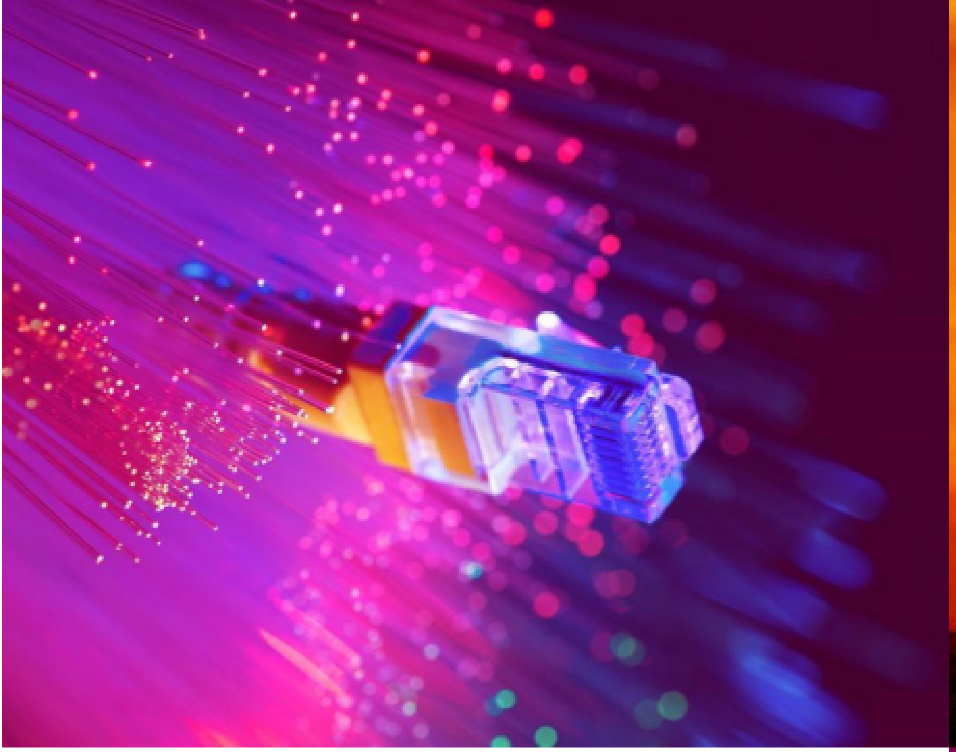
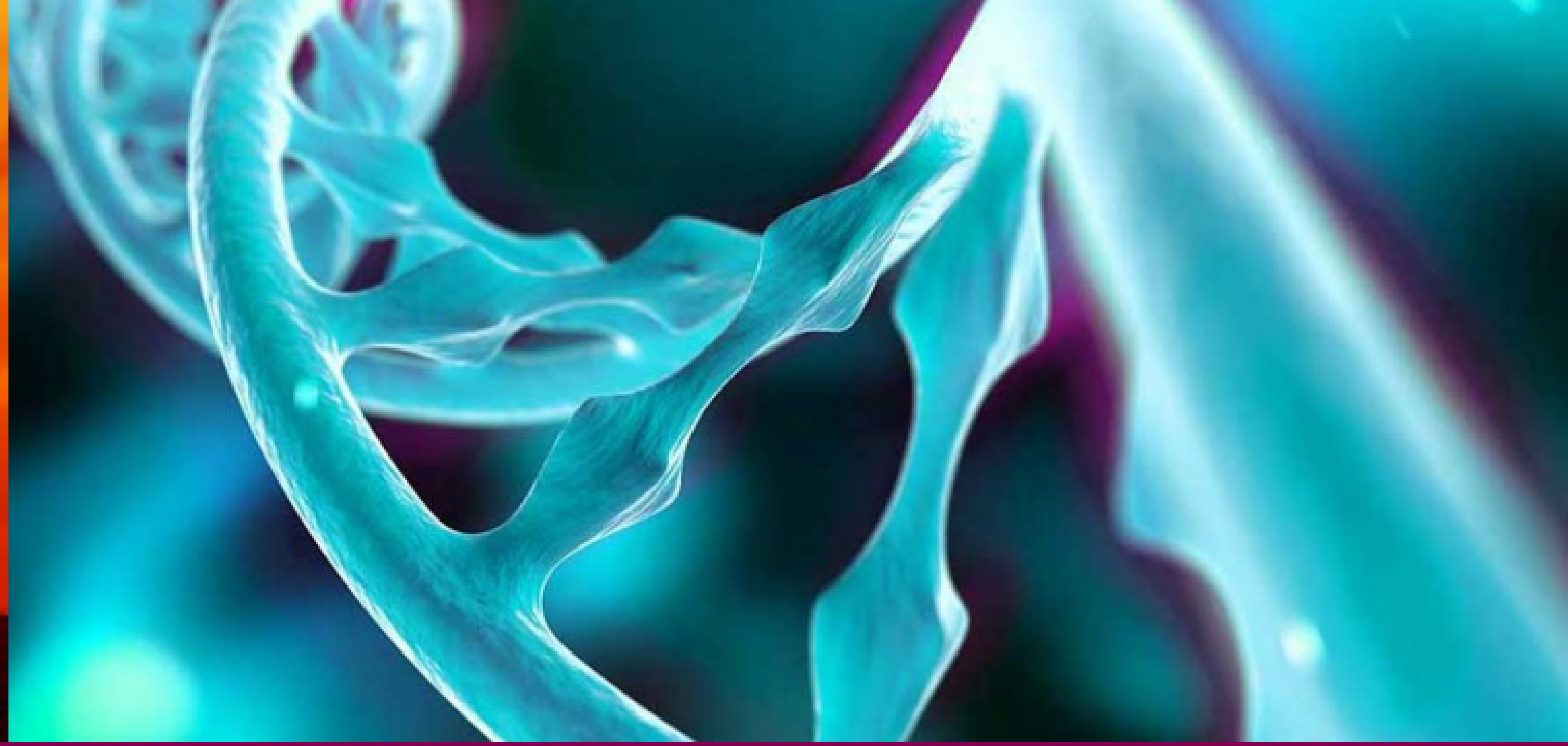




HIP101 Porting CUDA codes to HIP

CSC - IT Center for Science, 2021-02

Agenda (Times are in CET)

09:00 - 10:00 Introduction to AMD architecture and HIP

10:00 - 10:15 Break

10:15 - 10:45 Deep dive to Hipify tools and examples

10:45 - 11:30 Lunch

11:30 - 16:00 Hands-on sessions

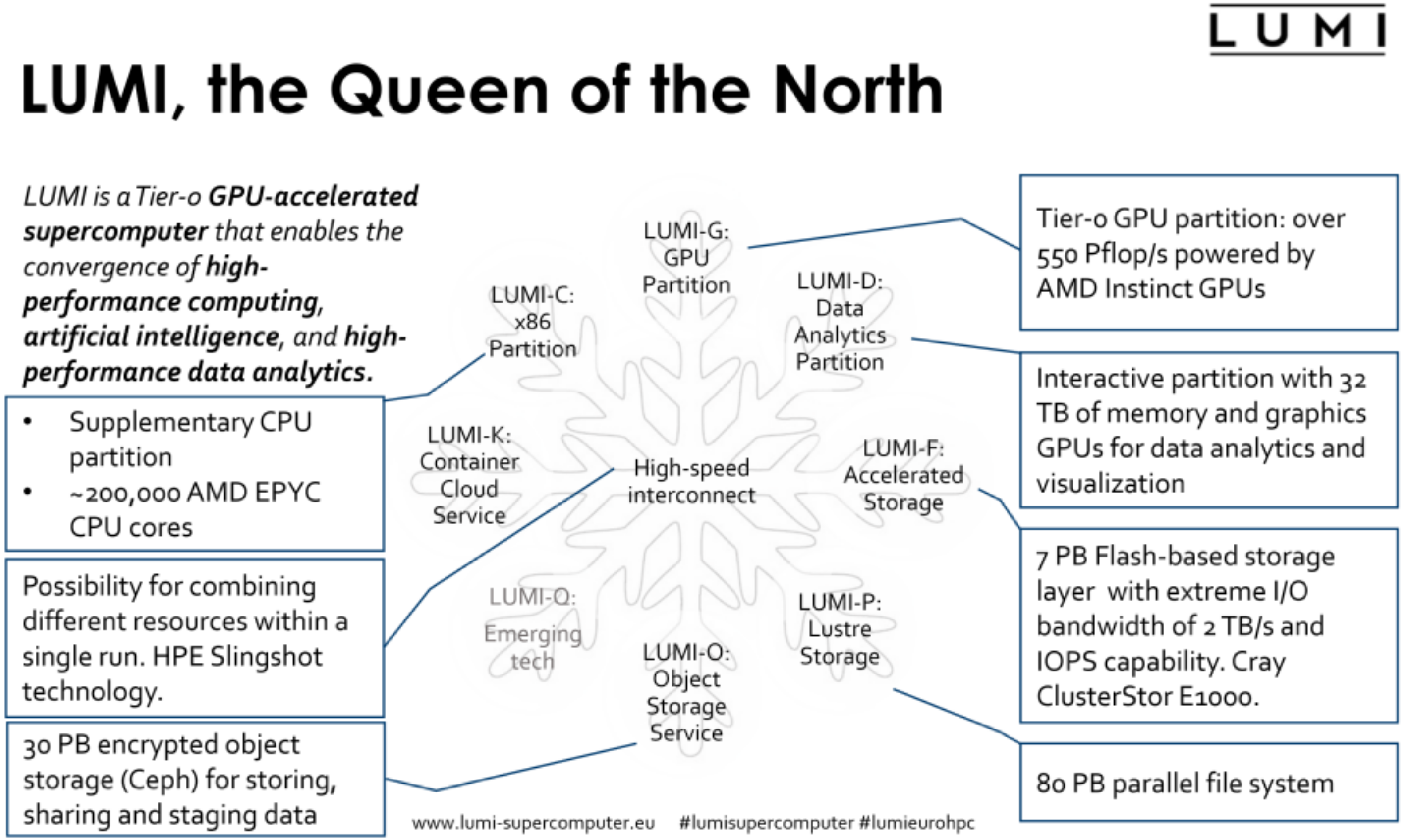
Disclaimer

- AMD ecosystem is under heavy development
- All the experiments took place on NVIDIA V100 GPU (Puhti supercomputer)

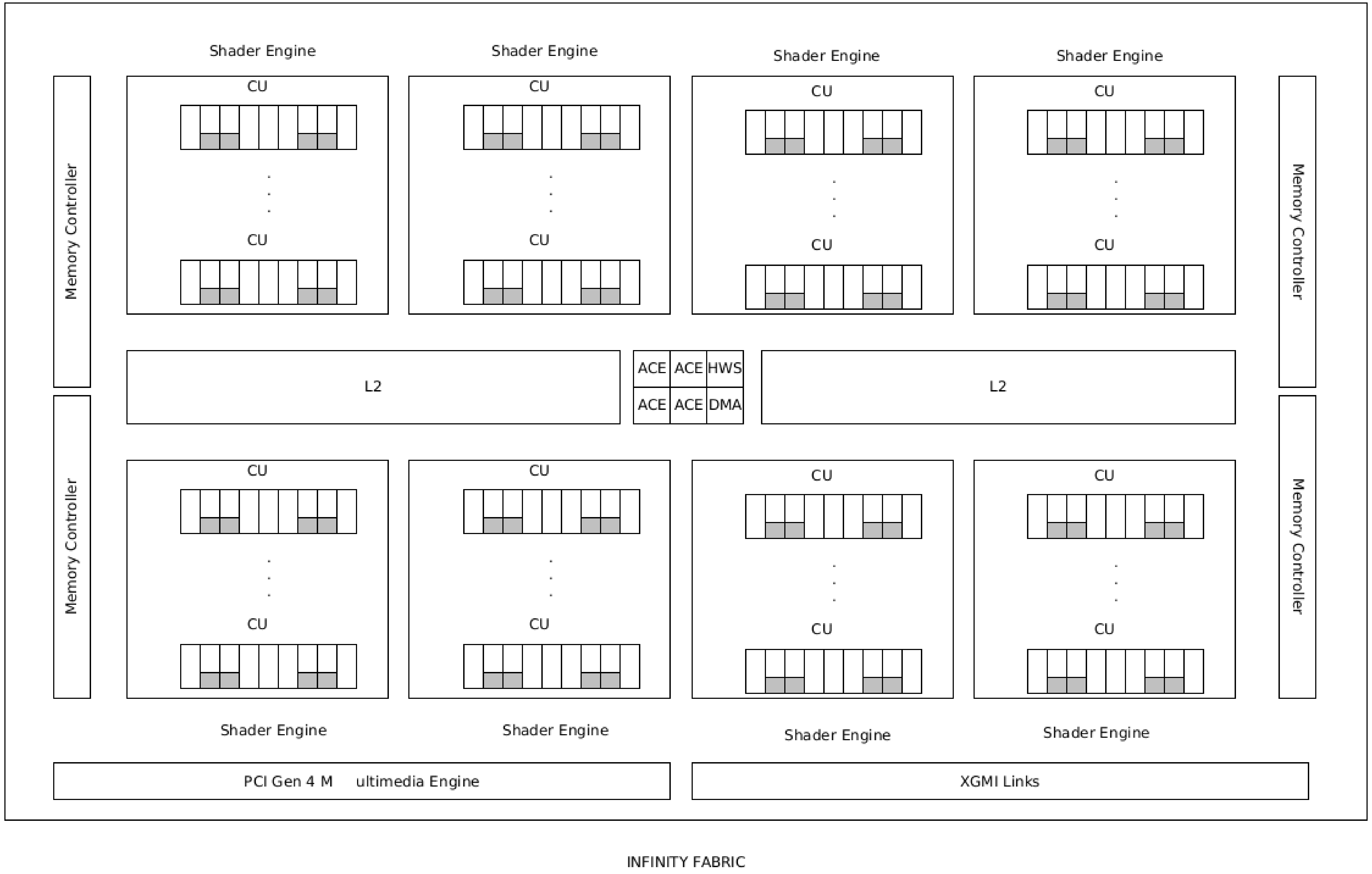
Motivation/Challenges

- LUMI will have AMD GPUs
- Need to learn how to port codes on AMD ecosystem
- Not yet access to AMD GPUs

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

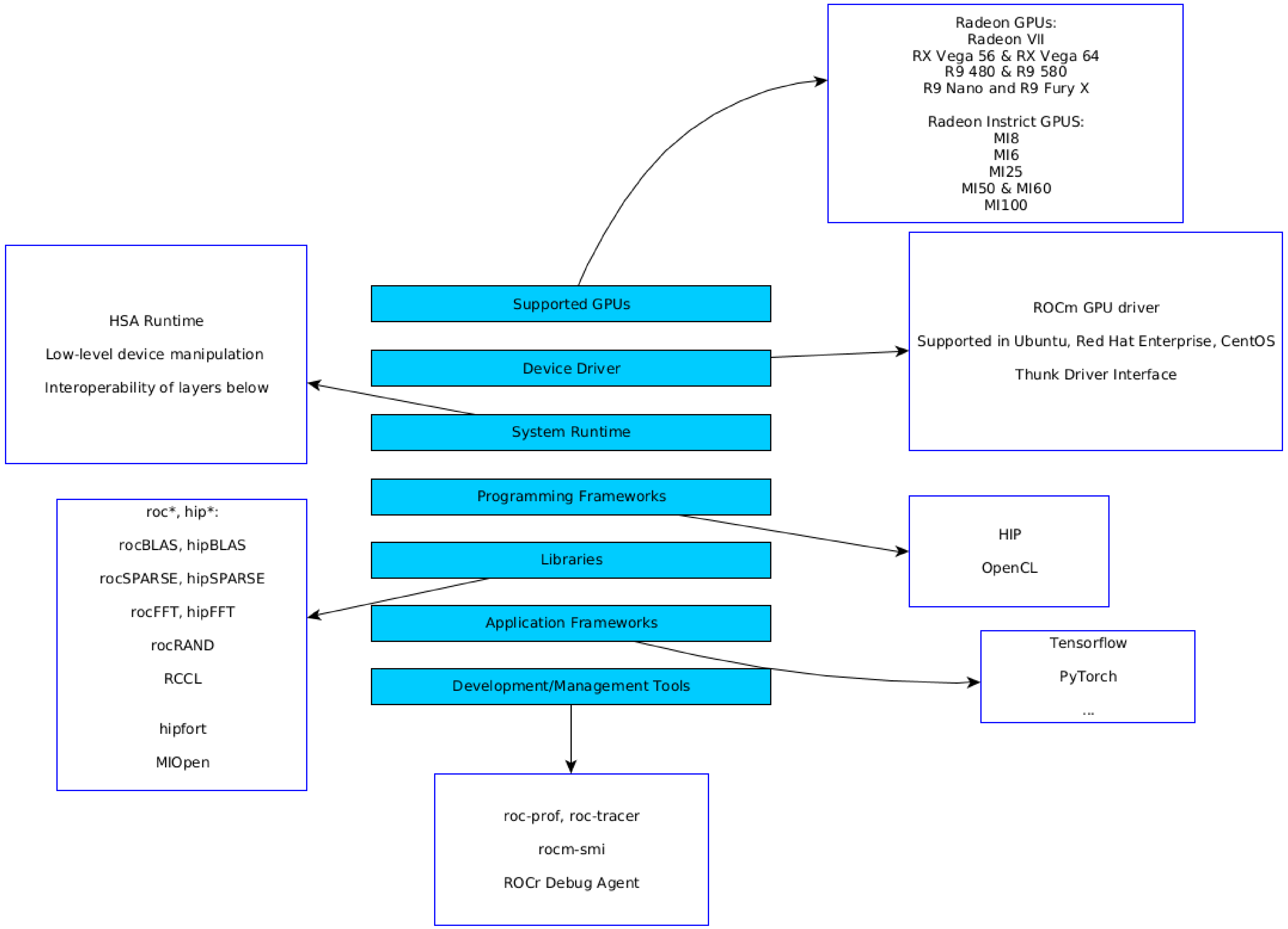
AMD GPUs (Mi100 example)



Differences between HIP and CUDA

- AMD GCN hardware wavefronts size is 64 (warp on CUDA is 32)
- Some CUDA library functions do not have AMD equivalents
- Shared memory and registers per thread can differ between AMD and NVIDIA hardware

ROCm



Radeon GPUs:
Radeon VII
RX Vega 56 & RX Vega 64
R9 480 & R9 580
R9 Nano and R9 Fury X
Radeon Instrict GPUS:
MI8
MI6
MI25
MI50 & MI60
MI100
HSA Runtime
Low-level device manipulation
Interoperability of layers below
Supported GPUs
Device Driver
System Runtime
Programming Frameworks
Libraries
Application Frameworks
Development/Management Tools
ROCm GPU driver
Supported in Ubuntu, Red Hat Enterprise, CentOS
Thunk Driver Interface
roc*, hip*:
rocBLAS, hipBLAS
rocSPARSE, hipSPARSE
rocFFT, hipFFT
rocRAND
RCCL
hipfort
MIOpen
HIP
OpenCL
Tensorflow
PyTorch
...
roc-prof, roc-tracer
rocm-smi
ROCr Debug Agent

ROCm Installation

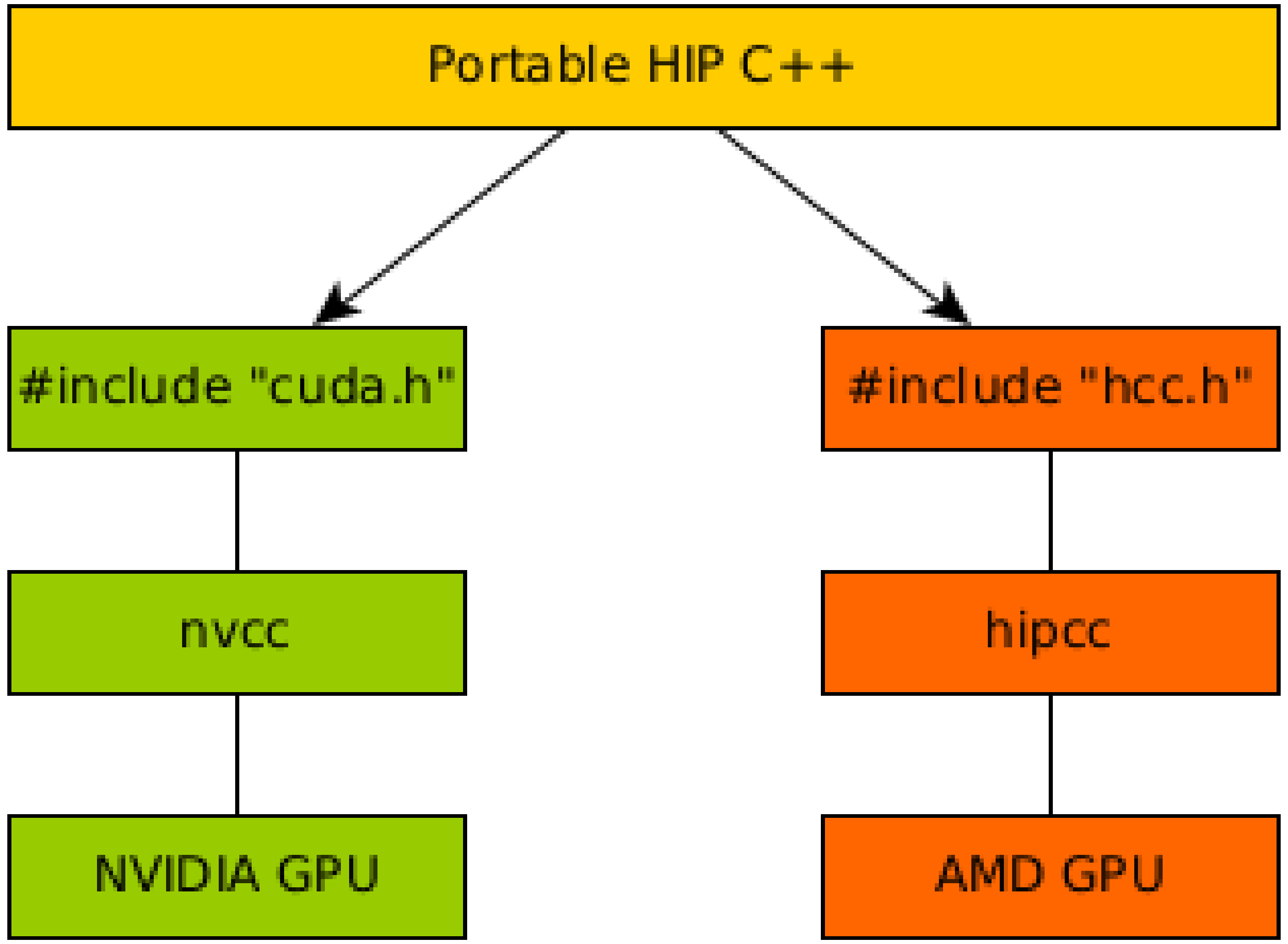
- Many components need to be installed
- Rocm-cmake
- HSA Runtime API
- ROCm LLVM/Clang
- ROCminfo (only for AMD HW)
- ROCM-Device-Libs
- ROCm-CompilerSupport
- ROCclr - Radeon Open Compute Common Language Runtime

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
- In some cases it is required to use AMD hardware for porting

HIP Portability

On a system with NVIDIA GPUs the hipcc, which is a compiler driver, will call the nvcc and not the hcc, as also a hip runtime will be included in the headers and it will be executed on NVIDIA GPU.



Differences between CUDA and HIP API

CUDA

```
#include "cuda.h"

cudaMalloc(&d_x, N*sizeof(double));

cudaMemcpy(d_x,x,N*sizeof(double),
                  cudaMemcpyHostToDevice);

cudaDeviceSynchronize();
```

HIP

```
#include "hip/hip_runtime.h"

hipMalloc(&d_x, N*sizeof(double));

hipMemcpy(d_x,x,N*sizeof(double),
                  hipMemcpyHostToDevice);

hipDeviceSynchronize();
```

Differences between CUDA and HIP Launch Kernels

CUDA

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

HIP

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

HIP Terminology

Term	Description
host	Executes the HIP API and can initiate kernel launches
default device	Each host maintains a default device.
active host thread	Thread running HIP API
HIP-Clang	Heterogeneous AMDGPU compiler
Hipify tools	Tools to convert CUDA code to HIP
hipconfig	Tool to report various configuration properties

HIP API

- Device management:
  - hipSetDevice(), hipGetDevice(), hipGetDeviceProperties()
- Memory Management:
  - hipMalloc(), hipMemcpy(), hipMemcpyAsync(), hipFree()
- Streams:
  - hipStreamCreate(), hipSynchronize(), hipStreamSynchronize(), hipStreamFree()
- Events:
  - hipEventCreate(), hipEventRecord(), hipStreamWaitEvent(), hipEventElapsedTime()
- Device Kernels:
  - __global__, __device__, hipLaunchKernelGGL()
- Device code:
  - threadIdx, blockIdx, blockDim, __shared__
  - Hundreds math functions covering entire CUDA math library
- Error handling:
  - hipGetLastError(), hipGetErrorString()

HIP Host memory visibility

HIP API	Synchronization effect	Fence	Coherent Host Memory Visibility	Non-coherent Host Memory Visibility
hipStreamSynchronize	host waits for all commands in the specified stream	system-scope release	yes	yes
hipDeviceSynchronize	host waits for all commands across all streams of the device	system-scope release	yes	yes

HIP Host memory visibility (cont.)

HIP API	Synchronization effect	Fence	Coherent Host Memory Visibility	Non-coherent Host Memory Visibility
hipEventSynchronize	host waits for specified event to complete	device-scope release	yes	depends
hipStreamWaitEvent	stream waits for specified event	none	yes	no

Libraries

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

Hipify Tools

- Hipify tools convert automatically CUDA codes
- It is possible that not all the code is converted, the remaining needs the implementaiton of the developer
- Hipify-perl: text-based search and replace
- Hipify-clang: source-to-source translator that uses clang compiler

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.)

```
$ ls src/
Makefile.am matMulAB.c matMulAB.h matMul.c
$ hipconvertinplace-perl.sh src
$ ls src/
Makefile.am matMulAB.c matMulAB.c.prehip matMulAB.h matMul.c matMul.c.prehip
```

No compilation took place, just conversion.

Hipify-perl (cont.)

The hipify-perl will return a report for each file with the option `--print-stats`, 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
```

Hipify-perl (cont.)

CUDA

```
#include <cuda_runtime.h>
#include "cublas_v2.h"

 if (cudaSuccess != cudaMalloc((void **) &a_dev,
sizeof(*a) * n * n) ||
cudaSuccess != cudaMalloc((void **) &b_dev,
sizeof(*b) * n * n) ||
cudaSuccess != cudaMalloc((void **) &c_dev,
sizeof(*c) * n * n)) {
printf("error: memory allocation (CUDA)\n");
cudaFree(a_dev); cudaFree(b_dev);
cudaFree(c_dev);
      cudaDestroy(handle);
      exit(EXIT_FAILURE);
    }
```

HIP

```
#include <hip/hip_runtime.h>
#include "hipblas.h”

if (hipSuccess != hipMalloc((void **) &a_dev, sizeof(*a)
* n * n) ||
        hipSuccess != hipMalloc((void **) &b_dev,
sizeof(*b) * n * n) ||
        hipSuccess != hipMalloc((void **) &c_dev,
sizeof(*c) * n * n)) {
      printf("error: memory allocation (CUDA)\n");

      hipFree(a_dev); hipFree(b_dev); hipFree(c_dev);
      hipblasDestroy(handle);
      exit(EXIT_FAILURE);
    }
```

Hipify-perl (cont.)

CUDA

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

HIP

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

Compilation

1. Compilation with **CC=hipcc**

matMulAB.c:21:10: fatal error: hipblas.h: No such file or directory 21 | #include "hipblas.h"

2. Install HipBLAS library
3. Compile again and the binary is ready. When the HIP is on NVIDIA hardware with extension **.cpp**, then use the option "--x cu" after hipcc

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 (when bug* is fixed) 96.5217
  - Conversion Success (when bug* is fixed) 99.2699

* https://github.com/ROCm-Developer-Tools/HIPIFY/issues/246

Hipify-clang

- Build from source
- Some times needs to include manually the headers -I/...

```
$ hipify-clang --print-stats -o matMul.o matMul.c
[HIPIFY] info: file 'matMul.c' statistics:
CONVERTED refs count: 0
UNCONVERTED refs count: 0
CONVERSION %: 0
REPLACED bytes: 0
TOTAL bytes: 4662
CHANGED lines of code: 1
TOTAL lines of code: 155
CODE CHANGED (in bytes) %: 0
CODE CHANGED (in lines) %: 1
20 TIME ELAPSED s: 22.94
```

Benchmark MatMul OpenMP oflload

- Use the benchmark https://github.com/pc2/OMP-Offloading for testing purposes, matrix multiplication of 2048 x 2048
- CUDA

```
matMulAB (11) : 1001.2 GFLOPS 11990.1 GFLOPS maxabserr = 0.0
```

- HIP

```
matMulAB (11) : 978.8 GFLOPS 12302.4 GFLOPS maxabserr = 0.0
```

- For the most executions, HIP version was equal or a bit better than CUDA version, for total 21 execution, there is ~2.23% overhead for HIP using NVIDIA GPUs

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
- HIP calls: hipMemcpy, hipMalloc, hipMemcpyHostToDevice, hipMemcpyDeviceToHost, hipLaunchKernelGGL, hipDeviceSynchronize, hip_runtime…
- 32768 number of small particles, 2000 time steps
- CUDA execution time: 68.5 seconds
- HIP execution time: 70.1 seconds, ~2.33% overhead

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 C function
    4. Things could change
- Use OpenMP offload to GPUs

Fortran SAXPY example

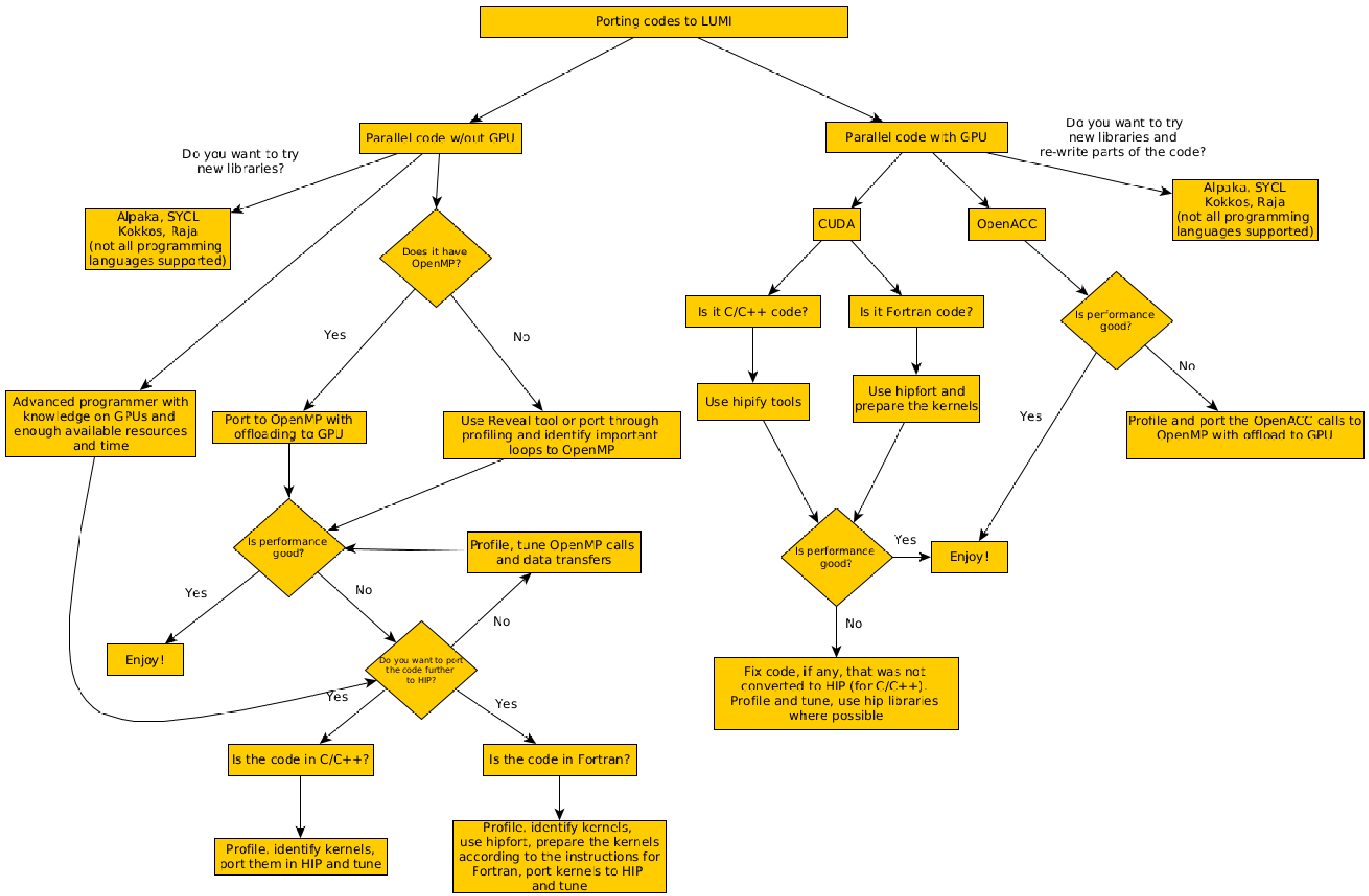
- Fortran CUDA, 29 lines of code
- Ported to HIP manually, two files of 52 lines, with more than 20 new lines.
- Quite a lot of changes for such a small code.
- Should we try to use OpenMP offload before we try to HIP the code?
- Need to adjust Makefile to compile the multiple files
- Example of Fortran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

OpenMP or HIP

Some users will be questioning about the approach

- OpenMP can provide a quick porting but it is expected with HIP to have better performance as we avoid some layers like that.
- For complicated codes and programming languages as Fortran, probably OpenMP could provide a benefit. Always profile your code to investigate the performance.

Porting codes to LUMI



Profiling/Debugging

- AMD will provide APIs for profiling and debugging
- Cray will support the profiling API through CrayPat
- Some well known tools are collaborating with AMD and preparing their tools 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));
```

Programming Models

- OpenACC will be probably available through the GCC as Mentor Graphics (now called Siemens EDA) is developing the OpenACC integration
- Kokkos, Raja, Alpaka, and SYCL should be able to be used on LUMI but they do not support all the programming languages

Hipify Gromacs

- GROMACS is a well-known molecular dynamics package
- This is an effort to hipify and not to execute on an optimized environment
- This showcases the procedure for a large application and possible issues
- No effort to optimize the code

Hands-on Demonstration

- View instructions:HackMD
- Ask question (at the end of the document below the feedback title): HackMD