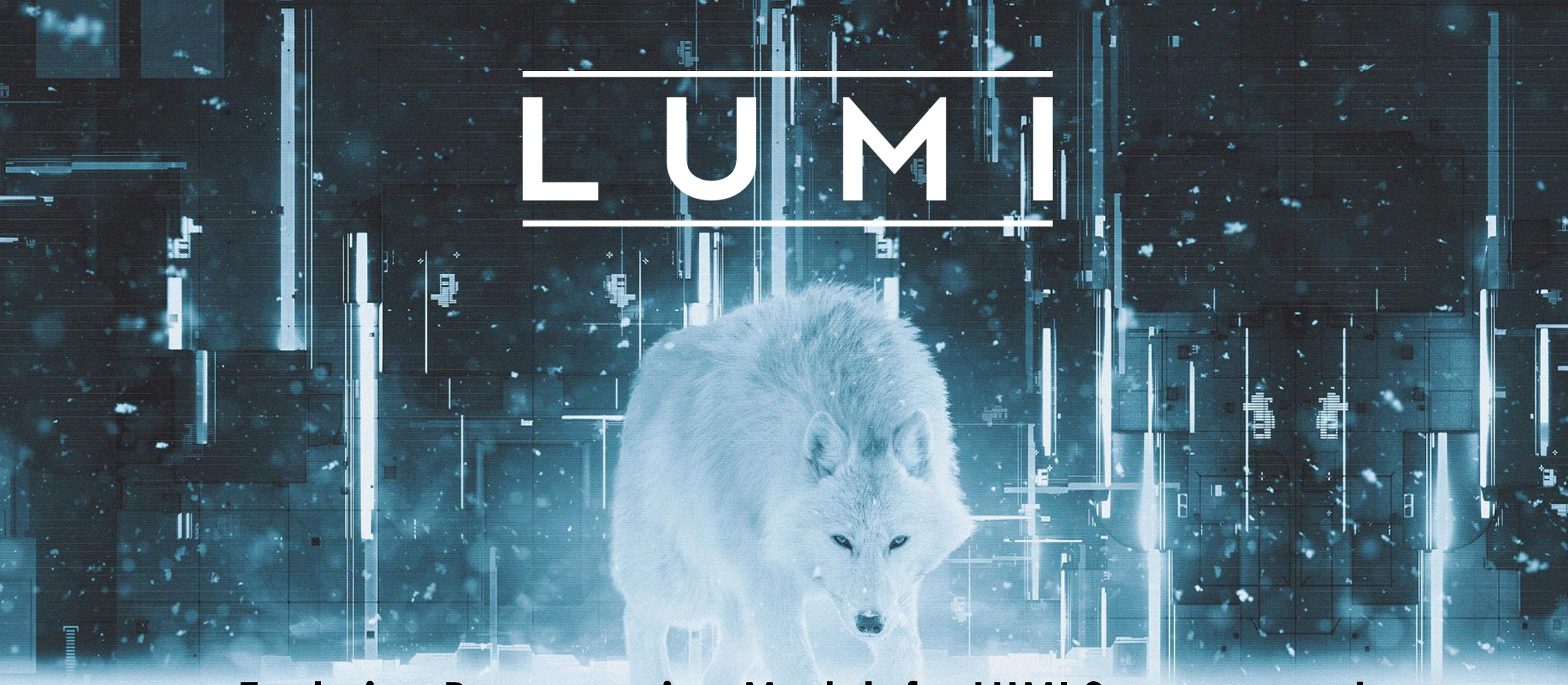
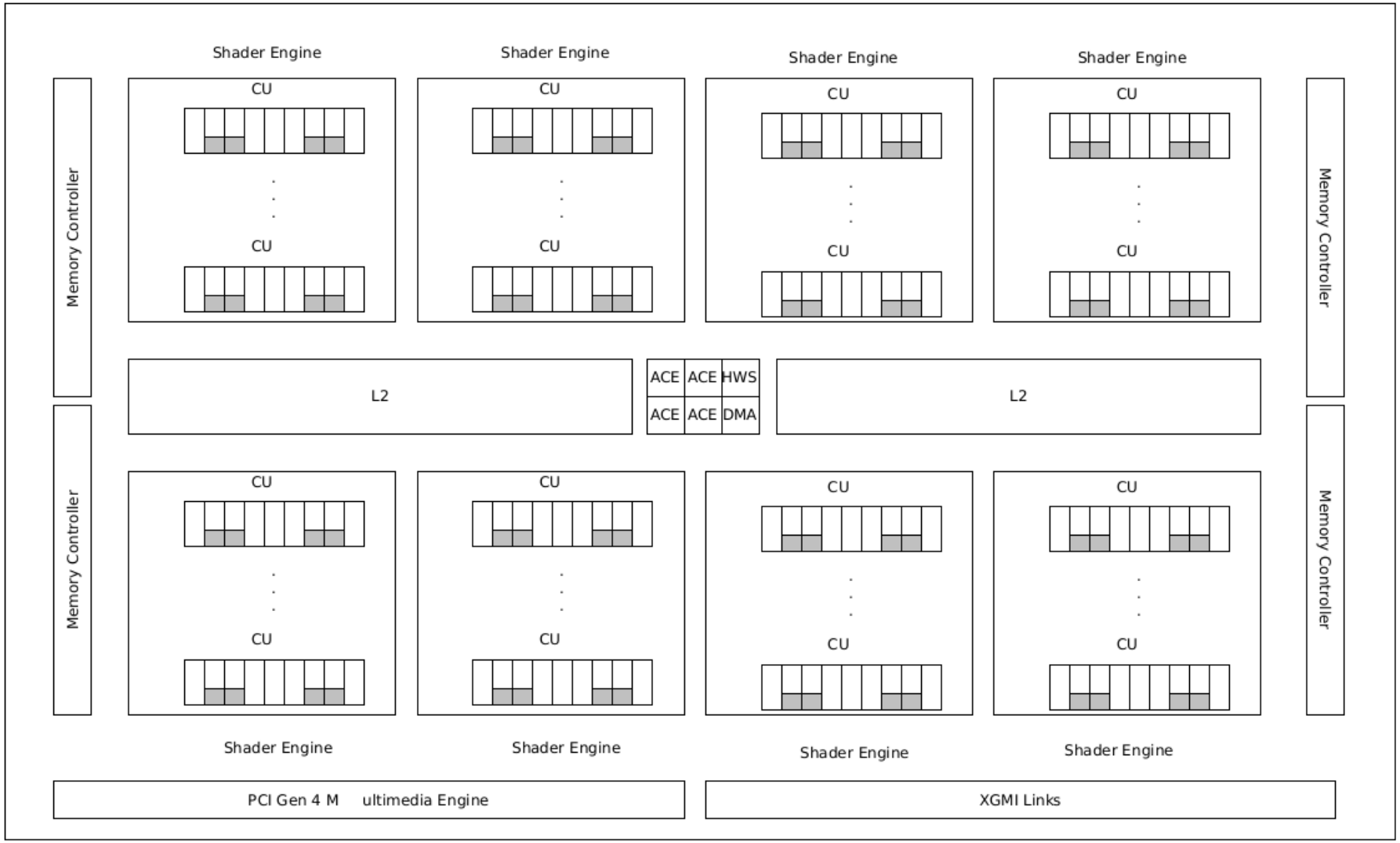


Exploring Programming Models for LUMI Supercomputer
ECMWF HPC Workshop on Meteorology, September 2021

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

AMD GPUs (MI100 example)



LUMI will have the next generation of AMD Instinct GPU

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

ECMWF Atlas

- Atlas is an ECMWF library for parallel data-structures supporting unstructured grids and function spaces

cd src
hipconvertinplace-perl.sh .

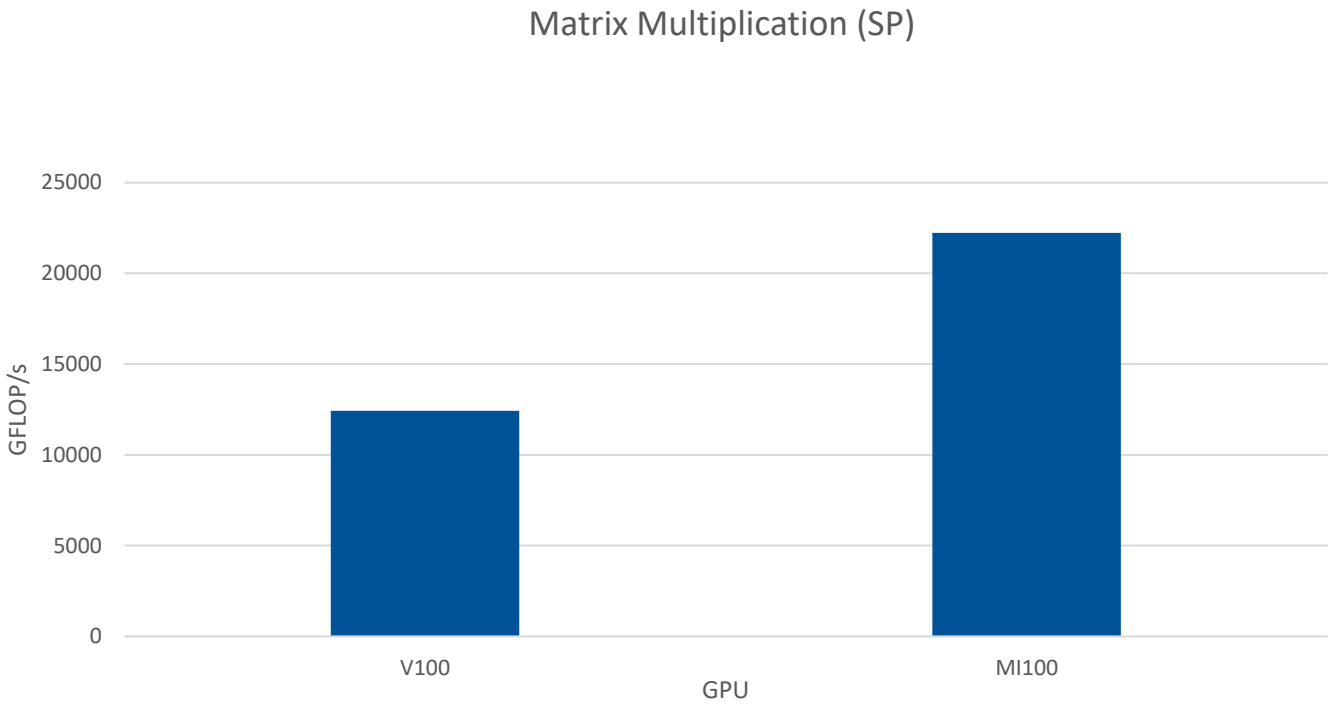
…
info: TOTAL-converted 92 CUDA->HIP refs (error:14 init:0 version:0 device:12 context:0 module:0 memory:17 virtual_memory:0 addressing:0 stream:0 event:0 external_resource_interop:0 stream_memory:0 execution:0 graph:0
…
kernels (5 total) : unpack_kernel(1) kernel_multiblock(1) kernel_ex(1) loop_kernel_ex(1) kernel_exe(1)

hipSuccess 15
hipDeviceSynchronize 12
hipLaunchKernelGGL 10
hipGetLastError 8
…
hipMemcpyDeviceToHost 1

- The code is hipified and now it is only C++ with HIP and Fortran with OpenACC.

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



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

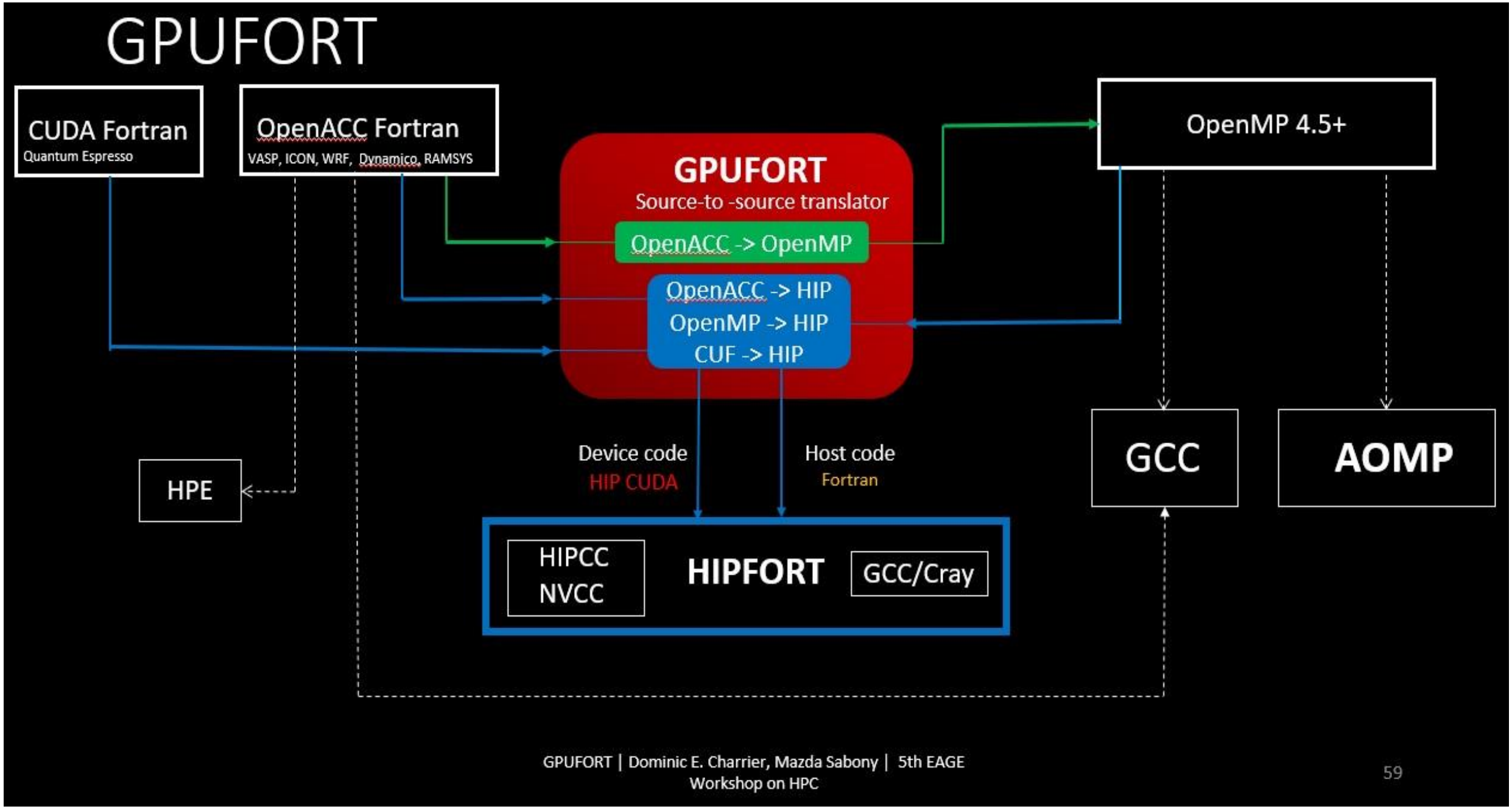
Fortran

- 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 hipify equivalent but there is another approach…
  - HIP functions are callable from C, using `extern C`
  - See hipfort

Hipfort

- 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 (see GPUFORT)
- Example of Fortran with HIP: https://github.com/cschpc/lumi/tree/main/hipfort
- Use OpenMP offload to GPUs

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


GPUFORT
CUDA Fortran
Quantum Espresso
OpenACC Fortran
VASP, ICON, WRF, Dynamico, RAMSYS
GPUFORT
Source-to -source translator
OpenACC -> OpenMP
OpenACC -> HIP
OpenMP -> HIP
CUF -> HIP
OpenMP 4.5+
HPE
Device code
HIP CUDA
Host code
Fortran
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)]);
 }
}
```

OpenACC

- GCC will provide OpenACC (Mentor Graphics contract, now called Siemens EDA). Checking functionality
- HPE is supporting for OpenACC v2.0 for Fortran. This is quite old OpenACC version. HPE announced support for OpenACC, newer versions for all the main programming languages (Fortran/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

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)

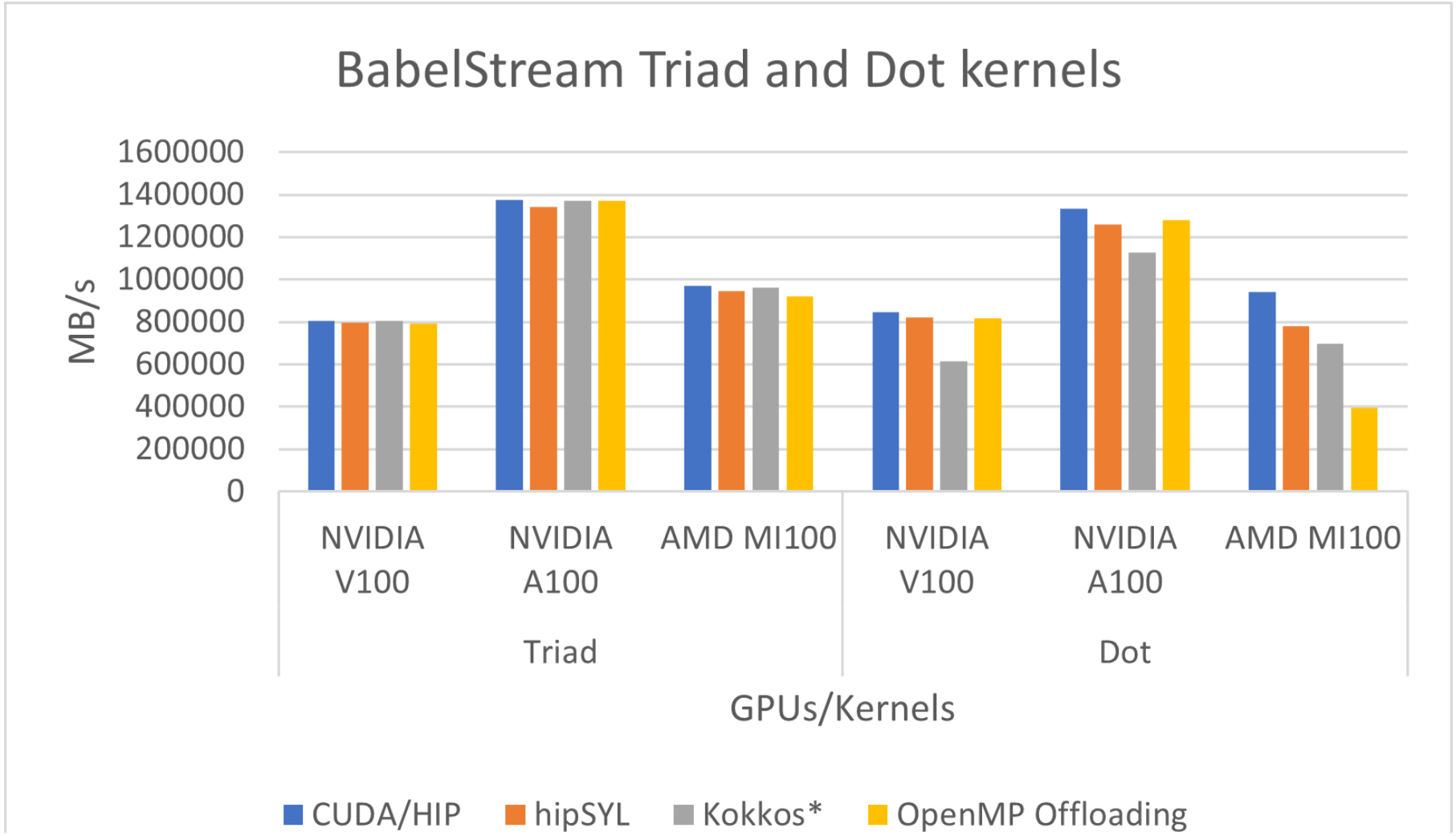
Kokkos

- **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.

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.

BabelStream Results



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 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, HPCToolkit)
- We have trained more than 80 people on HIP porting: http://github.com/csc-training/hip

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

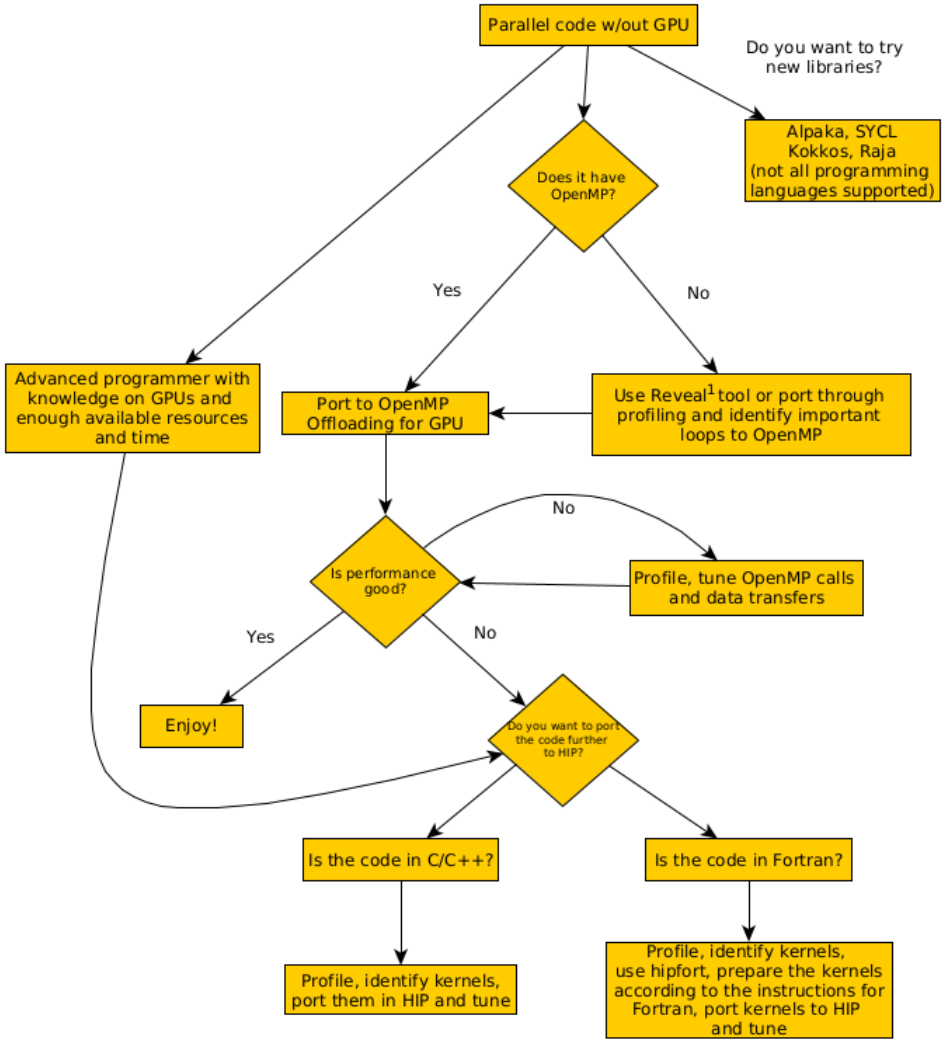
EuroHPC Joint Undertaking

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

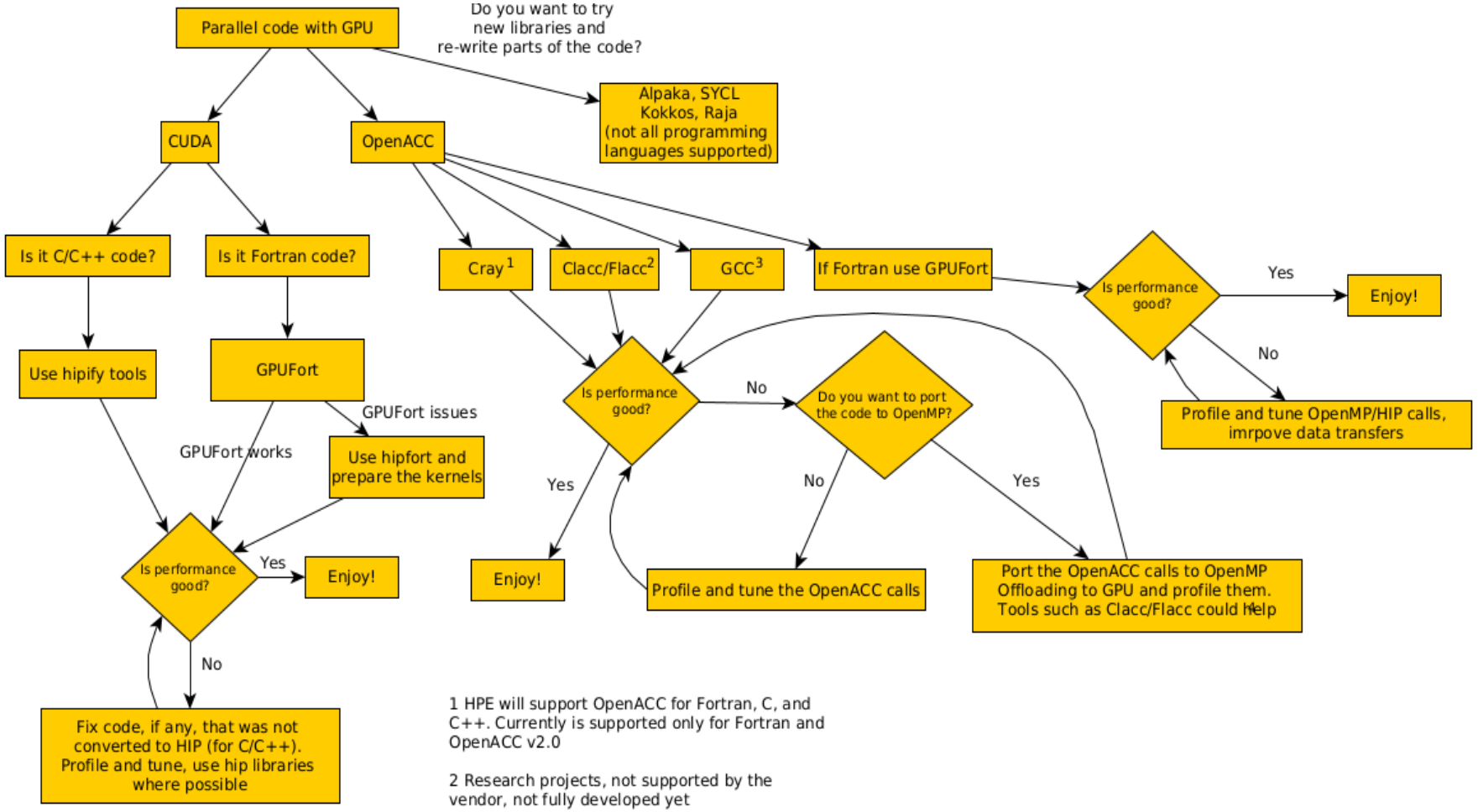
REGIONAL COUNCIL OF KAINUU

Porting Codes to LUMI (I)



1 Reveal will work good with Fortran codes and less with C, especially C++

Porting Codes to LUMI (II)



1 HPE will support OpenACC for Fortran, C, and C++. 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 all the calls

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