CSC

ICT Solutions for Brilliant Minds

Porting Codes to LUMI
Cray User Group
May 5th, 2021

George S. Markomanolis
Lead HPC Scientist, CSC – IT Center for Science Ltd.

Outline

- LUMI
- Approaches to port codes on LUMI
- Benchmarking
- Profiling
- Tuning

LUMI

LUMI, the Queen of the North

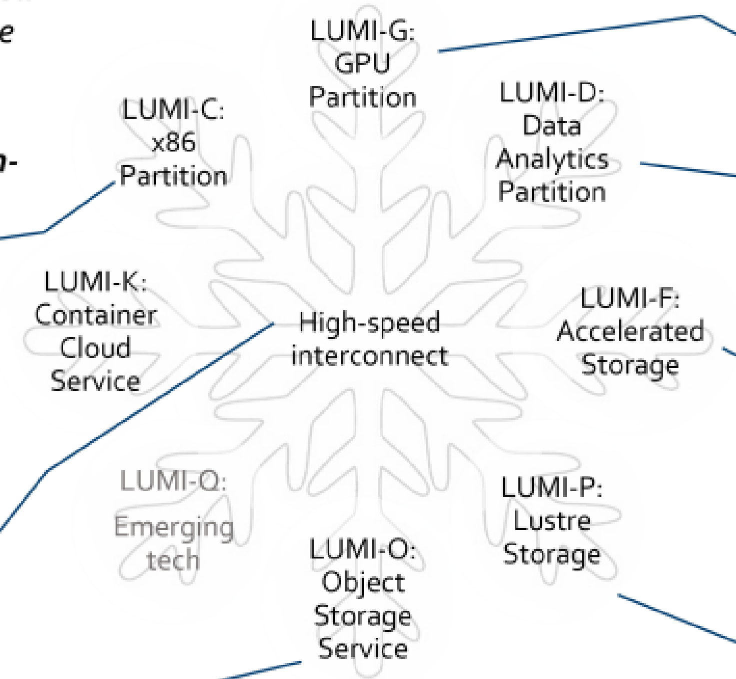
L U M I

LUMI is a Tier-0 ***GPU-accelerated supercomputer*** *that enables the convergence of* ***high-performance computing***, ***artificial intelligence***, *and* ***high-performance data analytics***.

- Supplementary CPU partition
- ~200,000 AMD EPYC CPU cores

Possibility for combining different resources within a single run. HPE Slingshot technology.

30 PB encrypted object storage (Ceph) for storing, sharing and staging data



Tier-0 GPU partition: over 550 Pflop/s powered by AMD Instinct GPUs

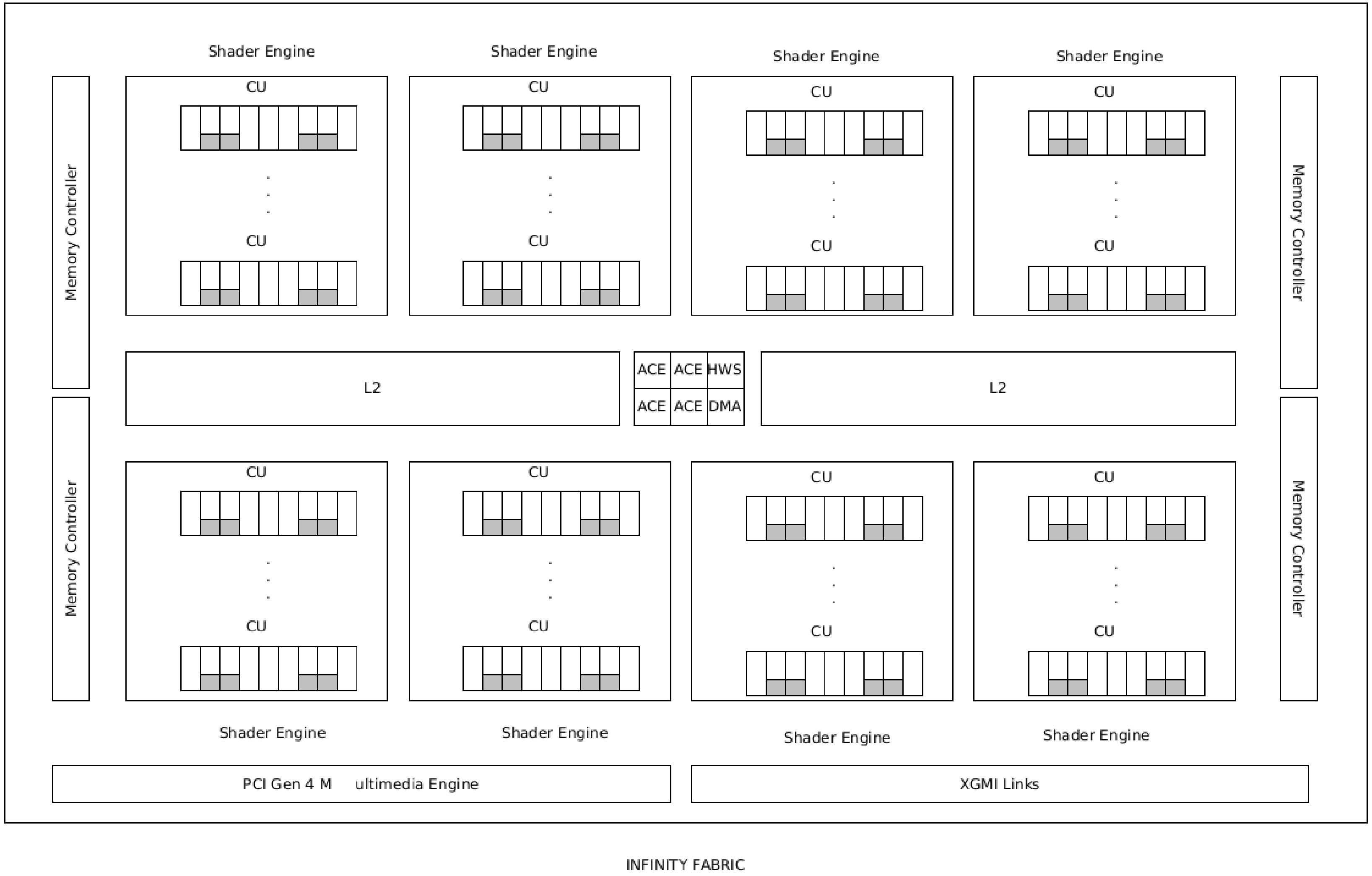
Interactive partition with 32 TB of memory and graphics GPUs for data analytics and visualization

7 PB Flash-based storage layer with extreme I/O bandwidth of 2 TB/s and IOPS capability. Cray ClusterStor E1000.

80 PB parallel file system

www.lumi-supercomputer.eu #lumisupercomputer #lumieurohpc

AMD GPUs (MI100 example)

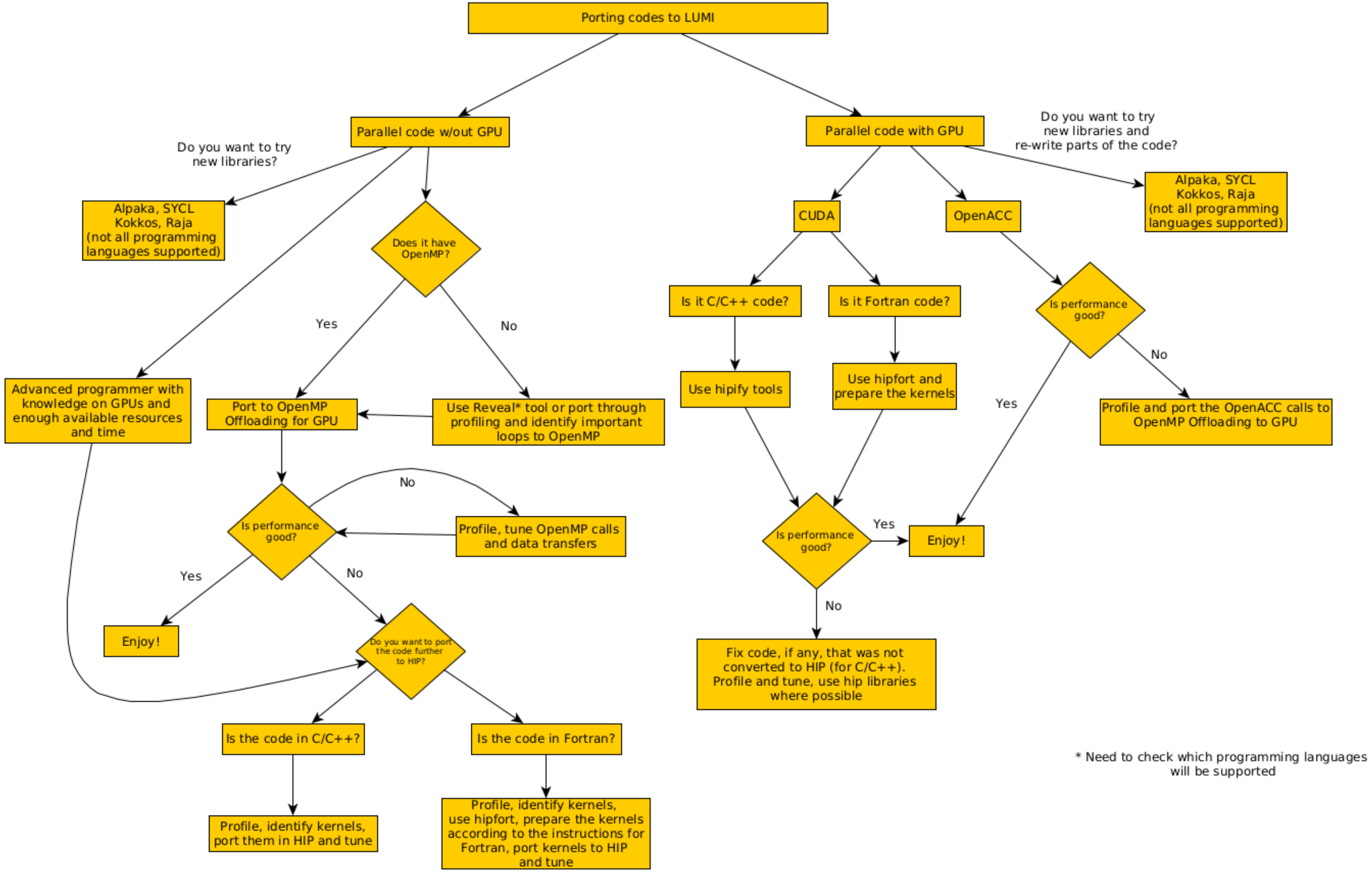


LUMI will have a different GPU

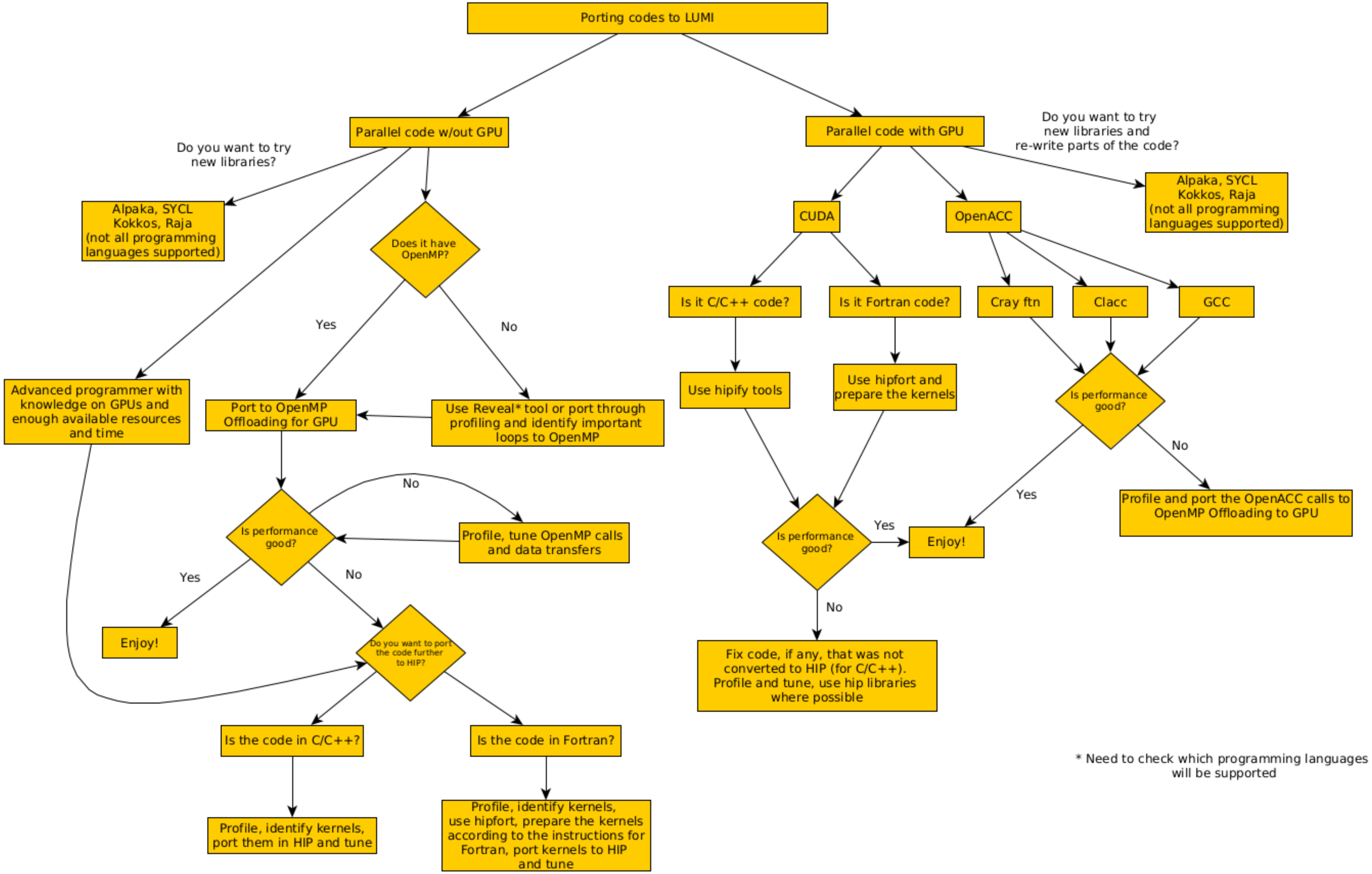
Differences between HIP and CUDA

- AMD GCN hardware wavefronts size is 64 (like warp for CUDA), some terminology is different
- Some CUDA library functions do not have AMD equivalents
- Shared memory and registers per thread can differ between AMD and NVIDIA hardware
- ROCM 4.1 just released and improves some functionalities (Warp-Level primitives was not supported by HIP but maybe this is improved)

Porting Codes to LUMI



Porting Codes to LUMI (experimental)



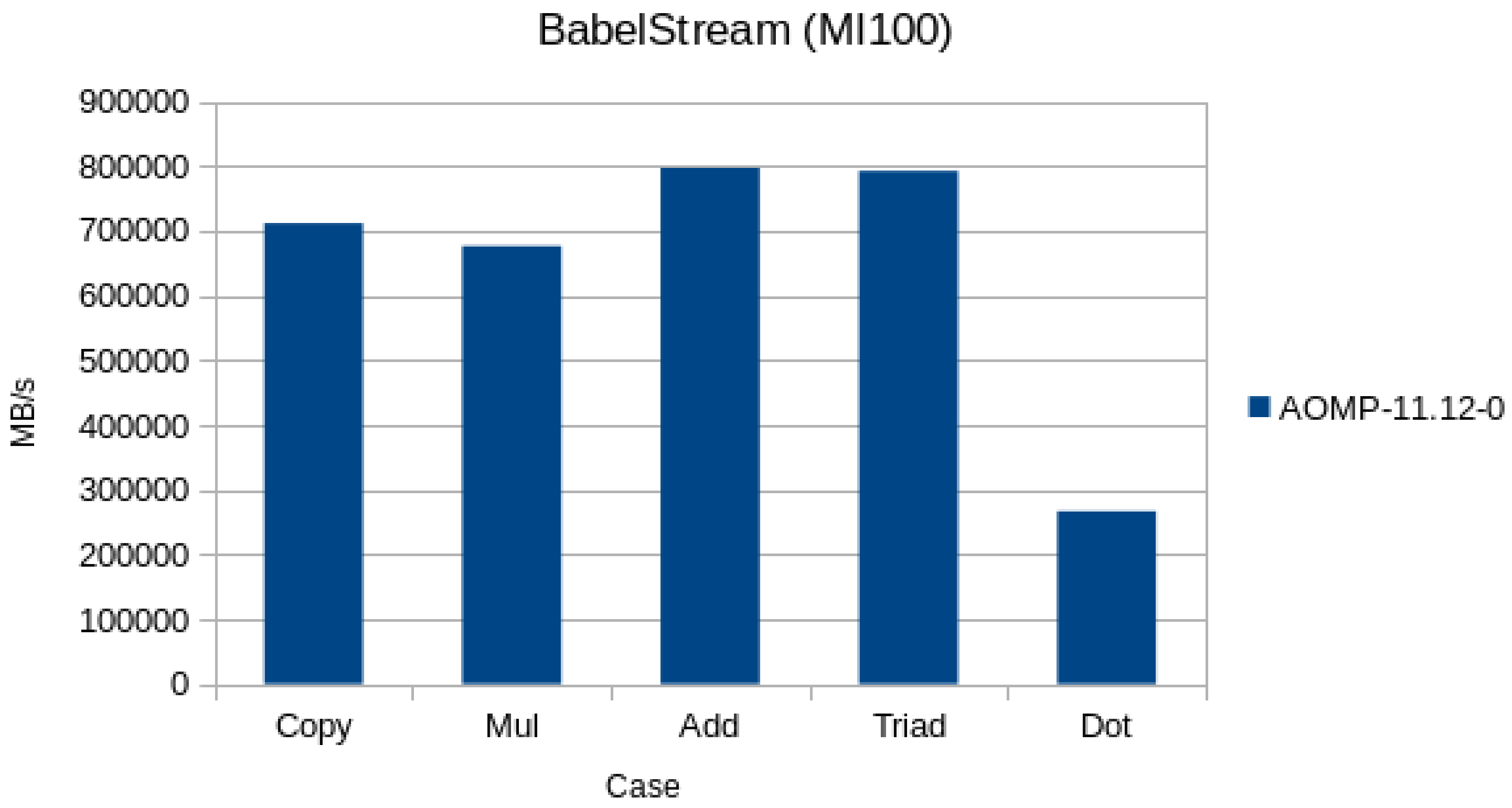
* Need to check which programming languages will be supported

OpenMP Offload

- There are many tutorials about OpenMP Offloading
- Some basic OpenMP useful constructs:
  - #pragma omp target enter/exit data map
  - #pragma omp target teams distribute parallel for simd
  - thread_limit(X) num_teams(Y)

- There are a lot of tutorials about OpenMP Offloading
- OpenMP 5.0, what is new: https://www.openmp.org/spec-html/5.0/openmpse71.html
- OpenMP 5.1, what is new: https://www.openmp.org/wp-content/uploads/OpenMP-API-Additional-Definitions-2-0.pdf
- OpenMP 5.0 tutorial: https://ecpannualmeeting.com/assets/overview/sessions/ff2020%20ECP-Tutorial-with-ECP-template.pdf

BabelStream (default settings)



Improving performance on BabelStream for MI100

- Original call:

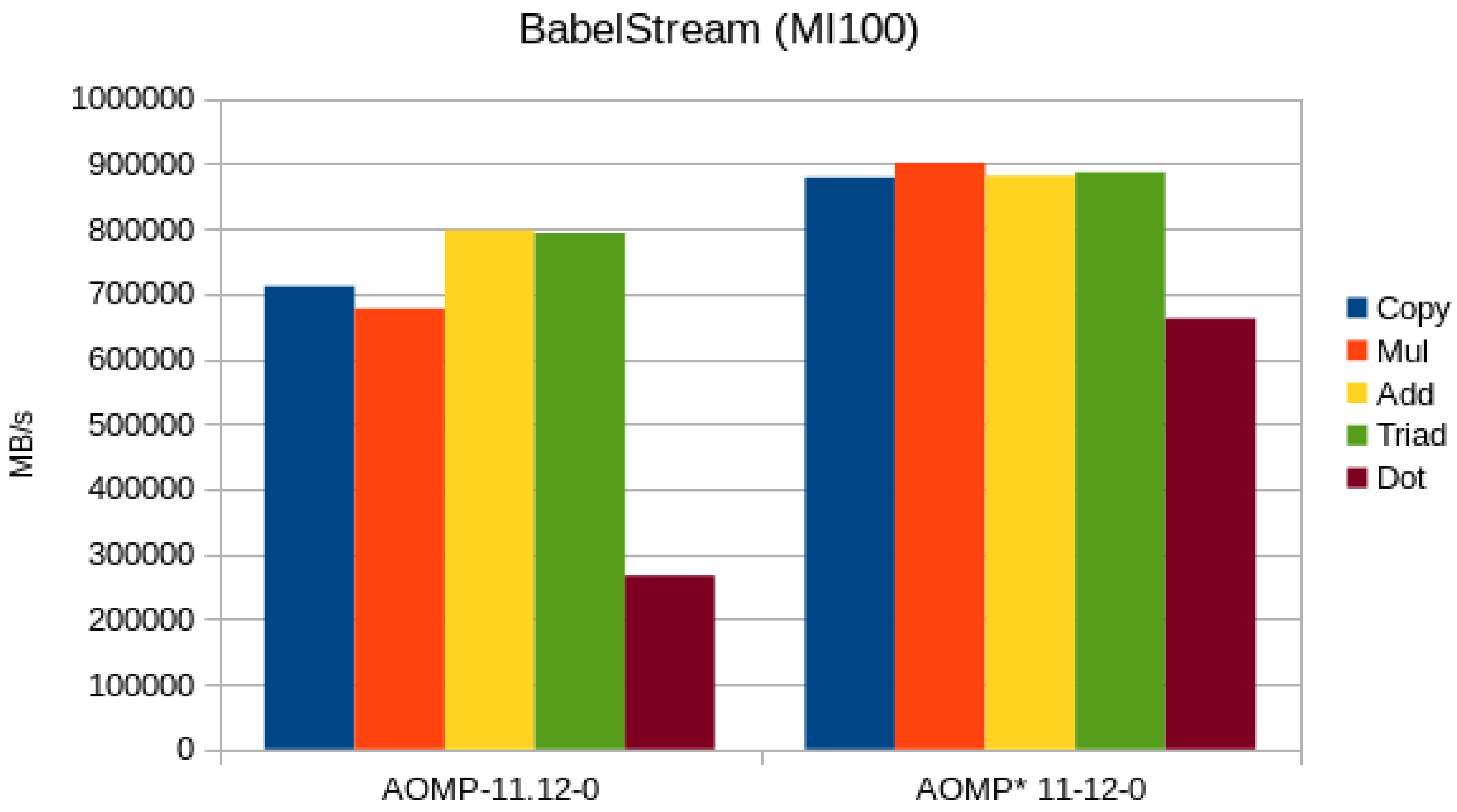
```
#pragma omp target teams distribute parallel for simd
```

- Optimized call

```
#pragma omp target teams distribute parallel for simd thread_limit(256) num_teams(240)
```

- For the dot case we used 720 teams

BabelStream, tune AOMP



Introduction to HIP

- HIP: Heterogeneous Interface for Portability is developed by AMD to program on AMD GPUs
- It is a C++ runtime API and it supports both AMD and NVIDIA platforms
- HIP is similar to CUDA and there is no performance overhead on NVIDIA GPUs
- Many well-known libraries have been ported on HIP
- New projects or porting from CUDA, could be developed directly in HIP

https://github.com/ROCm-Developer-Tools/HIP

Hipify

- Hipify tools convert automatically CUDA codes
- It is possible that not all the code is converted, the remaining needs the implementation of the developer
- Hipify-perl: text-based search and replace
- Hipify-clang: source-to-source translator that uses clang compiler
- Porting guide: https://github.com/ROCm-Developer-Tools/HIP/blob/main/docs/markdown/hip_porting_guide.md

Hipify-perl

- It can scan directories and converts CUDA codes with replacement of the cuda to hip (sed –e ’s/cuda/hip/g’)

$ hipify-perl --inplace filename

It modifies the filename input inplace, replacing input with hipified output, save backup in .prehip file.

$ hipconvertinplace-perl.sh directory

It converts all the related files that are located inside the directory

Hipify-perl (cont).

1) $ ls src/

Makefile.am matMulAB.c matMulAB.h matMul.c

2) $ hipconvertinplace-perl.sh src

3) $ ls src/

Makefile.am matMulAB.c matMulAB.c.prehip matMulAB.h matMulAB.h.prehip matMul.c matMul.c.prehip

No compilation took place, just convertion.

Hipify-perl (cont).

- The hipify-perl will return a report for each file, and it looks like this:

```
info: TOTAL-converted 53 CUDA->HIP refs ( error:0 init:0 version:0 device:1 ... library:16
... numeric_literal:12 define:0 extern_shared:0 kernel_launch:0 )
warn:0 LOC:888
kernels (0 total) :
hipFree 18
HIPBLAS_STATUS_SUCCESS 6
hipSuccess 4
hipMalloc 3
HIPBLAS_OP_N 2
hipDeviceSynchronize 1
hip_runtime 1
```

Differences between CUDA and HIP API

CUDA	HIP
#include "cuda.h"	#include "hip/hip_runtime.h"
cudaMalloc(&d_x, N*sizeof(double));	hipMalloc(&d_x, N*sizeof(double));
cudaMemcpy(d_x,x,N*sizeof(double), cudaMemcpyHostToDevice);	hipMemcpy(d_x,x,N*sizeof(double), hipMemcpyHostToDevice);
cudaDeviceSynchronize();	hipDeviceSynchronize();

Launching kernel with CUDA and HIP

CUDA

```
kernel_name <<<gridsize, blocksize,
               shared_mem_size,
               stream>>>
              (arg0, arg1, ...);
```

HIP

```
hipLaunchKernelGGL(kernel_name,
                   gridsize,
                   blocksize,
                   shared_mem_size,
                   stream,
                   arg0, arg1, ... );
```

Libraries (not exhaustive)

NVIDIA	HIP	ROCm	Description
cuBLAS	hipBLAS	rocBLAS	Basic Linear Algebra Subroutines
cuRAND	hipRAND	rocRAND	Random Number Generator Library
cuFFT	hipFFT	rocFFT	Fast Fourier Transfer Library
cuSPARSE	hipSPARSE	rocSPARSE	Sparse BLAS + SPMV
NCCL		RCCL	Communications Primitives Library based on the MPI equivalents
CUB	hipCUB	rocPRIM	Low Level Optimized Parallel Primitives

Benchmark MatMul cuBLAS, hipBLAS

- Use the benchmark https://github.com/pc2/OMP-Offloading
- Matrix multiplication of 2048 x 2048, single precision
- All the CUDA calls were converted and it was linked with hipBlas
- CUDA (V100)

matMulAB (10) : 1011.2 GFLOPS 12430.1 GFLOPS

- HIP (MI100)

matMulAB (10) : 2327.6 GFLOPS 22216.7 GFLOPS

- MI100 achieves close to the theoretical peak for single precision

N-BODY SIMULATION

- N-Body Simulation (https://github.com/themathgeek13/N-Body-Simulations-CUDA) AllPairs_N2
- 171 CUDA calls converted to HIP without issues, close to 1000 lines of code
- 32768 number of small particles, 2000 time steps

CUDA execution time on V100 : 68.5 seconds

HIP execution time on MI100: 95.57 seconds, 39.5% worse performance

- Tune the number of threads per block to 256 instead of 1024, then:

HIP execution time on Mi100: 54.32 seconds, 26.1% better performance than V100

Fortran

- First Scenario: Fortran + CUDA C/C++
  - Assuming there is no CUDA code in the Fortran files.
  - Hipify CUDA
  - Compile and link with hipcc
- Second Scenario: CUDA Fortran
  - There is no HIP equivalent
  - HIP functions are callable from C, using `extern C`
  - See hipfort

Hipfort

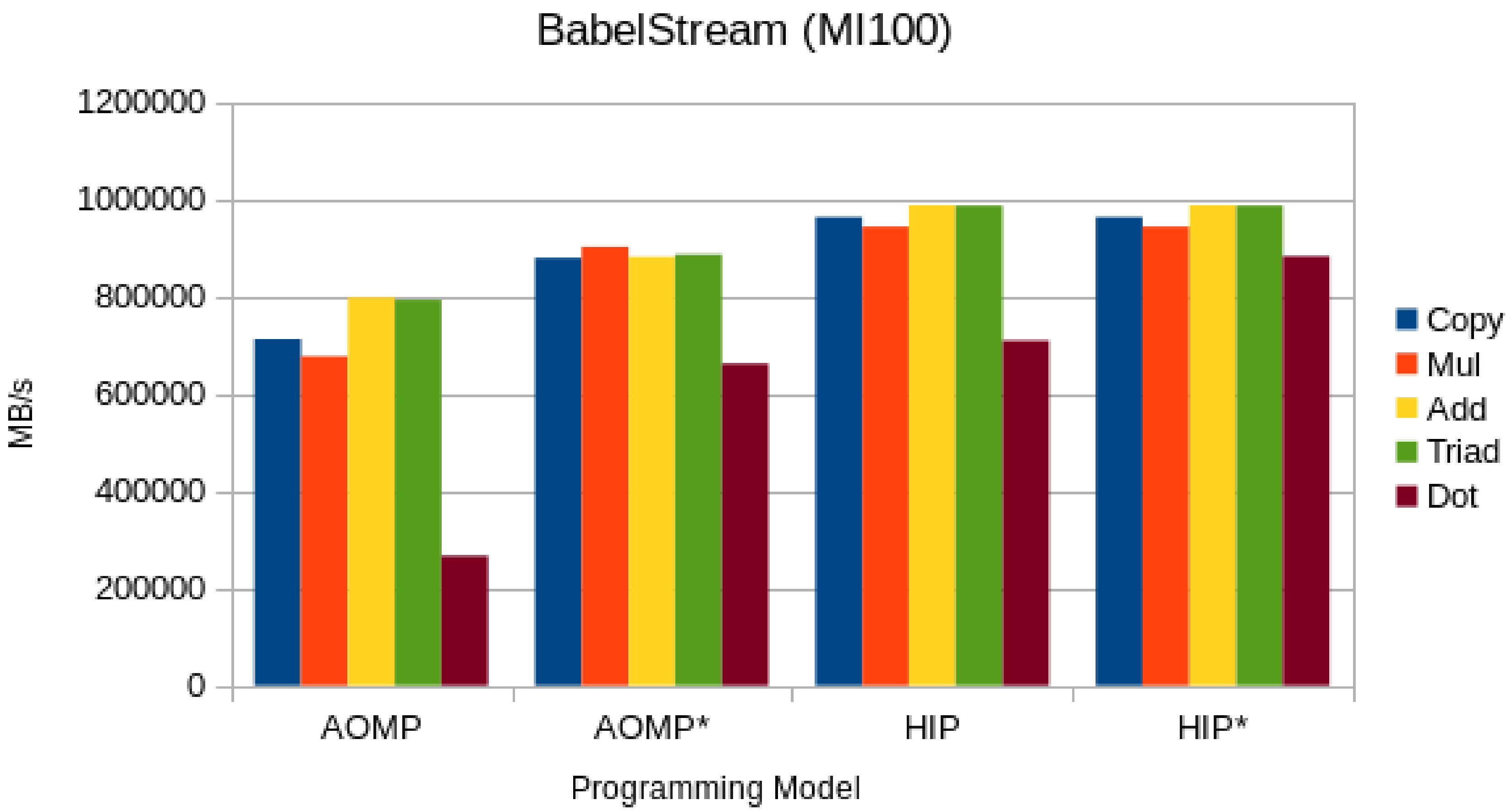
- The approach to port Fortran codes on AMD GPUs is different, the hipify tool does not support it.
- We need to use hipfort, a Fortran interface library for GPU kernel *
- Steps:
  1) We write the kernels in a new C++ file
  2) Wrap the kernel launch in a C function
  3) Use Fortran 2003 C binding to call the function
  4) Things **could** change in the future
- Use OpenMP offload to GPUs

* https://github.com/ROCmSoftwarePlatform/hipfort

Fortran CUDA example

- Saxpy example
- Fortran CUDA, 29 lines of code
- Ported to HIP manually, two files of 52 lines, with more than 20 new lines.
- Quite a lot of changes for such a small code.
- Should we try to use OpenMP offload before we try to HIP the code?
- Need to adjust Makefile to compile the multiple files
- Example of Fortran with HIP: https://github.com/cschpc/lumi/tree/main/hipfort

BabelStream on MI100 (HIP vs AOMP)



Megahip

- https://github.com/zjin-lcf/oneAPI-DirectProgramming
- 115 Applications/Examples with CUDA, SYCL, OpenMP offload and HIP
- Testing hipify tool, create a megahip script to convert all the CUDA examples to HIP
- ./megahip.sh

  3287 CUDA calls were converted to HIP

  115 applications totally 45692 lines of code, there are warnings for 4 of them, there are totally 24 warnings that something was wrong, check warnings.txt
  Application Success 96.5217%
  Conversion Success 99.2699%

OpenACC

- GNU will provide OpenACC (Mentor Graphics contract, now called Siemens EDA)
- HPE will use the provided GNU compiler for OpenACC support
- HPE will support for OpenACC v2.0 for Fortran. This is quite old OpenACC version.
- Clacc from ORNL: https://github.com/llvm-doe-org/llvm-project/tree/clacc/master
  OpenACC from LLVM only for C (Fortran and C++ in the future)
  - Translate OpenACC to OpenMP Offload

Clacc

- It supports C programming language, Fortran is on the way, C++ not started(??) yet

$ clang -fopenacc-print=omp *-fopenacc-structured-ref-count-omp=no-hold -fopenacc-present-omp=no-present* jacobi.c

Original code:

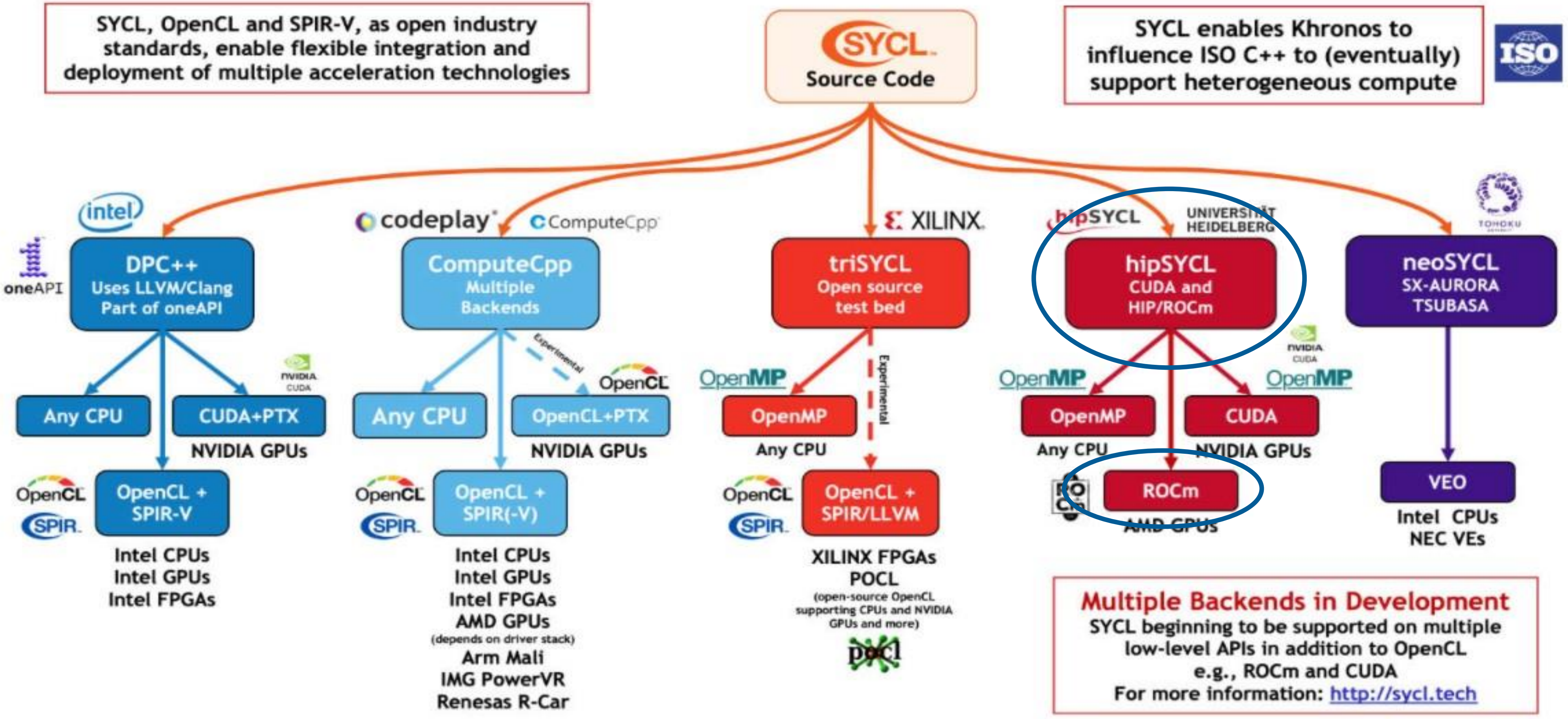
```
#pragma acc parallel loop reduction(max:lnorm) private(i,j) present(newarr, oldarr) collapse(2)
for (i = 1; i < nx + 1; i++) {
   for (j = 1; j < ny + 1; j++) {
```

New code:

```
#pragma omp target teams map(alloc: newarr,oldarr) map(tofrom: lnorm) shared(newarr,oldarr)
firstprivate(nx,ny,factor) reduction(max: lnorm) \
#pragma omp distribute private(i,j) collapse(2)
for (i = 1; i < nx + 1; i++) {
   for (j = 1; j < ny + 1; j++) {
```

SYCL Implementations in Development

SYCL implementations are available from an increasing number of vendors, including adding support for diverse acceleration API back-ends in addition to OpenCL.



hipSYCL and SYCL 2020: https://github.com/hipSYCL/featuresupport

SAXPY SYCL

create queue

```
sycl::queue q(sycl::default_selector{});
const float A(aval);
sycl::buffer<float,1> d_X { h_X.data(), sycl::range<1>(h_X.size()) };
sycl::buffer<float,1> d_Y { h_Y.data(), sycl::range<1>(h_Y.size()) };
sycl::buffer<float,1> d_Z { h_Z.data(), sycl::range<1>(h_Z.size()) };
q.submit([&](sycl::handler& h) {
 auto X = d_X.template get_access<sycl::access::mode::read>(h);
 auto Y = d_Y.template get_access<sycl::access::mode::read>(h);
 auto Z = d_Z.template get_access<sycl::access::mode::read_write>(h);
 h.parallel_for<class nstream>( sycl::range<1>{length}, [=] (sycl::id<1> it) {
       const int i = it[0];
       Z[i] = A * X[i] + Y[i];
   });
 });
q.wait();
```

```
sycl::queue q(sycl::host_selector{});
sycl::queue q(sycl::cpu_selector{});
sycl::queue q(sycl::gpu_selector{});
sycl::queue q(sycl::accelerator_selector{});
```

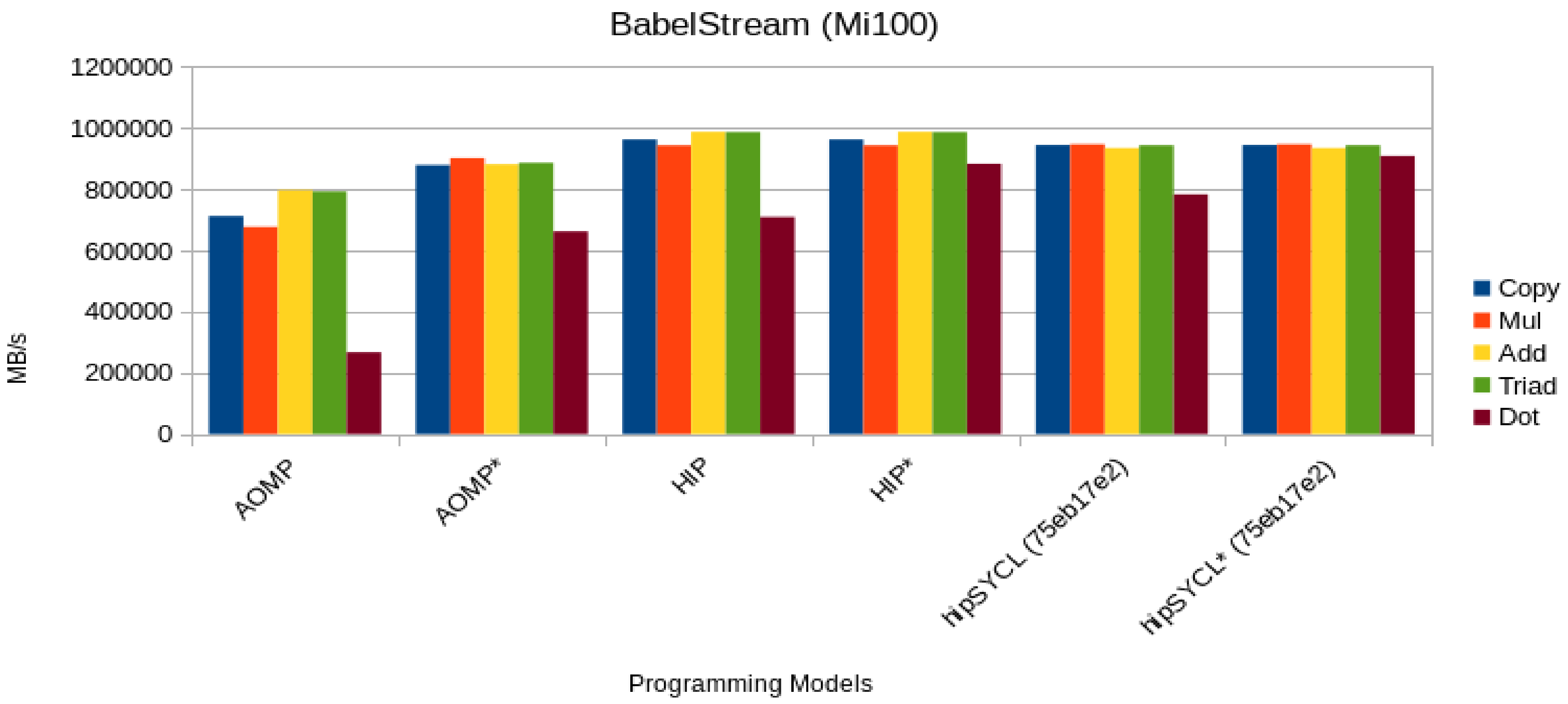
Declare SYCL buffers to handle data on the device

SYCL accesors they generate a dataflow graph that the compiler and runtime can use to move data across devices

SYCL 2020

```
q.parallel_for( sycl::range<1>{length}, [=] (sycl::id<1> i) {
       d_Z[i] += A * d_X[i] + d_Y[i];
   });
```

Results of BabelStream on Mi100 (AOMP vs HIP vs hipSYCL)



Profiling/Debugging

- AMD provides APIs for profiling and debugging
- Some simple environment variables such as ***AMD_LOG_LEVEL=4*** will provide some information.
- More information about a hipMemcpy error:

```
hipError_t err = hipMemcpy(c,c_d,nBytes,hipMemcpyDeviceToHost);
 printf("%s ",hipGetErrorString(err));
```

- ROCprofiler, ROCgdb
- Some profiling tools work with AMD GPUs

TAU profiling

Name △	Exclusive TAUGPU...	Inclusive TAUGPU_...	Calls	Child Calls
.TAU application	0.31	0.633	1	501
void add_kernel<double>(double const*, double const*, double*) [clone .kd]	0.08	0.08	100	0
void copy_kernel<double>(double const*, double*) [clone .kd]	0.052	0.052	100	0
void dot_kernel<double>(double const*, double const*, double*, int) [clone .kd]	0.059	0.059	100	0
void init_kernel<double>(double*, double*, double*, double, double, double) [clone .k	0.001	0.001	1	0
void mul_kernel<double>(double*, double const*) [clone .kd]	0.052	0.052	100	0
void triad_kernel<double>(double*, double const*, double const*) [clone .kd]	0.08	0.08	100	0

tau_exec -T rocm,serial -rocm ./hip-stream

Rocprof

- Statistics for kernels and names (see the created csv fie):

```
rocprof --stats ./hip-stream
```

- Create a metrics.txt file with content (choose metrics):

```
pmc: GPUBusy Wavefronts VALUInsts SALUInsts SFetchInsts MemUnitStalled VALUUtilization
VALUBusy SALUBusy WriteUnitStalled
range: 0:100
gpu: 0
kernel: add_kernel copy_kernel triad_kernel dot_kernel mul_kernel
```

```
rocprof -i metrics.txt ./hip-stream
```

KernelName	GPUBusy	Wavefronts	VALUInsts	SALUInsts	SFetchInsts	MemUnitStal	VALUUtilizati	VALUBusy	SALUBusy	WriteUnitStalled
copy_kernel<	100	524288	9	2	2	36	100	6	1	14
mul_kernel<d	100	524288	10	4	2	34	100	7	2	14
add_kernel<d	100	524288	13	2	3	24	100	5	0	0
triad_kernel<	100	524288	13	4	3	18	99	5	1	0
dot_kernel<d	100	4096	1727	289	5	0	99	7	1	0

Tuning

- Multiple wavefronts per compute unit (CU) is important to hide latency and instruction throughput
- Tune number of threads per block, number of teams for OpenMP offloading etc.
- Memory coalescing increases bandwidth
- Unrolling loops allow compiler to prefetch data
- Small kernels can cause latency overhead, adjust the workload
- Use of Local Data Share (LDS) memory

Conclusion/Future work

- A code written in C/C++ and MPI+OpenMP is a bit easier to be ported to OpenMP offload compared to other approaches.
- The hipSYCL could be a good option considering that the code is in C++. Good support from hipSYCL.
- There can be challenges, depending on the code and what GPU functionalities are integrated to an application
- It will be required to tune the code for high occupancy
- Profiling should be used to identify bottlenecks
- Track historical performance among new compilers
- GCC for OpenACC and OpenMP Offloading for AMD GPUs
- Tracking how profiling tools work on AMD GPUs
- We have trained more than 80 people on HIP porting: http://github.com/csc-training/hip

Acknowledgements

- My colleagues from CSC
- Nicholas Malaya from AMD for the many conversations and emails that we have exchanged
- Michael Klemm from AMD for the OpenMP discussions
- Many people from AMD discussing about issues and future
- HPE for the conversations and presentations

CSC

Questions?

Georgios.Markomanolis@csc.fi

facebook.com/CSCfi

twitter.com/CSCfi

youtube.com/CSCfi

linkedin.com/company/csc---it-center-for-science

github.com/CSCfi