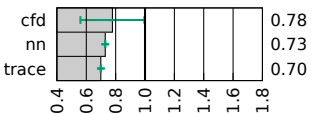
For some benchmarks, the performance difference is not attributable to any measurable GPU operations.

- Enqueuing GPU operations can be relatively costly.
- Only affects workloads with very small absolute runtimes.
  - E.g. 250μs for *trace* on MI100 with OpenCL.
  - See variance for *nbody* on A100 - a single tiny workload.
- *Generally* OpenCL is slower, but there are exceptions.

A100 - OpenCL/CUDA



MI100 – OpenCL/HIP

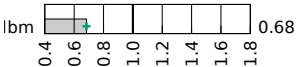


Cause: Bounds checking

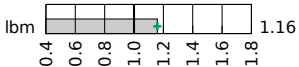
Bounds checking is done with a program transformation that induces somewhat unusual control flow, which sometimes negatively affects the kernel compiler.[3]

- *lbm* is most affected benchmark.
  - Not clear why it is so sensitive.
- Surprisingly not consistent whether OpenCL is the one that gets slower.

A100 - OpenCL/CUDA



MI100 – OpenCL/HIP



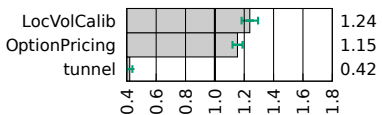
[3] *Bounds Checking on GPU*, HLPP'20

Probably a cause somewhere: Inexplicable behaviour

Sometimes the GPU kernels are just slower for reasons that I am unable to identify.

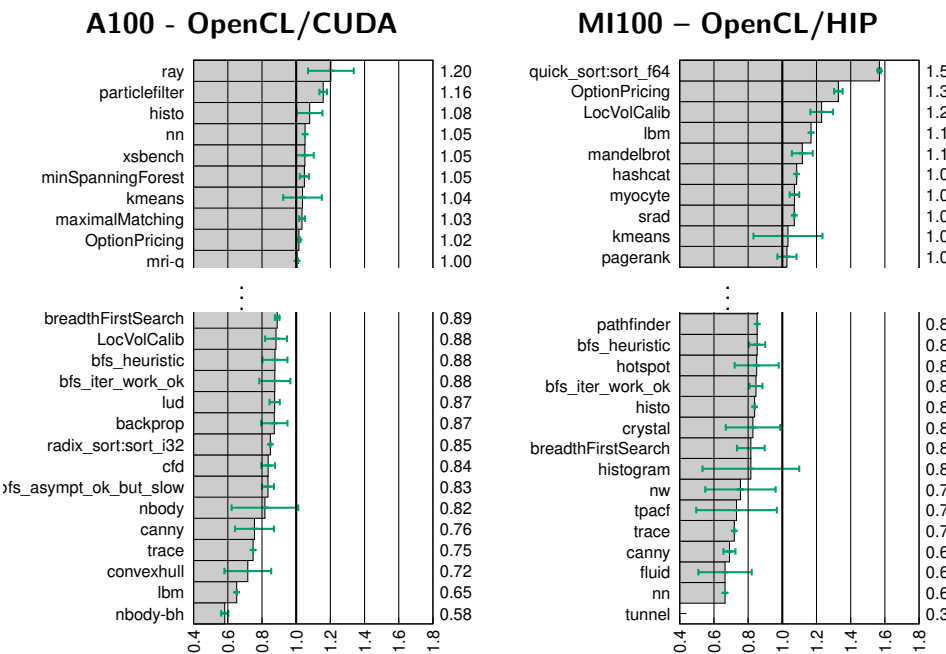
- Performance difference is clearly due to certain compute-bound kernels.
- I suspect minor differences in register allocation or similar.

MI100 – OpenCL/HIP



Can we solve some of these issues?

- Pass `-cl-fp32-correctly-rounded-divide-sqrt` to OpenCL.
- Manually provide hardware information.


A100 - OpenCL/CUDA
ray 1.20
particlefilter 1.16
histo 1.08
nn 1.05
xsbench 1.05
minSpanningForest 1.05
kmeans 1.04
maximalMatching 1.03
OptionPricing 1.02
mri-q 1.00
breadthFirstSearch 0.89
LocVolCalib 0.88
bfs_heuristic 0.88
bfs_iter_work_ok 0.88
lud 0.87
backprop 0.87
radix_sort:sort_i32 0.85
cfd 0.84
bfs_asympt_ok_but_slow 0.83
nbody 0.82
canny 0.76
trace 0.75
convexhull 0.72
lbm 0.65
nbody-bh 0.58
0.4 0.6 0.8 1.0 1.2 1.4 1.6 1.8
MI100 – OpenCL/HIP
quick_sort:sort_f64 1.57
OptionPricing 1.33
LocVolCalib 1.23
lbm 1.17
mandelbrot 1.12
hashcat 1.08
myocyte 1.07
srad 1.07
kmeans 1.03
pagerank 1.03
pathfinder 0.85
bfs_heuristic 0.85
hotspot 0.85
bfs_iter_work_ok 0.85
histo 0.84
crystal 0.83
breadthFirstSearch 0.82
histogram 0.82
nw 0.76
tpacf 0.73
trace 0.72
canny 0.69
fluid 0.67
nn 0.67
tunnel 0.38
0.4 0.6 0.8 1.0 1.2 1.4 1.6 1.8

Conclusions

- It is not difficult to target all of OpenCL, CUDA, and HIP.
  - Assuming you don't need fancy features like dynamic parallelism, tensor cores, warp level operations, ...
  - Probably also the related similar APIs.

Conclusions

- It is not difficult to target all of OpenCL, CUDA, and HIP.
  - Assuming you don't need fancy features like dynamic parallelism, tensor cores, warp level operations, ...
  - Probably also the related similar APIs.
- Performance portability is tricky.
  - Different kernel compiler defaults.
  - OpenCL fails to expose some hardware features.
  - The kernel compilers do their own thing.
- If you only care about NVIDIA/AMD GPUs, targeting OpenCL is perhaps not worth it.
  - Just do CUDA and HIP, they are very similar.

Experimental setup: `https://github.com/diku-dk/futhark-fproper24`
Futhark website: `https://futhark-lang.org`