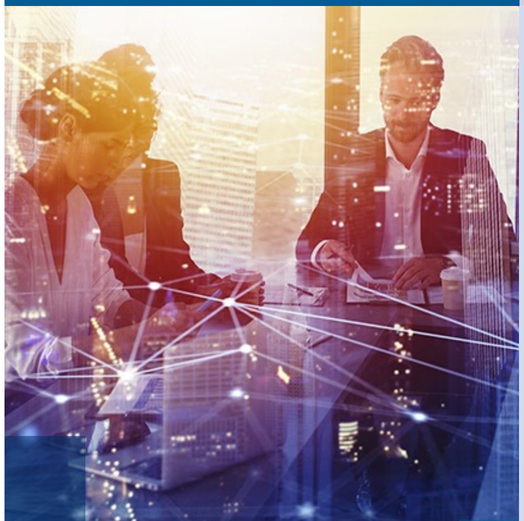
CSC

ICT Solutions for Brilliant Minds



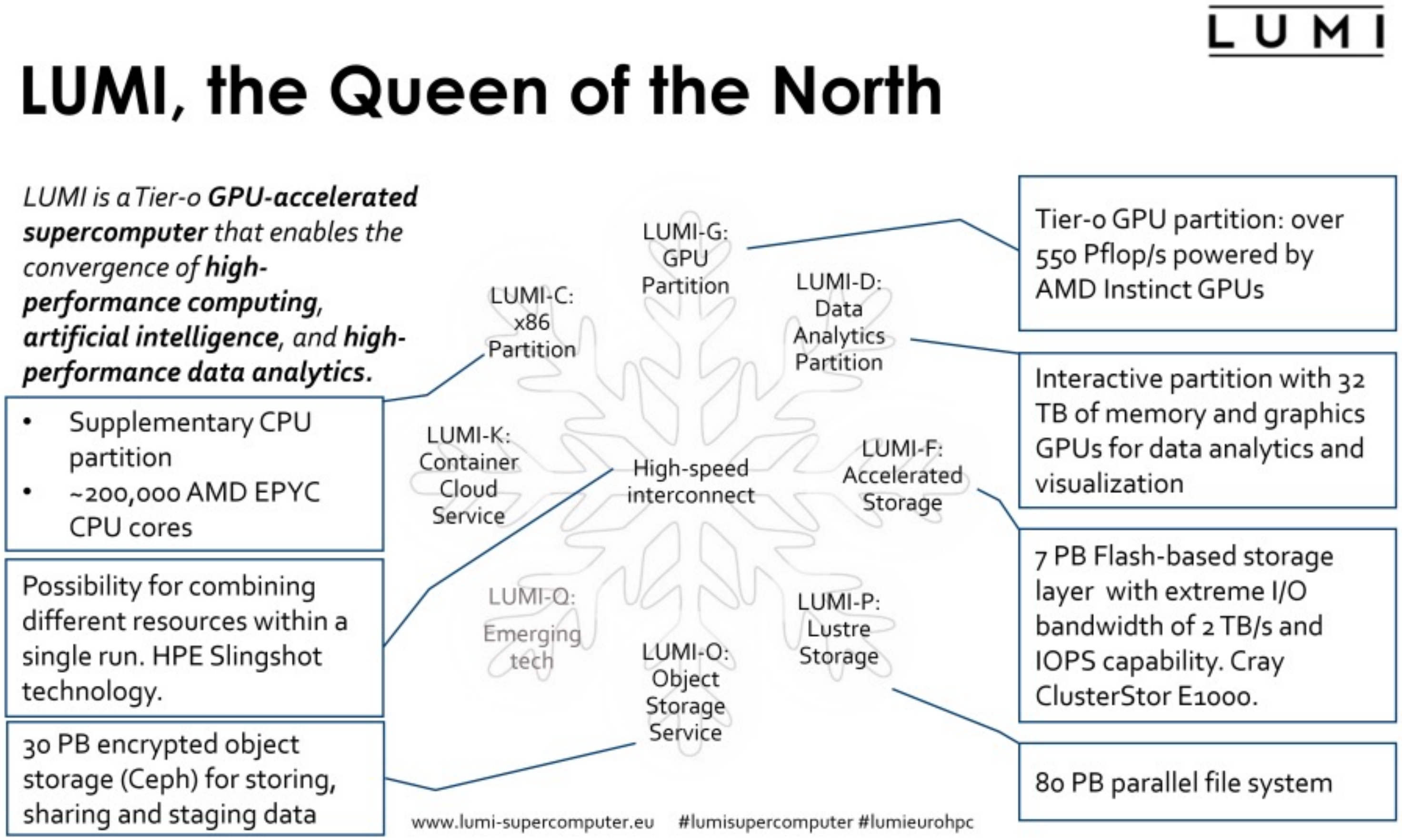
Utilizing AMD GPUs: Tuning, programming models, and roadmap
FOSDEM’22 HPC, Big Data and Data Science devroom

February 6th, 2022

George S. Markomanolis

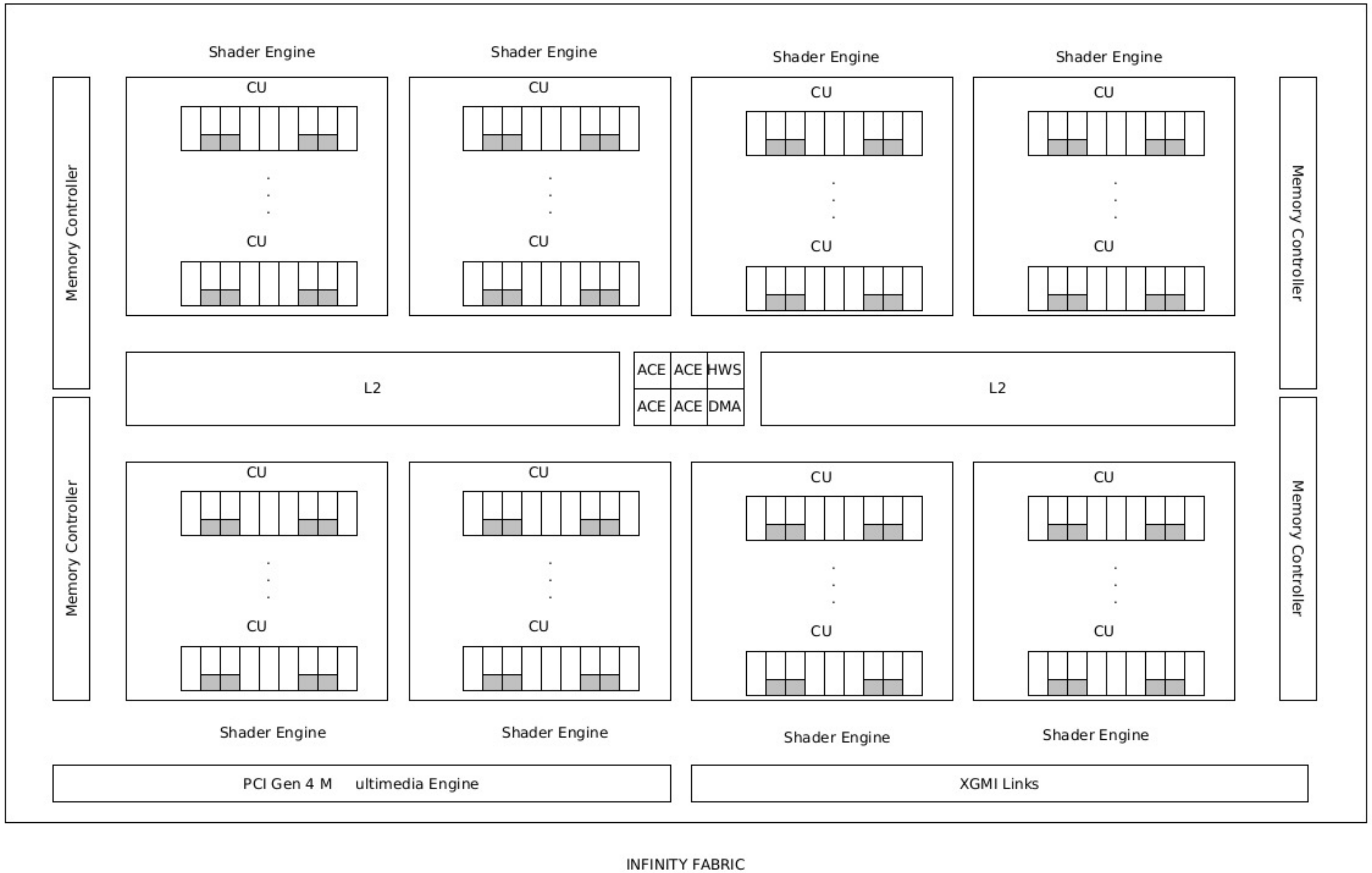
Lead HPC Scientist, CSC – IT Center For Science Ltd.

LUMI


L U M I
LUMI, the Queen of the North
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
LUMI-G: GPU Partition
LUMI-C: x86 Partition
LUMI-D: Data Analytics Partition
LUMI-K: Container Cloud Service
High-speed interconnect
LUMI-F: Accelerated Storage
LUMI-Q: Emerging tech
LUMI-O: Object Storage Service
LUMI-P: Lustre Storage
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

csc

AMD GPUs (MI100 example)



AMD MI100

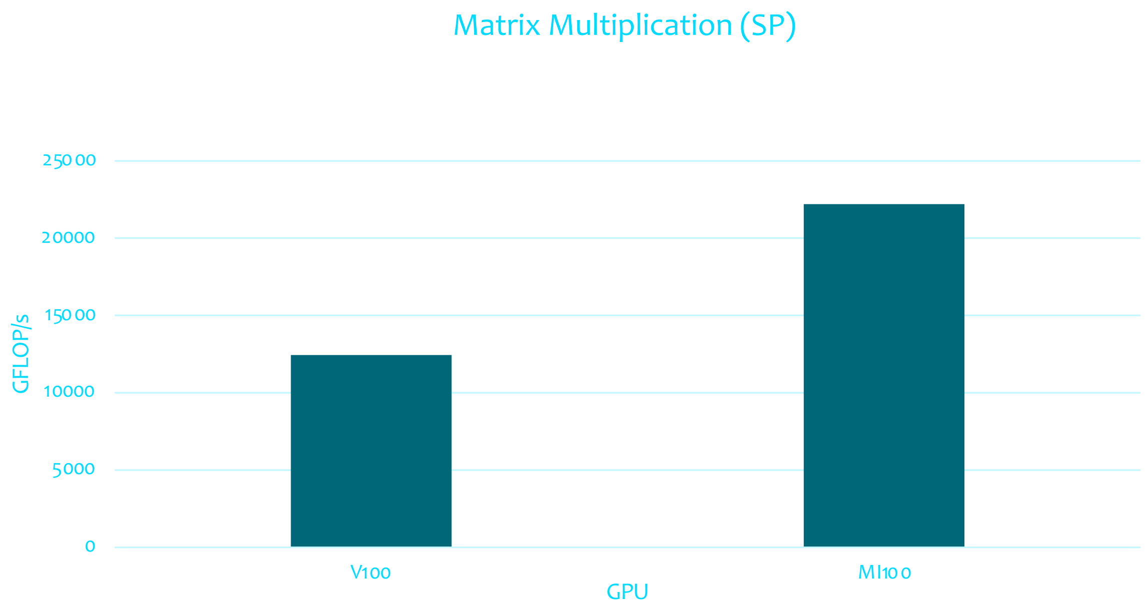
Introduction to HIP

- Radeon Open Compute Platform (ROCm)
- 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
- The supported CUDA API is called with HIP prefix (cudamalloc -> hipmalloc)

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

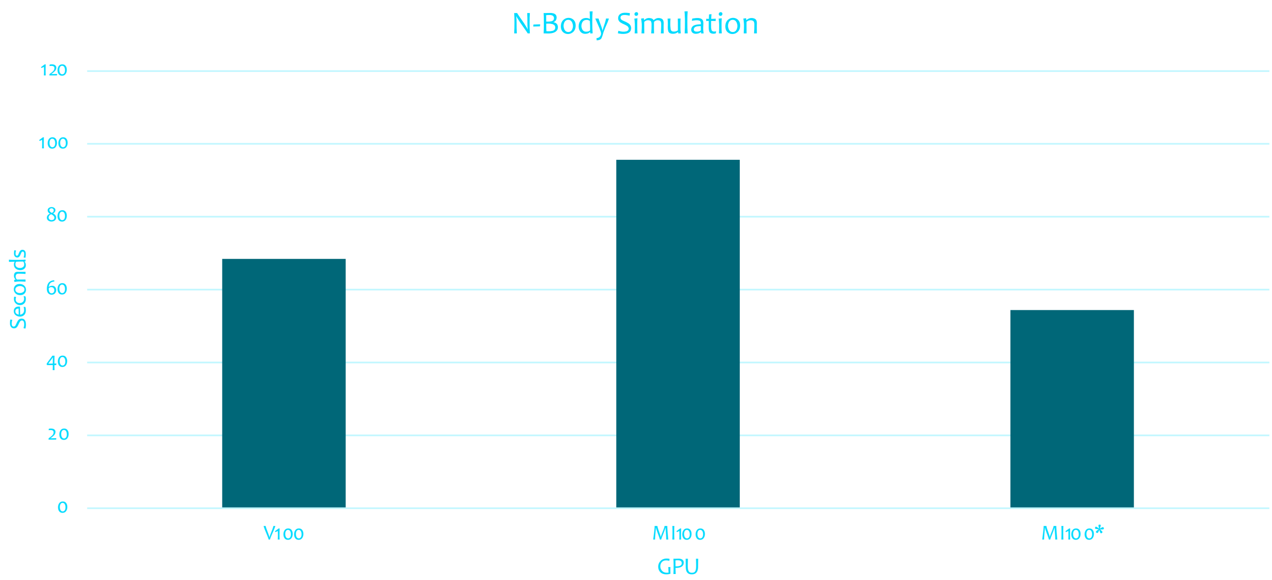
Benchmark MatMul cuBLAS, hipBLAS

- 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



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
- Tune the number of threads equal to 256 than 1024 default at ROCm 4.1



BabelStream

- A memory bound benchmark from the university of Bristol
- Five kernels
  - add (a[i]=b[i]+c[i])
  - multiply (a[i]=b*c[i])
  - copy (a[i]=b[i])
  - triad (a[i]=b[i]+d*c[i])
  - dot (sum = sum+d*c[i])

Improving OpenMP performance on BabelStream for MI100

- Original call:

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

- Optimized call

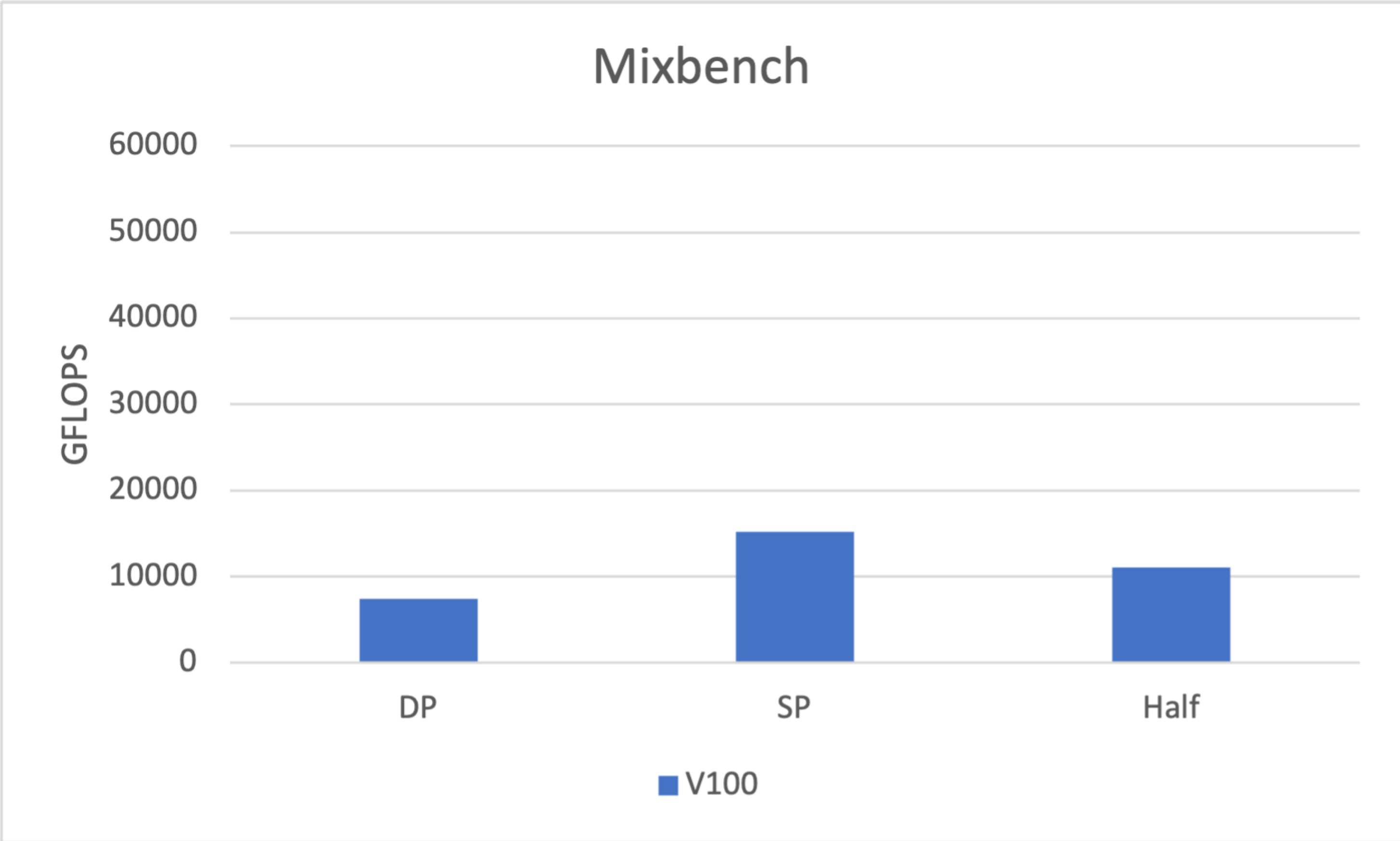
```
#pragma omp target teams distribute parallel for simd thread_limit(256) num_teams(240)
```

- For the dot kernel we used 720 teams

Mixbench

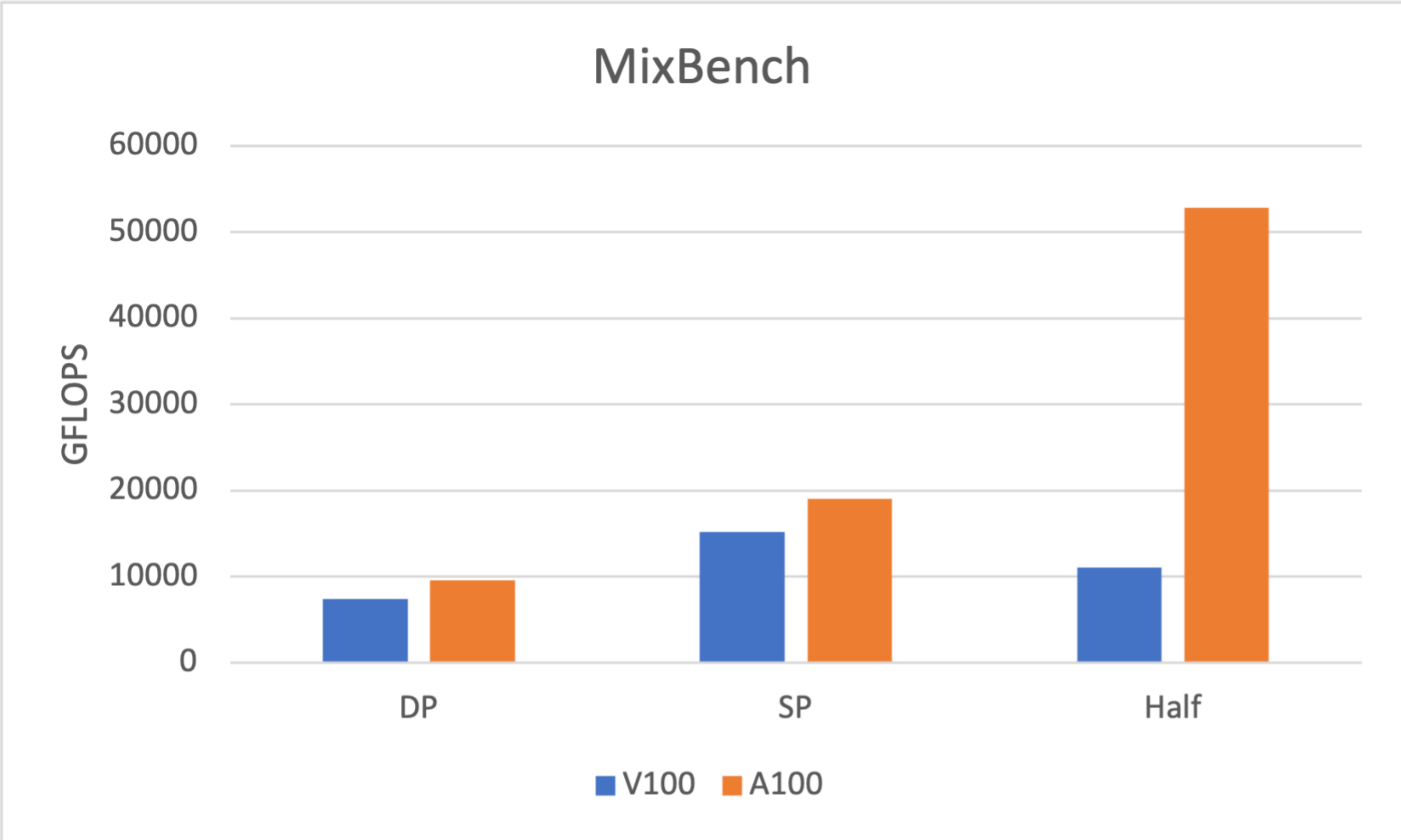
- The purpose of this benchmark tool is to evaluate performance bounds of GPUs on mixed operational intensity kernels.
- The executed kernel is customized on a range of different operational intensity values.
- Supported programming models: CUDA, HIP, OpenCL and SYCL
- We use three types of experiments combined with global memory accesses:
  - Single precision Flops (multiply-additions)
  - Double precision Flops (multiply-additions)
  - Half precision Flops (multiply-additions)
- Following results present peak performance
- Source: https://github.com/ekondis/mixbench

Mixbench

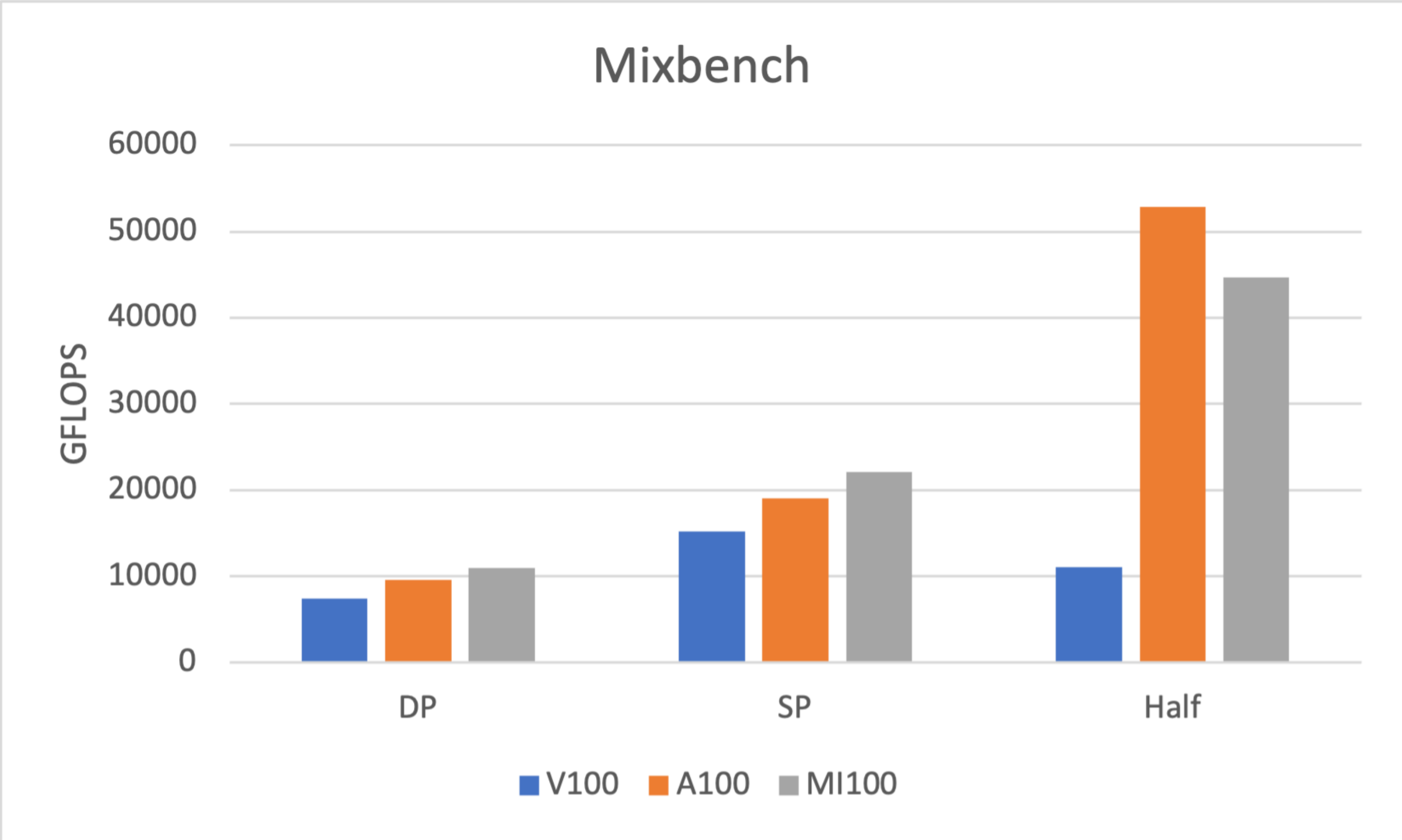


Mixbench
60000
50000
40000
30000
20000
10000
0
GFLOPS
DP
SP
Half
V100

Mixbench


MixBench
GFLOPS
60000
50000
40000
30000
20000
10000
0
DP
SP
Half
V100
A100

Mixbench


Mixbench
60000
50000
40000
30000
20000
10000
0
GFLOPS
DP
SP
Half
V100
A100
MI100

Programming Models

- We have utilized with success at least the following programming models/interfaces on AMD MI100 GPU:
  - HIP
  - OpenMP Offloading
  - hipSYCL
  - Kokkos
  - Alpaka

SYCL (hipSYCL)

- C++ Single-source Heterogeneous Programming for Acceleration Offload
- Generic programming with templates and lambda functions
- Big momentum currently, NERSC, ALCF, Codeplay partnership
- SYCL 2020 specification was announced early 2021
- Terminology: Unified Shared Memory (USM), buffer, accessor, data movement, queue
- hipSYCL supports CPU, AMD/NVIDIA GPUs, Intel GPU (experimental)
- https://github.com/illuhad/hipSYCL

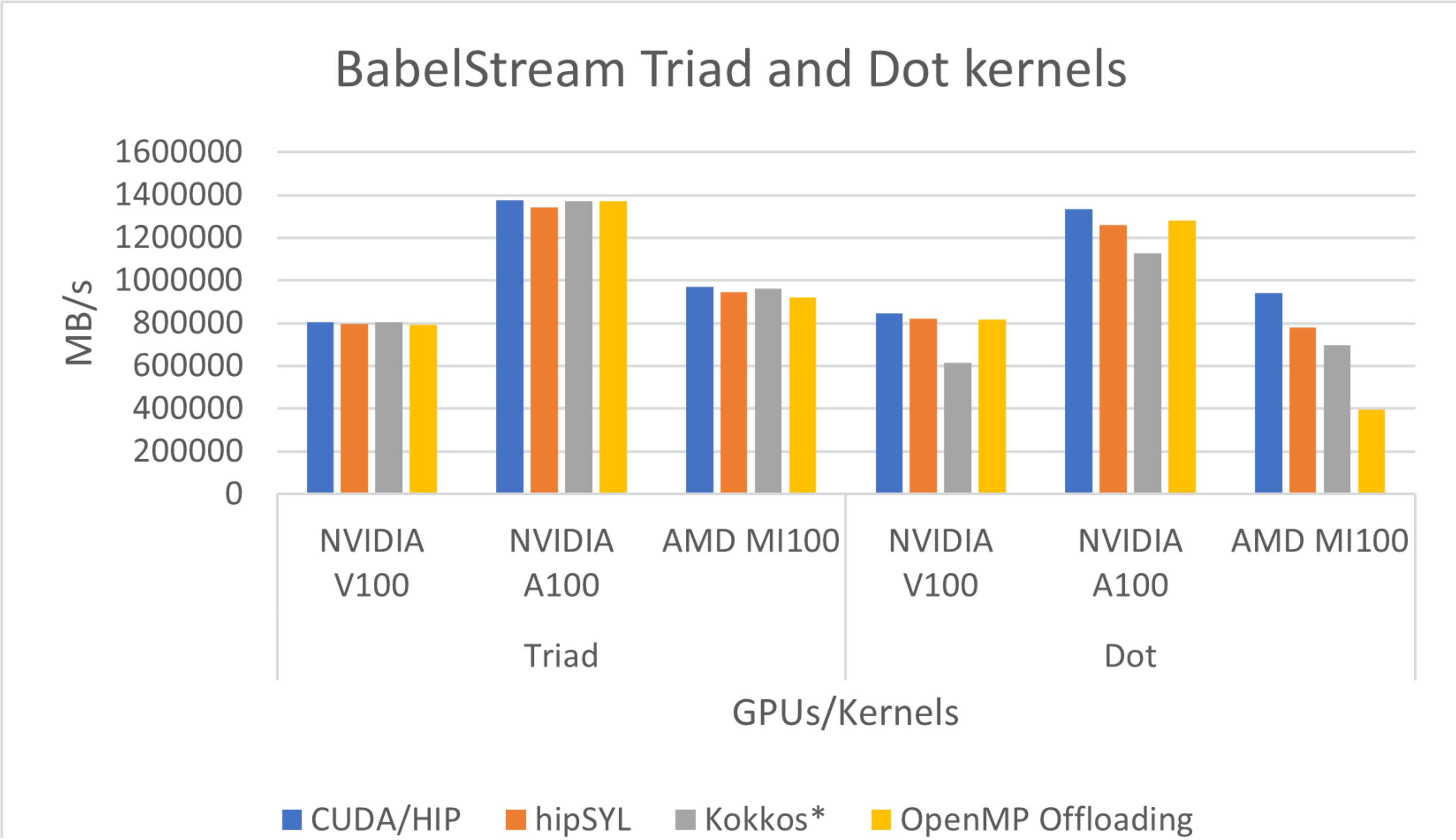
Kokkos

- **Kokkos** Core implements a programming model in C++ for writing performance portable applications targeting all major HPC platforms. It provides abstractions for both parallel execution of code and data management. (ECP/NNSA)
- Terminology: view, execution space (serial, threads, OpenMP, GPU,…), memory space (DRAM, NVRAM, …), pattern, policy
- Supports: CPU, AMD/NVIDIA GPUs, Intel KNL etc.
- https://github.com/kokkos

Alpaka

- Abstraction Library for Parallel Kernel Acceleration (**Alpaka)** library is a header-only C++14 abstraction library for accelerator development. Developed by HZDR.
- Similar to CUDA terminology, grid/block/thread plus element
- Platform decided at the compile time, single source interface
- Easy to port CUDA codes through CUPLA
- Terminology: queue (non/blocking), buffers, work division
- Supports: HIP, CUDA, TBB, OpenMP (CPU and GPU) etc.
- https://github.com/alpaka-group/alpaka

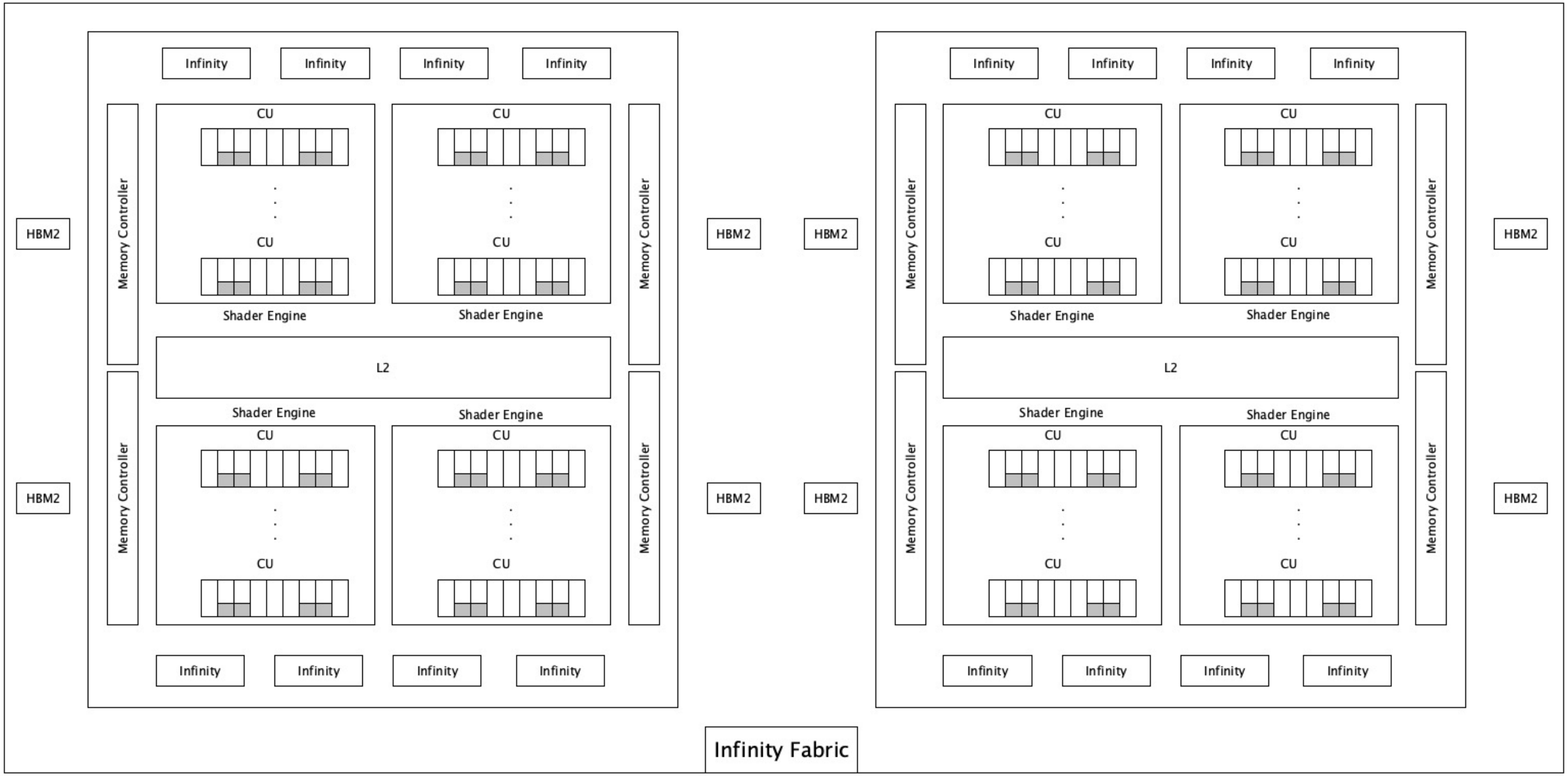
BabelStream Results



AMD Instinct MI250X

- Two graphics compute dies (GCDs)
- 64GB of HBM2e memory per GCD (total 128GB)
- 26.5 TFLOPS peak performance per GCD
- 1.6 TB/s memory bandwidth per GCD
- 110 CU per GCD, totally 220 CU per GPU
- Both GCDs are interconnected with 200 GB/s per direction
- The interconnection is attached on the GPU (not on the CPU)

MI250X



Infinity
Infinity
Infinity
Infinity
HBM2
Memory Controller
CU
CU
Shader Engine
CU
CU
Shader Engine
Memory Controller
HBM2
L2
Memory Controller
Shader Engine
CU
CU
Shader Engine
CU
CU
Memory Controller
HBM2
Infinity
Infinity
Infinity
Infinity
HBM2
Infinity
Infinity
Infinity
Infinity
Memory Controller
CU
CU
Shader Engine
CU
CU
Shader Engine
Memory Controller
HBM2
L2
Memory Controller
Shader Engine
CU
CU
Shader Engine
CU
CU
Memory Controller
HBM2
Infinity
Infinity
Infinity
Infinity
Infinity Fabric

Using MI250X

- Utilize CRAY MPICH with GPU Support (export MPICH_GPU_SUPPORT_ENABLED=1)
- Use 1 MPI process per GCD, so 2 MPI processes per GPU and 8 MPI processes per node, if you plan to utilize 4 GPUs
- MI250x can have multiple contexts sharing in the same GPU , thus supports many MPI processes per GPU by default
- Be careful with contention as multiple contexts share resources
- If the applications requires it, use different number of MPI processes

OpenACC

- GCC will provide OpenACC (Mentor Graphics contract, now called Siemens EDA). Checking functionality
- HPE is supporting OpenACC v2.6 for Fortran. This is quite old OpenACC version. HPE announced that they will **not** support OpenACC for C/C++
- 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 Offloading
- If the code is in Fortran, we could use GPUFort

Clacc

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

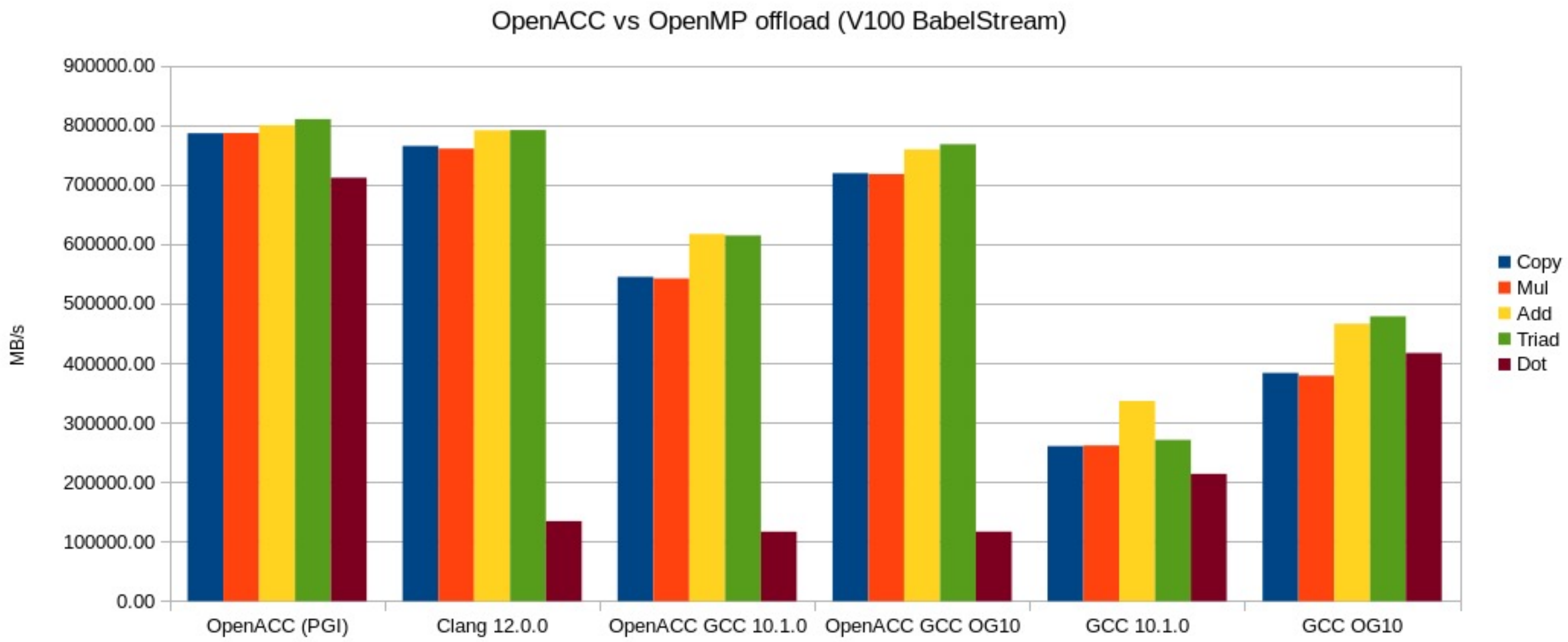
Original code:

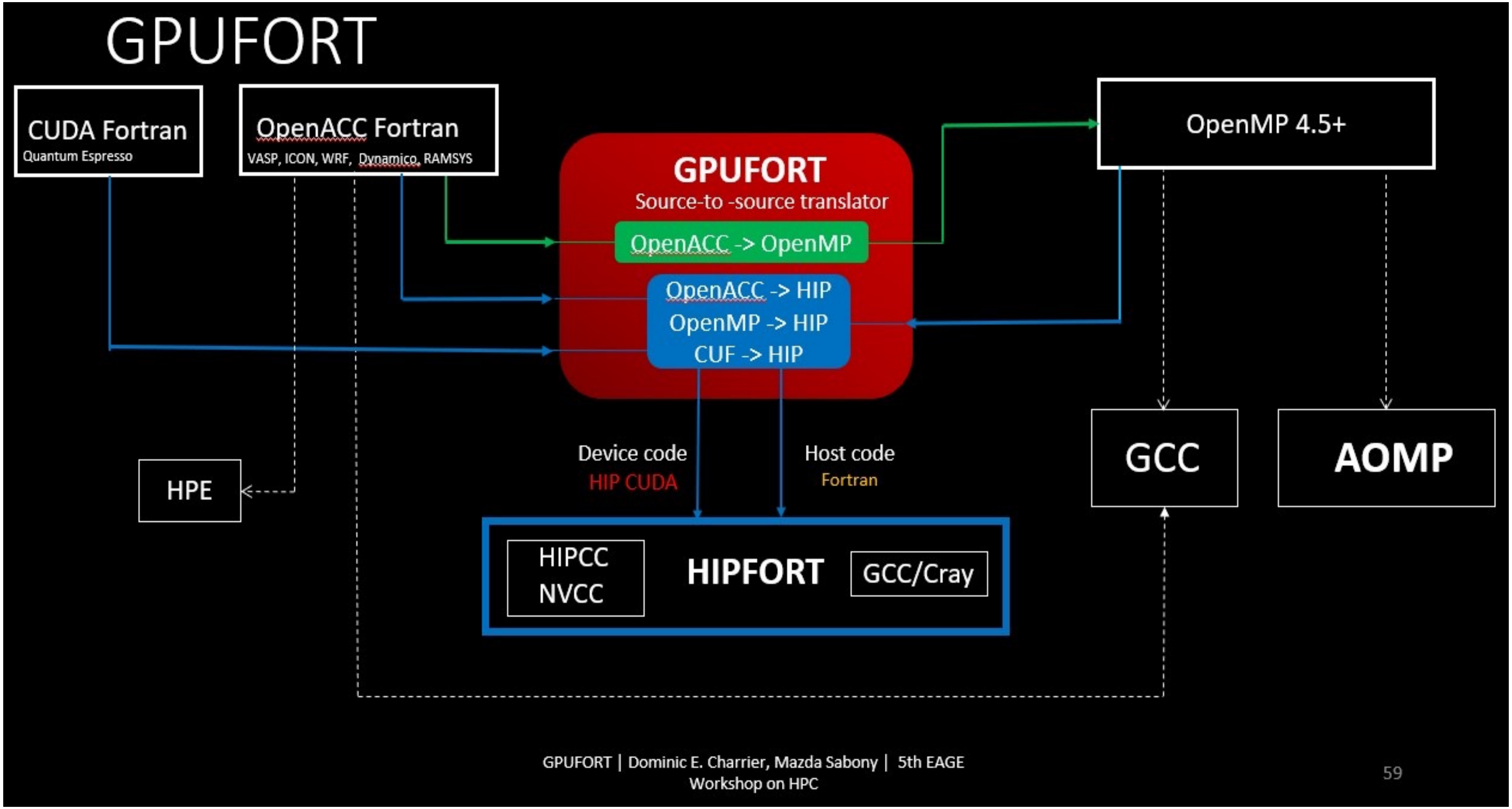
```
#pragma acc parallel loop reduction(max:lnorm) private(i,j) \
present(newarr, oldarr) collapse(2)
for (i = 1; i < nx + 1; i++) {
  for (j = 1; j < ny + 1; j++) {
```

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++) {
```

Results of BabelStream on NVIDIA V100




GPUFORT
CUDA Fortran
Quantum Espresso
OpenACC Fortran
VASP, ICON, WRF, Dynamico, RAMSYS
OpenMP 4.5+
GPUFORT
Source-to -source translator
OpenACC -> OpenMP
OpenACC -> HIP
OpenMP -> HIP
CUF -> HIP
Device code
HIP CUDA
Host code
Fortran
HPE
GCC
AOMP
HIPCC
NVCC
HIPFORT
GCC/Cray
GPUFORT | Dominic E. Charrier, Mazda Sabony | 5th EAGE Workshop on HPC
59

GPUFort – Fortran with OpenACC (1/2)

```
program saxpy

 implicit none
 integer, parameter :: N = 8192
 real :: y(N), x(N), a
 integer :: i
 a=2.0
 x(1)=5.0

 !$acc data copy(x(1:N),y(1:N))
 !$acc parallel loop
 do i = 1, N
  y(i) = a * x(i) + y(i)
 enddo
 !$acc end data

 print *, y(1)
end program
```

Ifdef original file

```
#ifdef __GPUFORT
 call gpufort_acc_enter_region()
 dev_x = gpufort_acc_copy(x(1:N))
 dev_y = gpufort_acc_copy(y(1:N))

 ! extracted to HIP C++ file
 call launch_axpy_12_b2e350_auto(0,c_null_ptr,dev_y,size(y,1),lbound(y,1),a,dev_x,size(x,1),lbound(x,1),n)
 call gpufort_acc_wait()
 call gpufort_acc_exit_region()
#else
!$acc data copy(x(1:N),y(1:N))

 !$acc parallel loop
 do i = 1, N
  y(i) = a * x(i) + y(i)
 enddo
 !$acc end data
#endif
```

GPUFort – Fortran with OpenACC (2/2)

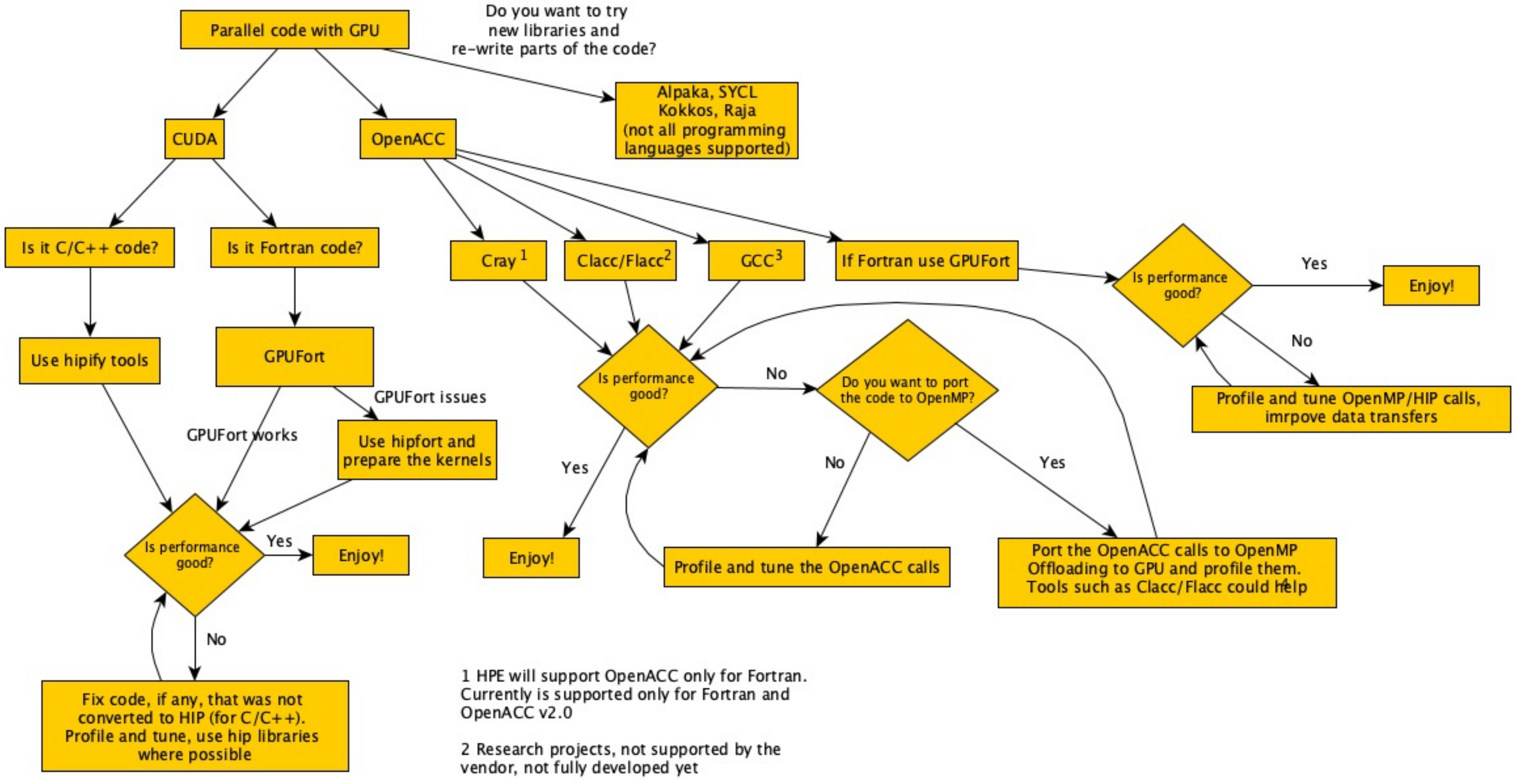
Extern C routine

```
extern "C" void launch_axpy_13_b2e350_auto(
   const int sharedmem,
   hipStream_t stream,
   float * __restrict__ y,
   const int y_n1,
   const int y_lb1,
   float a,
   float * __restrict__ x,
   const int x_n1,
   const int x_lb1,
   int n) {
  const int axpy_13_b2e350_blockX = 128;
  dim3 block(axpy_13_b2e350_blockX);
  const int axpy_13_b2e350_NX = (1 + ((n) - (1)));
  const int axpy_13_b2e350_gridX = divideAndRoundUp( axpy_13_b2e350_NX, axpy_13_b2e350_blockX );
  dim3 grid(axpy_13_b2e350_gridX);
// launch kernel
  hipLaunchKernelGGL((axpy_13_b2e350), grid, block, sharedmem, stream, y,y_n1,y_lb1,a,x,x_n1,x_lb1,n);
```

Kernel

```
__global__ void  axpy_13_b2e350(
   float * __restrict__ y,
   const int y_n1,
   const int y_lb1,
   float a,
   float * __restrict__ x,
   const int x_n1,
   const int x_lb1,
   int n) {
  #undef _idx_y
  #define _idx_y(a) ((a-(y_lb1)))
  #undef _idx_x
  #define _idx_x(a) ((a-(x_lb1)))
  int i = 1 + (1)*(threadIdx.x + blockIdx.x * blockDim.x);
  if (loop_cond(i,n,1)) {
   y[_idx_y(i)]=(a*x[_idx_x(i)]+y[_idx_y(i)]);
  }
}
```

Porting diagram and Software Roadmap



1 HPE will support OpenACC only for Fortran. Currently is supported only for Fortran and OpenACC v2.0

2 Research projects, not supported by the vendor, not fully developed yet

3 ORNL has a contract with Mentor Graphics to deliver GCC with OpenACC, not supported by the vendor

4 Depending on the programming language and if Clacc/Flacc can handle the supported calls

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 and other programming models
- 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
- Profiling, this could be a bit difficult without proper tools

Conclusion/Future work

- A code written in C/C++ and MPI+OpenMP is a bit easier to be ported to OpenMP offloading compared to other approaches.
- The hipSYCL, Kokos, and Alpaka could be a good option considering that the code is in C++.
- 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
- Track historical performance among new compilers
- GCC for OpenACC and OpenMP Offloading for AMD GPUs (issues will be solved with GCC 12.x and LLVM 13.x)
- Tracking how profiling tools work on AMD GPUs (rocprof, TAU, Score-P, HPCToolkit)
- Paper "Evaluating GPU programming models for the LUMI Supercomputer" will be presented at Supercomputing Asia 2022

LUMI

George Markomanolis

Lead HPC Scientist

CSC – IT Center for Science Ltd.

georgios.markomanolis@csc.fi

Follow us

Twitter: @LUMIhpc

LinkedIn: LUMI supercomputer

YouTube: LUMI supercomputer

www.lumi-supercomputer.eu

contact@lumi-supercomputer.eu

EuroHPC Joint Undertaking

The acquisition and operation of the EuroHPC supercomputer is funded jointly by the EuroHPC Joint Undertaking, through the European Union's Connecting Europe Facility and the Horizon 2020 research and innovation programme, as well as the of Participating States FI, BE, CH, CZ, DK, EE, IS, NO, PL, SE.

Leverage from the EU 2014–2020

European Union
European Regional Development Fund

REGIONAL COUNCIL OF KAINUU

EURO