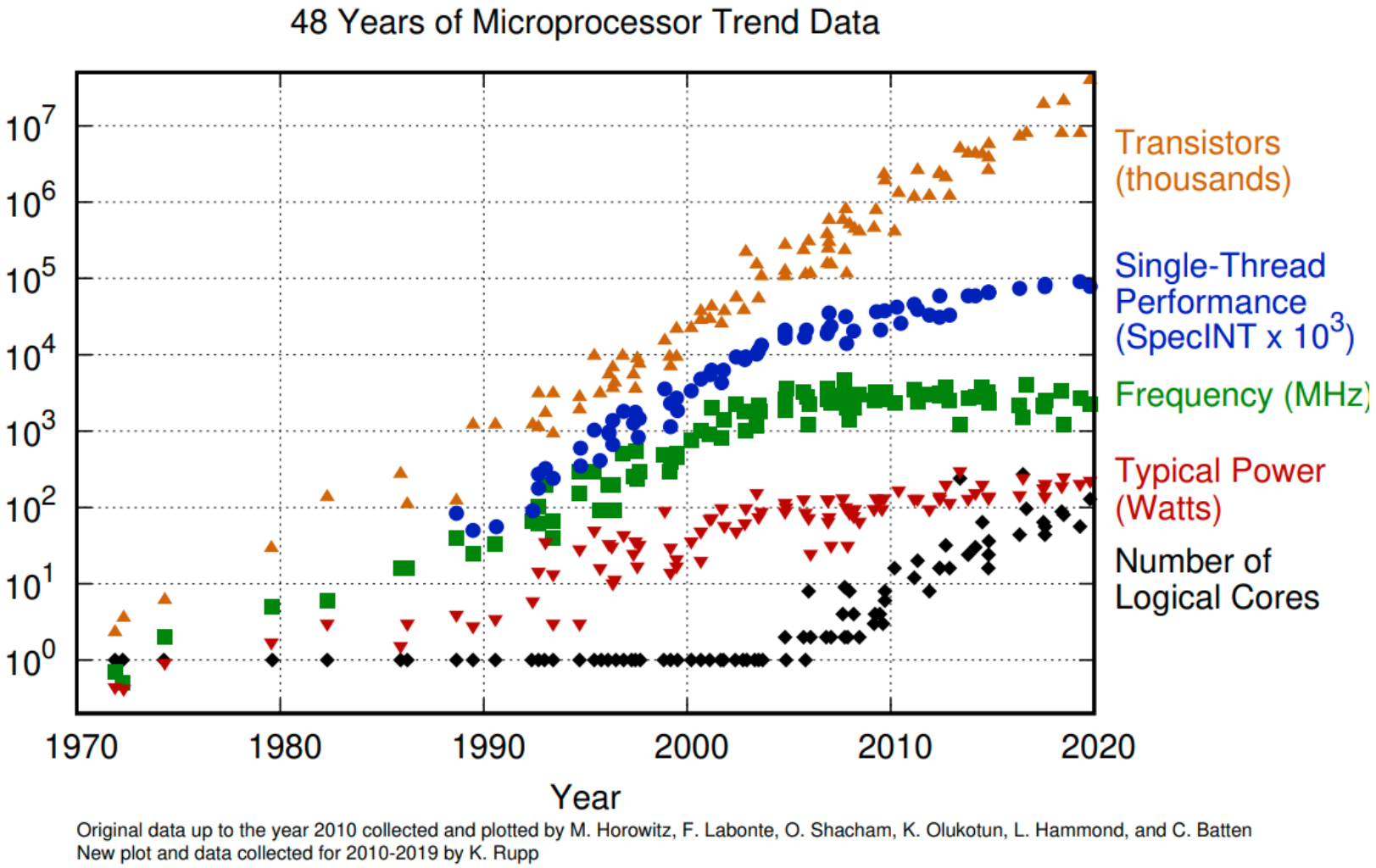
Introduction to GPU course

Jonathan Vincent

Outline

- Why GPUs
- What problems fit on GPUs
- GPU hardware and software Ecosystem
- GPU programming concepts

Why GPUs

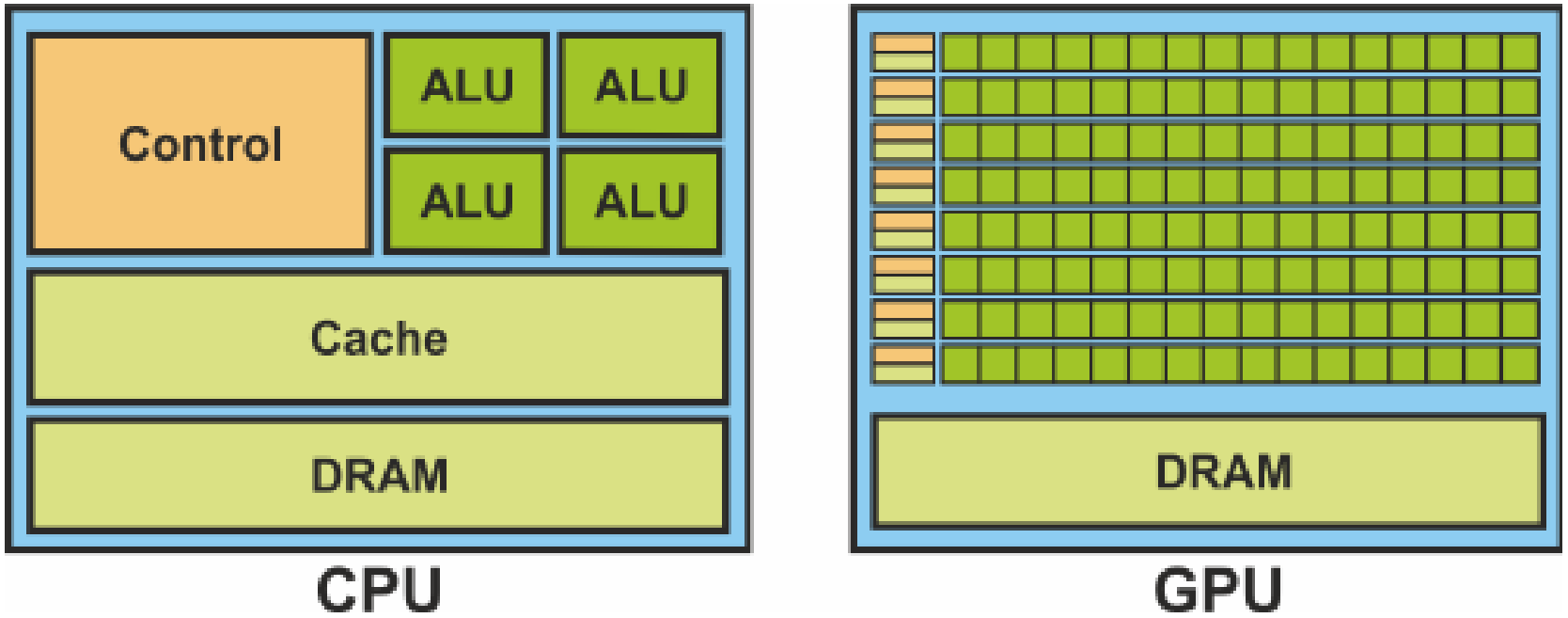


Historically CPU performance increases have been due to increasing transistor count and reduction of transistor size

CPUs single thread performance has peaked, because of power requirements of this method.

More recently total system performance is increased by increasing the number of cores, requiring parallel programming

Why GPUs



- GPUs are massively parallel
- Much more of the available space used for compute
- Much lower cache and control logic
  - Memory access can be very slow, but memory bandwidth very high
  - No out of order execution
  - SINGLE THREADED PERFORMANCE POOR ON GPU

Why GPUs

- GPUs are massively parallel and have very high peak performance
  - Much more of available space given to compute
- Improved energy efficiency
  - GPUs have much higher calculations per watt than CPUs
- Memory performance largest single factor in performance
  - GPUs have very high memory bandwidth
  - Memory latency very high
  - GPUs are very efficient at swapping threads to hide latency
    - *Requires many threads to hide latency effectively*

What problems fit onto GPUs

- GPU Advantages
  - Very high Floating-Point Performance (FLOPS)
  - Very high memory bandwidth
- GPU Disadvantages
  - Poor single thread performance
  - Less memory overall than CPU (in general)
- To effectively run on GPUs we need
  - High parallelism
  - Low enough memory requirement to fit in GPU memory
    - *Dardel GPU nodes have*
      - 512 GB memory on CPU
      - 128 GB memory per Mi250x (512 GB Total)
  - High arithmetic intensity

Arithmetic Intensity

- Arithmetic Intensity is the number of floating point operations per memory access
- Examples
  - C[i] = A[i] + B[i]
    - *Low Arithmetic intensity, one FLOP per three memory operations*
  - Matrix multiplication
    - *High intensity, 2N/3 operations per memory access for NxN matrices.*
  - FFT
    - *5 logN/3 operations per memory access*

GPU hardware and software ecosystem

- CUDA Software
  - Compilers and libraries
- HIP software
  - Compilers and libraries
- GPU systems available to Swedish researches
  - Dardel
  - Lumi
  - Alvis (Specifically for AI research)

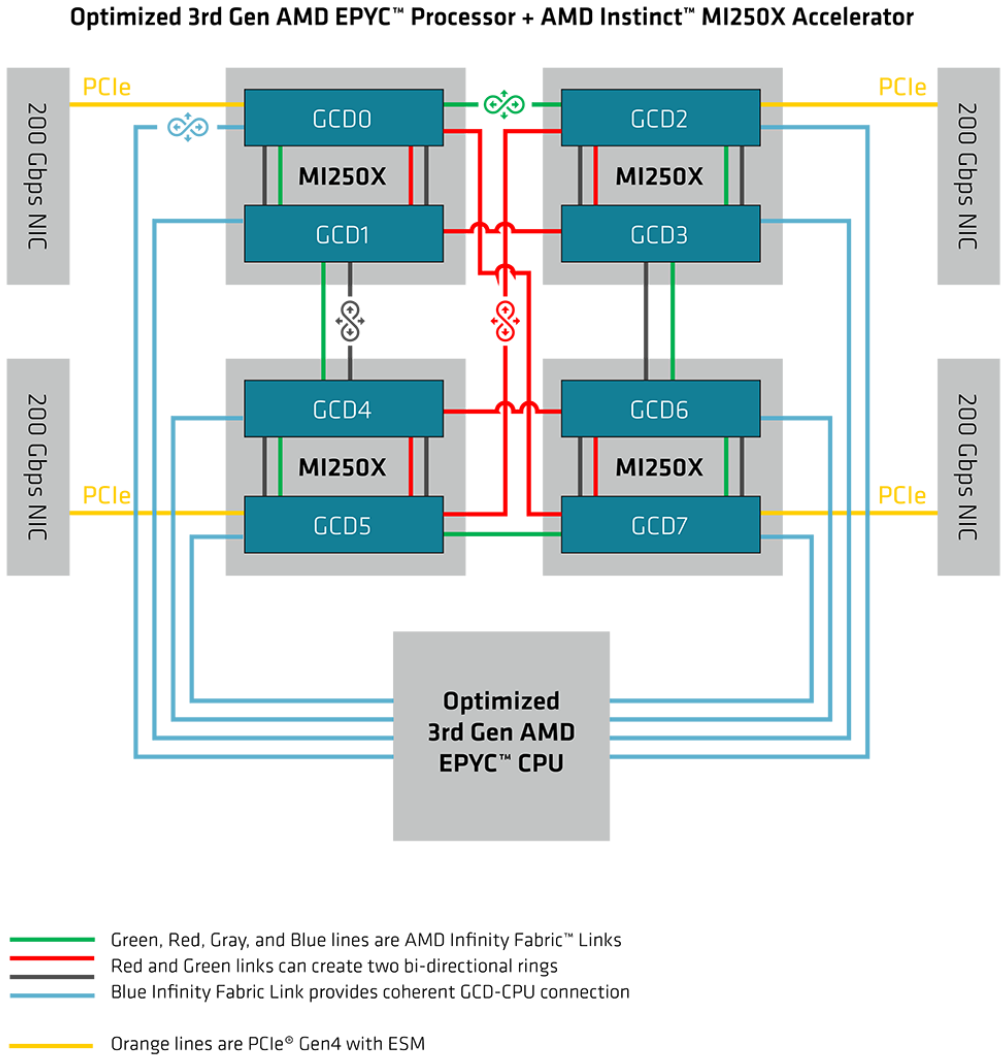
Compilers and Libraries

- AMD
  - Hipcc
  - Blas
- Nvidia
  - Nvcc
  - nvfort

Tools

- AMD
  - Rocprof
  - Omnitrace
- Nvidia
  - Nsight Compute
  - Nsight Systems

Dardel GPU nodes



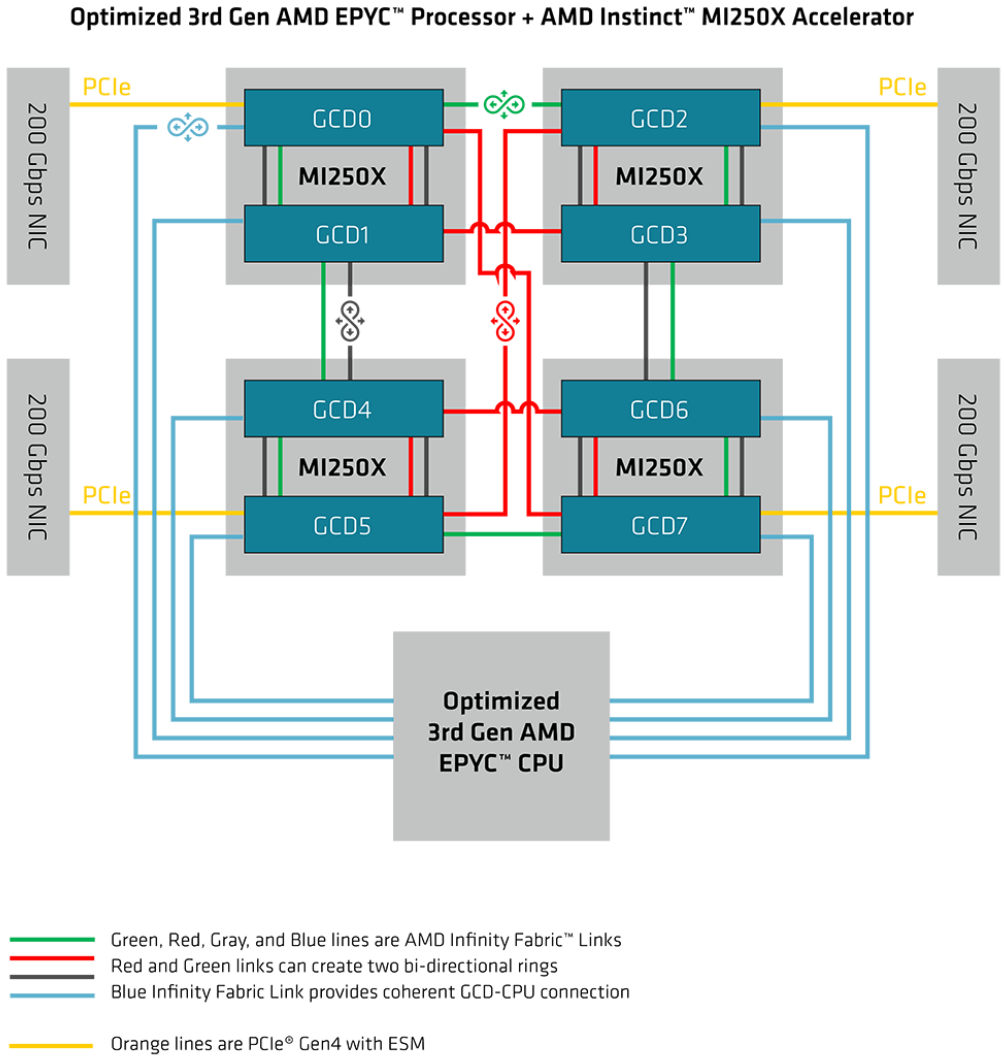
Dardel has 56 GPU nodes

Each GPU node has

- One AMD EPYC™ processor with 64 cores
- Four AMD Instinct™ MI250X GPUs connected by AMD Infinity Fabric™ Links
  - Each MI250x has two GCDs (Graphics Compute Die), so 8 GCDs per node
- 4 Network cards (one connected to each MI250x)

Total MI250x FP performance 47.9 TFLOPS

LUMI GPU nodes



Lumi-G has 2560 GPU nodes

Each GPU node is the same as Dardel with

- One AMD EPYC™ processor with 64 cores
- Four AMD Instinct™ MI250X GPUs connected by AMD Infinity Fabric™ Links
  - Each MI250x has two GPU dies, so 8 GPUs per node
- 4 Network cards (one connected to each MI250x)

Total MI250x FP performance 47.9 TFLOPS

Alvis (AI Only)

- Alvis is NAISS resource dedicated for AI and Machine learning research
  - https://www.c3se.chalmers.se/about/Alvis/
- Around 200 nodes with mixture of NVIDIA GPUs

More info in the MI250x

- Two Sub-units
- 220 compute units
  - 14080 Stream processors
- 128 GB Ram
  - 16 KB L1 Cache (per CU)
  - 16 MB L2 Cache
- Peak performance 47.9 Tflops
- Peak matrix performance 95.7 TFlops
- https://www.amd.com/en/products/server-accelerators/instinct-mi250x

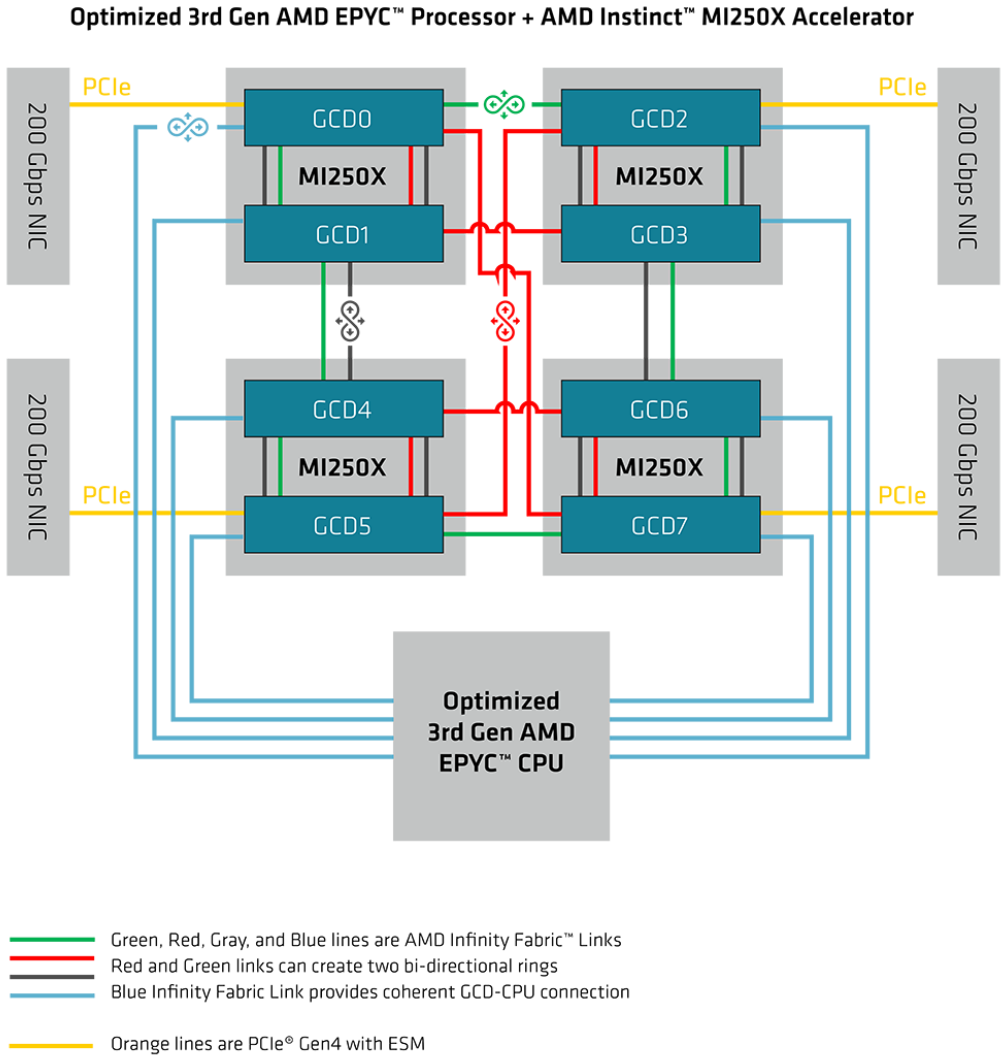
Questions?

Email – jonvin@kth.se

GPU Programming concepts

- Host and Device
- Memory Hierarchy and shared memory
- Threads and occupancy
- Latency hiding
- Locality
- Matrix cores

Host and Device



Each GPU its own memory and is controlled from the CPU.

The CPU is known as the HOST
The GPU is known as the DEVICE

Pageable and Pinned memory

- On the Host memory can be allocated as Pagable memory or Pinned memory
  - Pagable memory is the default
    - > *Pageable memory cannot be accessed directly by the GPU*
    - > *Pageable memory can potentially be paged out to disk for example by CPU*
    - > *Copies from pageable memory to GPU require temporary allocation of pinned memory*
  - Pinned memory
    - > *Pinned memory cannot be paged out to disk*
    - > *Pinned memory can be copied directly to the GPU from CPU*
      - Significantly higher bandwidth if Pinned memory is used for copies

Memory Hierarchy

- GPU memory
  - Registers
  - L1 Cache/Shared memory (combined pool)
  - L2 Cache
  - Main memory

  In order to do computational work, the data needs to be in the registers

  Only registers and shared memory can be easily influenced by programmer.
  - Shared memory directly controlled
  - Registers controlled via compiler directives

Threads and occupancy

- Threads
- Threadblocks. Max 1024 threads in a block
- occupancy

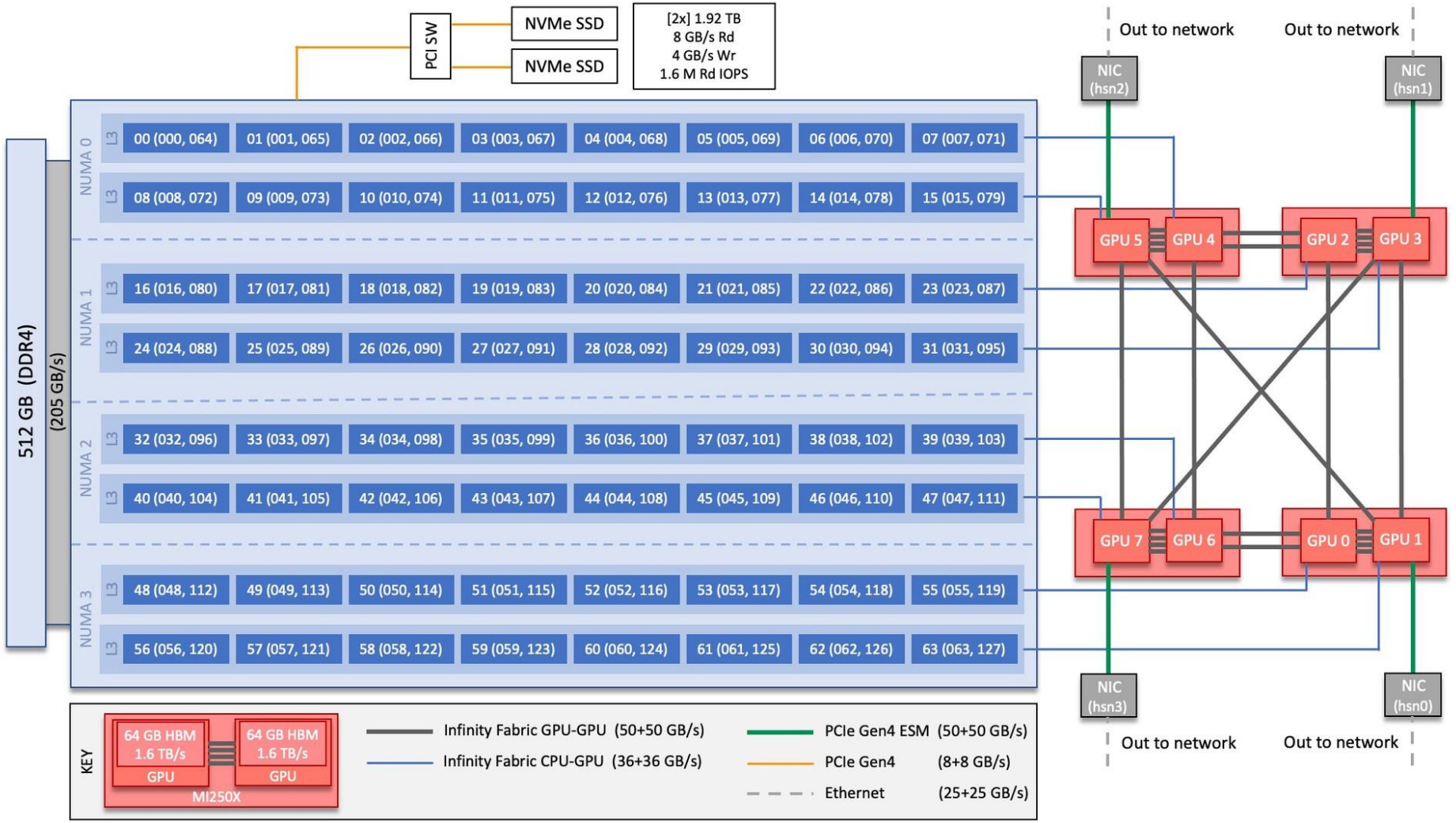
Register spilling

- Scratch memory (or called local memory by NVIDIA) and be used for register spilling.
- Register spilling used to reduce number of active registers, increasing the potential number of active threads.
- Scratch/local memory relatively slow, so not always advantageous to spill
  - Compiler uses heuristics, which can be wrong
- Number of registers used can be influenced by compiler directives.

Latency Hiding

- Copying data from main memory takes a very long time
- GPUs try to hide this by having many threads active at the same time
- Number of active threads is limited by
  - Hardware limit
  - Registers per thread
  - Shared memory used “per thread”
    - *Technically per thread block, divided by threads in the block*
- Occupancy being high is generally good for performance but not everything

GPU Locality and GPU Binding - LUMI



64 CPU cores, single socket CPU

CPU has 4 NUMA nodes, and 8 L3 cache regions

4 MI250x GPUs, with 2 GCDs per GPU

4 Network interface cards attached to odd numbered GCDs

Source : https://docs.olcf.ornl.gov/systems/frontier_user_guide.html#frontier-compute-nodes

NUMA Nodes – CPU

- Multi-processor systems where resources are divided into multiple nodes or domains
- A NUMA domain is a grouping of cores, memory and other peripherals (e.g. PCIe busses etc.)
- Each CPU core is attached to its own local memory while being able to access memory attached to other processors
- Local memory accesses are fast while remote memory accesses have a higher latency and lower bandwidth

Job Script – One MPI rank per GPU

```
cat << EOF > select_gpu
#!/bin/bash
export ROCR_VISIBLE_DEVICES=\$SLURM_LOCALID
exec \$*
EOF

chmod +x ./select_gpu
CPU_BIND="map_cpu:48,56,16,24,1,8,32,40"
export MPICH_GPU_SUPPORT_ENABLED=1
srun --cpu-bind=${CPU_BIND} ./select_gpu <executable> <args>
```

Matrix Cores

- Specialised hardare
- Initially developed to accelerate Deep Learning workflows
- Accelerates calculations of the form
  - C = a AB + bC
- Simplest to access them to use libraries (other methods very advanced)
  - rocBLAS or rocWMMA

Questions?

Email – jonvin@kth.se