DOI:10.5281/zenodo.10556523

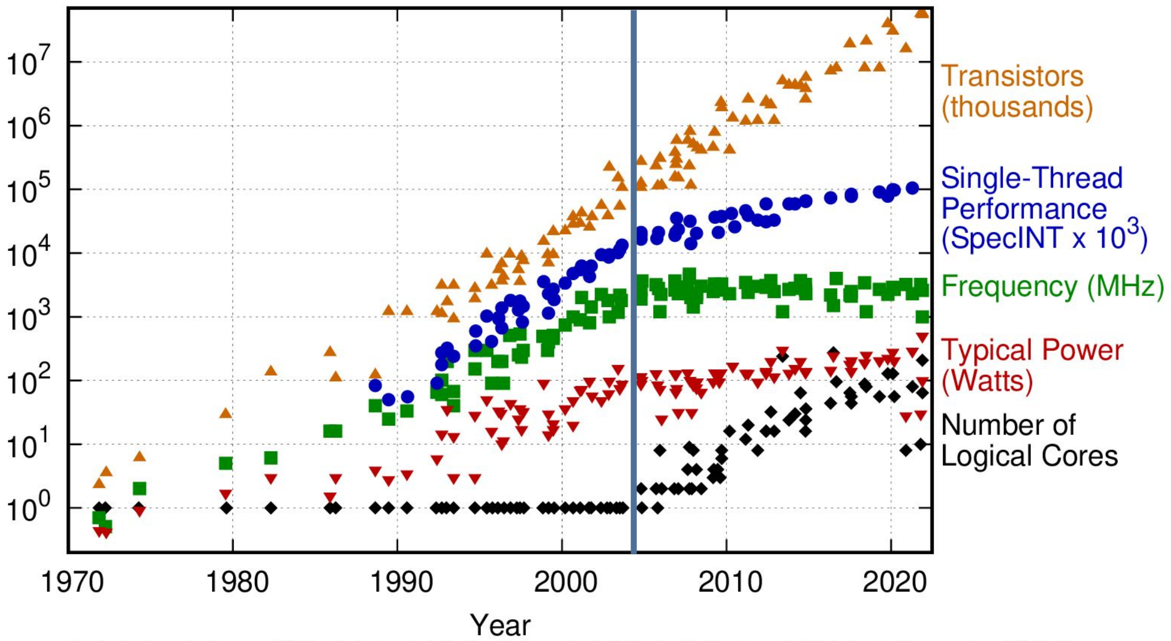
AMD GPU support in GROMACS

Andrey Alekseenko
KTH Royal Institute of Technology & SciLifeLab
andreyal@kth.se

Moore’s law

- “Number of transistors in an integrated circuit doubles about every two years”
- Before ~2005: increasing single-core performance
- After ~2005: increasing the number of cores

50 Years of Microprocessor Trend Data



Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten
New plot and data collected for 2010-2021 by K. Rupp

https://github.com/karlrupp/microprocessor-trend-data

Graphics processing units

- GPU: Most common type of *accelerator*
- Specialized hardware
  - Highly parallel
  - Not self-sufficient
- Separate memory
- Different programming paradigm

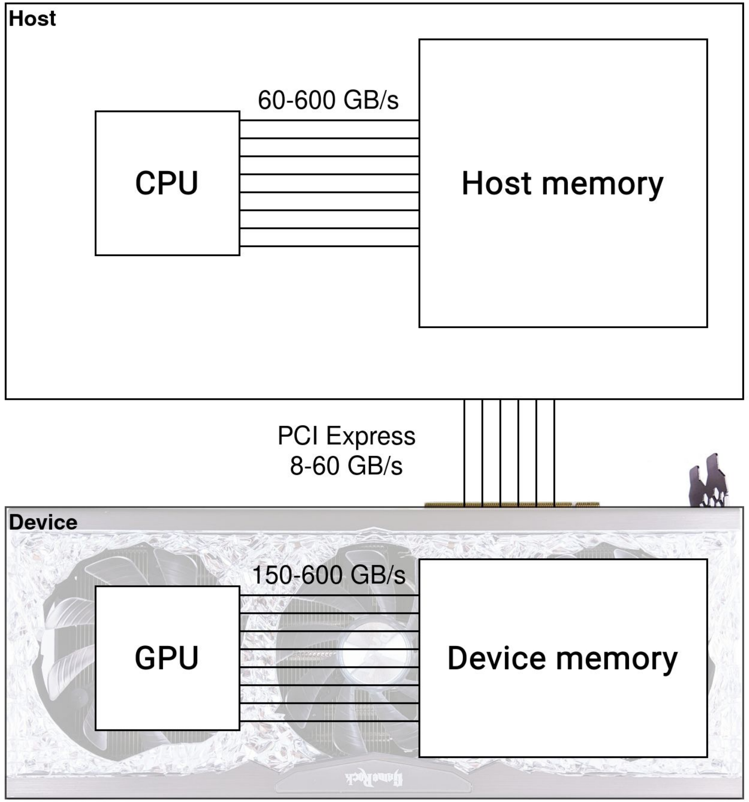
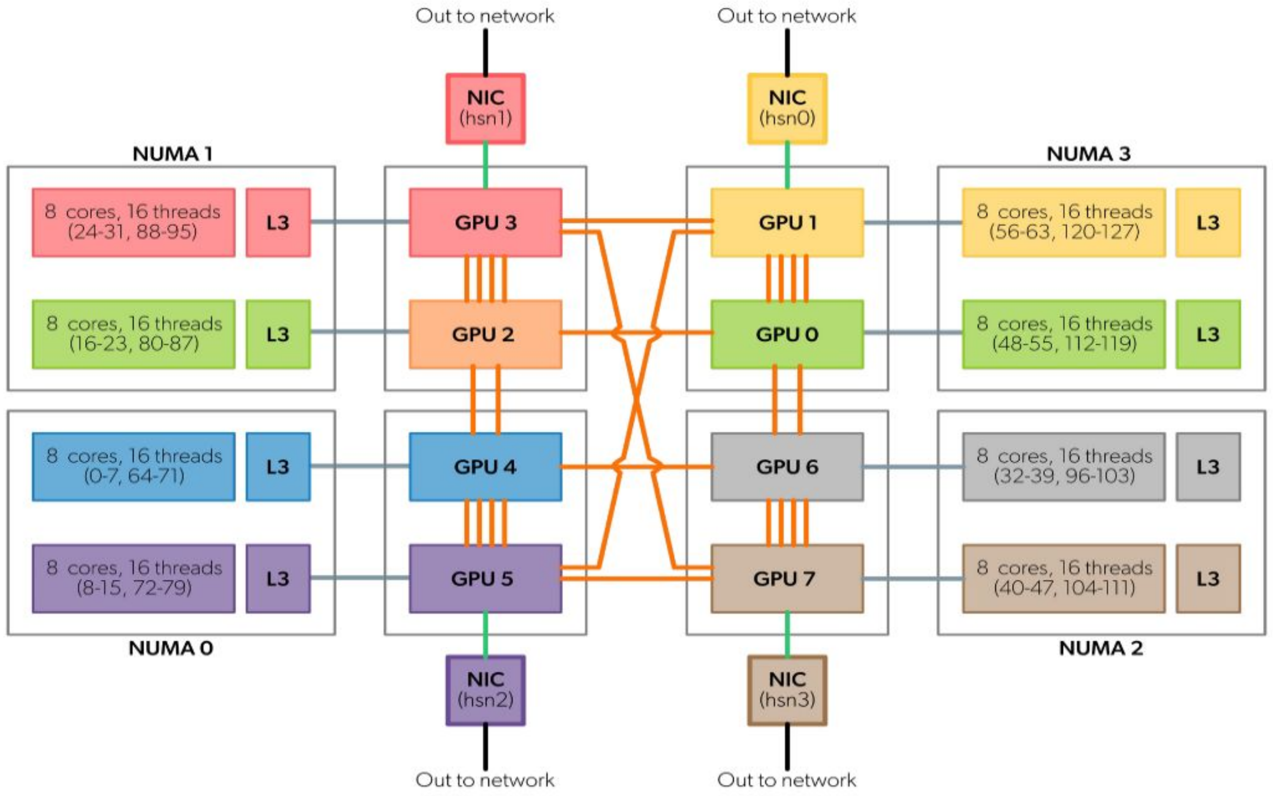


Figure adapted from the Carpentry GPU Programming lesson; GPU photo by @zelebb on Unsplash

LUMI-G node architecture

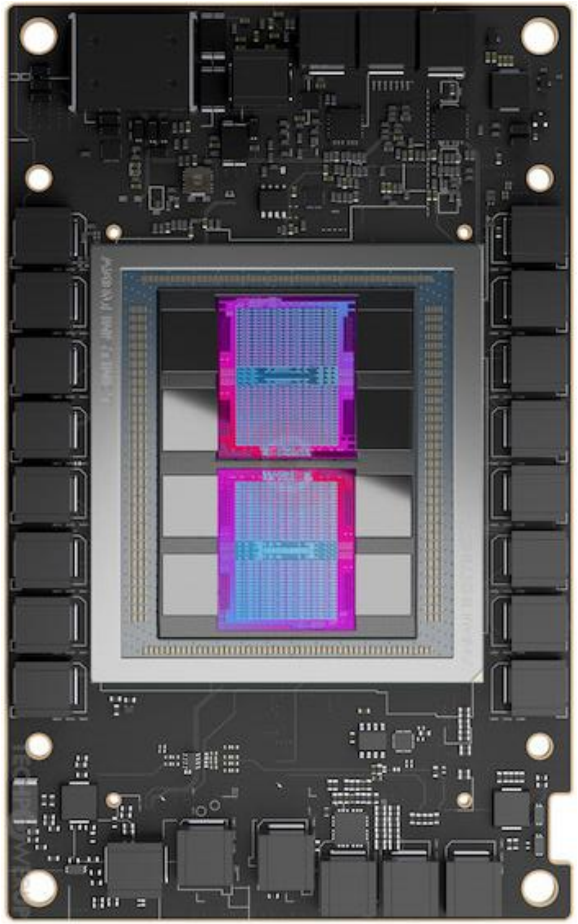


AMD MI250X GPU

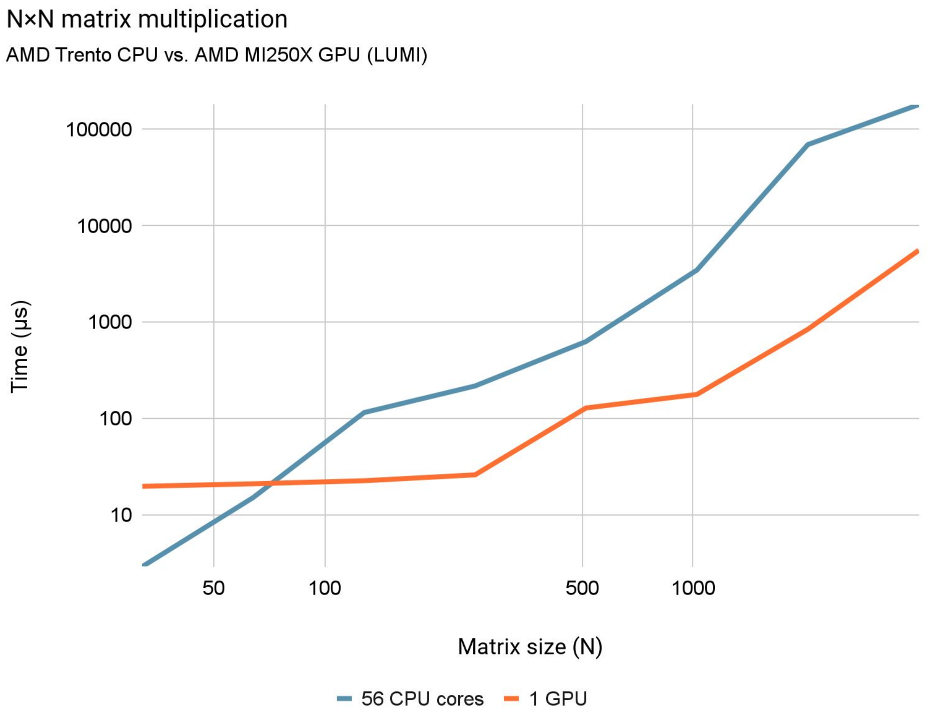
- One MI250X GPU has two GCDs
- Each GCD is a logical GPU

GCD specs:

- AMD CDNA2 architecture (gfx90a)
- Peak Engine Clock: 1700 MHz
- Peak FP32 Performance: 90 TFLOPs
- Dedicated Memory Size: 64 GB



Bandwidth vs. latency



```julia
using AMDGPU
using BenchmarkTools

N = 5:12
for n in N
  A = rand(2^n, 2^n);
  A_d = ROCArray(A);
  @btime $A * $A;
  @btime begin
    $A_d * $A_d;
    AMDGPU.synchronize()
  end
end
```

Homogeneous / heterogeneous acceleration

- Homogeneous: use a single type of hardware
  - Accelerator challenges
    - Porting effort / feature support
    - Scaling
- Heterogeneous: use different hardware together
  - Typically, CPU + GPU
  - Challenges
    - Data movement
    - Latencies
    - Load balancing

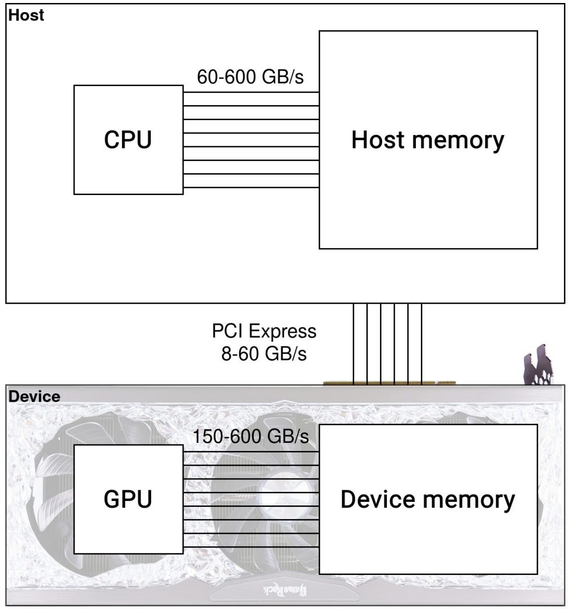
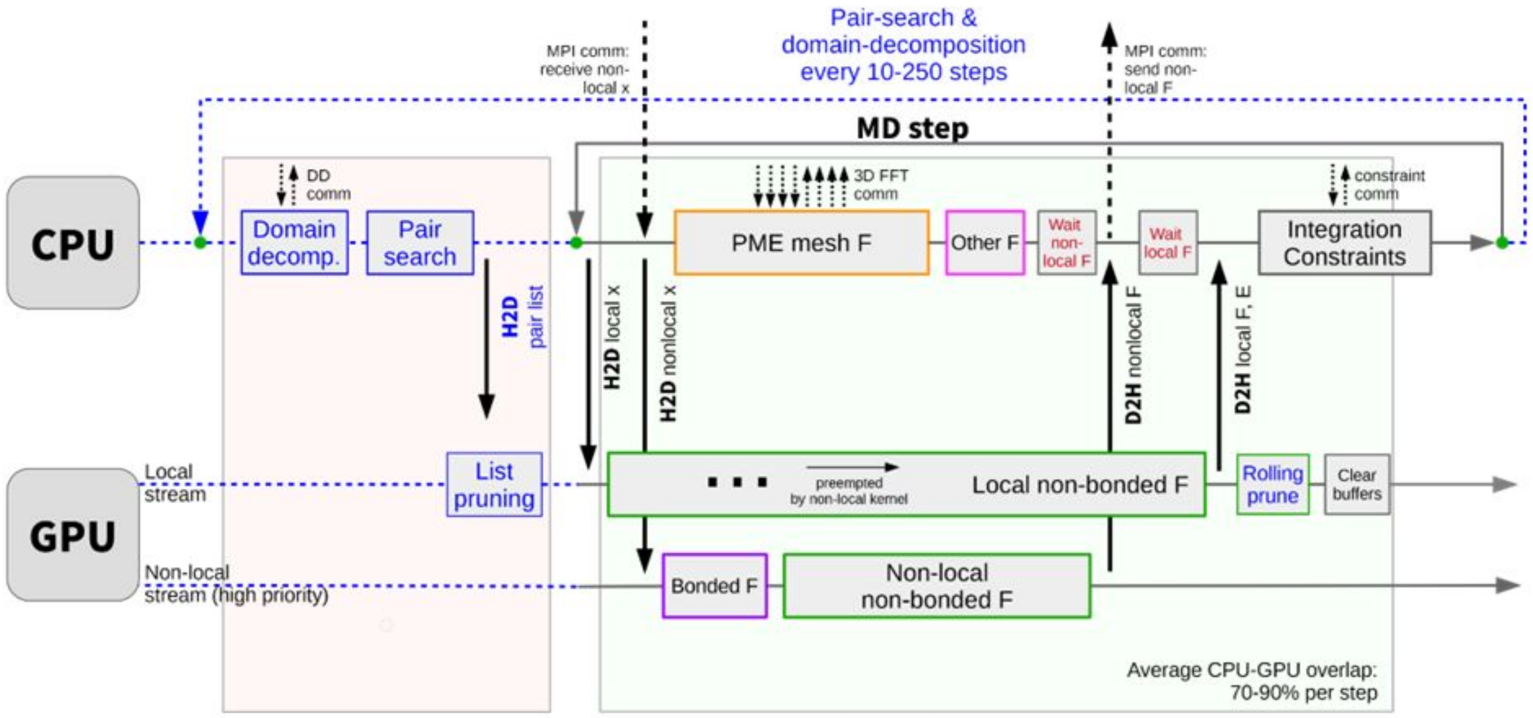


Figure adapted from the Carpentry GPU Programming lesson; GPU photo by @zelebb on Unsplash

GPU scheduling



Using GPUs for compute

- Existing software
  - Many HPC and AI codes already support GPUs
- Task-specific frameworks / libraries
  - ML (PyTorch), CV (OpenCV), math (cuBLAS), …
- High-level languages
  - Python (PyCUDA, Numba), Julia (AMDGPU.jl), …
- Directive-based methods
  - OpenMP, OpenACC
- Native GPU code
  - CUDA, HIP, OpenCL, SYCL, Kokkos, …

Criteria:

- Effort
- Portability
- Performance
- Openness

GPU frameworks (subjective comparison)

		NVIDIA CUDA	ROCm	OpenCL	SYCL
	Maturity of API / ecosystem	+++ / +++	++ / +	+++ / ++	++ / +
	Open standard	X	X	√	√
	Single-source model	√	√	X	√
	Modern C++ support	√	√	X	√
HW support	AMD	X	√√	√	++
	intel	X	X	√√	√√√
	NVIDIA	√√√	+	√	++

Why SYCL™?

- Open standard
- Two independent free (libre) implementations
  - Intel oneAPI DPC++
  - AdaptiveCpp (aka hipSYCL)
- Broad hardware support
- Single source, modern C++
- Standardized interoperability with native libraries

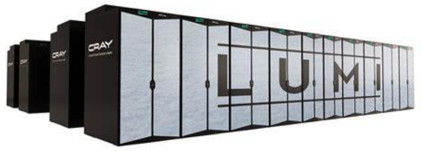
GPU frameworks in GROMACS 2020

	NVIDIA CUDA	OpenCL	SYCL
Short-range non-bonded (-nb)	√	√	
Long-range PME (-pme)	√	√	
Bonded (-bonded)	√		
Update (-update)	√		
PME Decomposition			
Direct GPU communications	√		
Graph-based scheduling			
Supported GPUs	NVIDIA	AMD,Intel, NVIDIA	

GPU frameworks in GROMACS 2023

	NVIDIA CUDA	OpenCL	SYCL
Short-range non-bonded (-nb)	√	√	√
Long-range PME (-pme)	√	√	√
Bonded (-bonded)	√		√
Update (-update)	√		√
PME Decomposition	√	X	√*
Direct GPU communications	√	X	√
Graph-based scheduling	√	X	
Supported GPUs	NVIDIA	AMD, Apple, Intel, NVIDIA	AMD, Intel, NVIDIA

GPU frameworks in GROMACS 2023



	NVIDIA CUDA	OpenCL	SYCL
Short-range non-bonded (-nb)	√	√	√
Long-range PME (-pme)	√	√	√
Bonded (-bonded)	√		√
Update (-update)	√		√
PME Decomposition	√	X	√*
Direct GPU communications	√	X	√
Graph-based scheduling	√	X	
Supported GPUs	NVIDIA	AMD, Apple, Intel, NVIDIA	AMD, Intel, NVIDIA

SYCL compared to HIP

- AMD suggests HIP/ROCm stack for their GPUs
  - Used by most other codes targeting AMD GPUs
- SYCL is open, portable, modern
  - Saved developer time: more features, less bugs
  - Two supported implementations: DPC++ and AdaptiveCpp
- GROMACS recommends AdaptiveCpp (hipSYCL) for AMD GPUs
- AdaptiveCpp is built on top of HIP
  - SYCL implementation, developed at Heidelberg University
  - Supports AMD, NVIDIA, Intel GPUs

SYCL compared to HIP

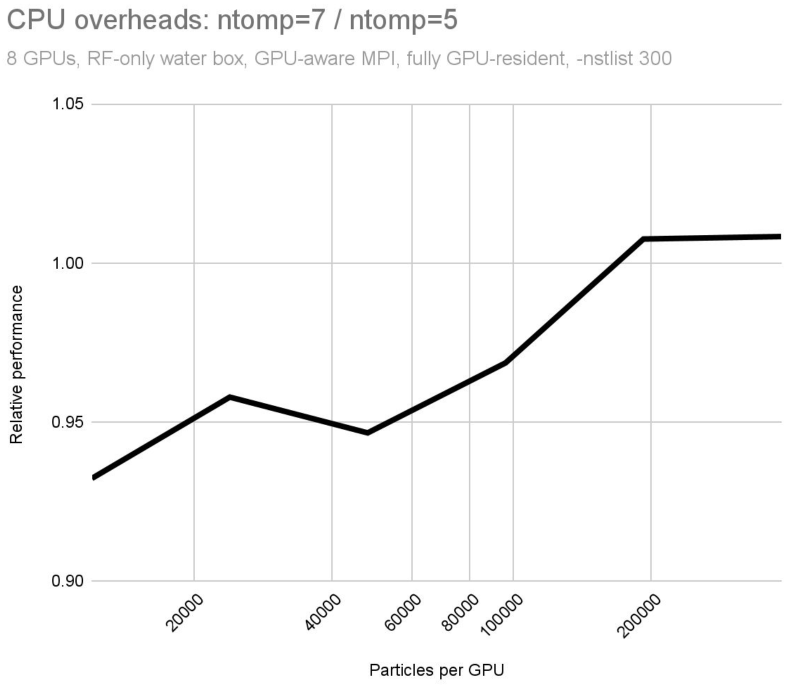
- AdaptiveCpp is a layer on top of HIP
  - Same compiler
  - Different runtime / scheduler

- Minimal overhead for *bandwidth-bound* simulation
- Noticeable overhead for *latency-bound* simulations (< 50k particles per GPU)
  - To be improved soon
  - Reducing CPU usage (lower -ntomp) can help

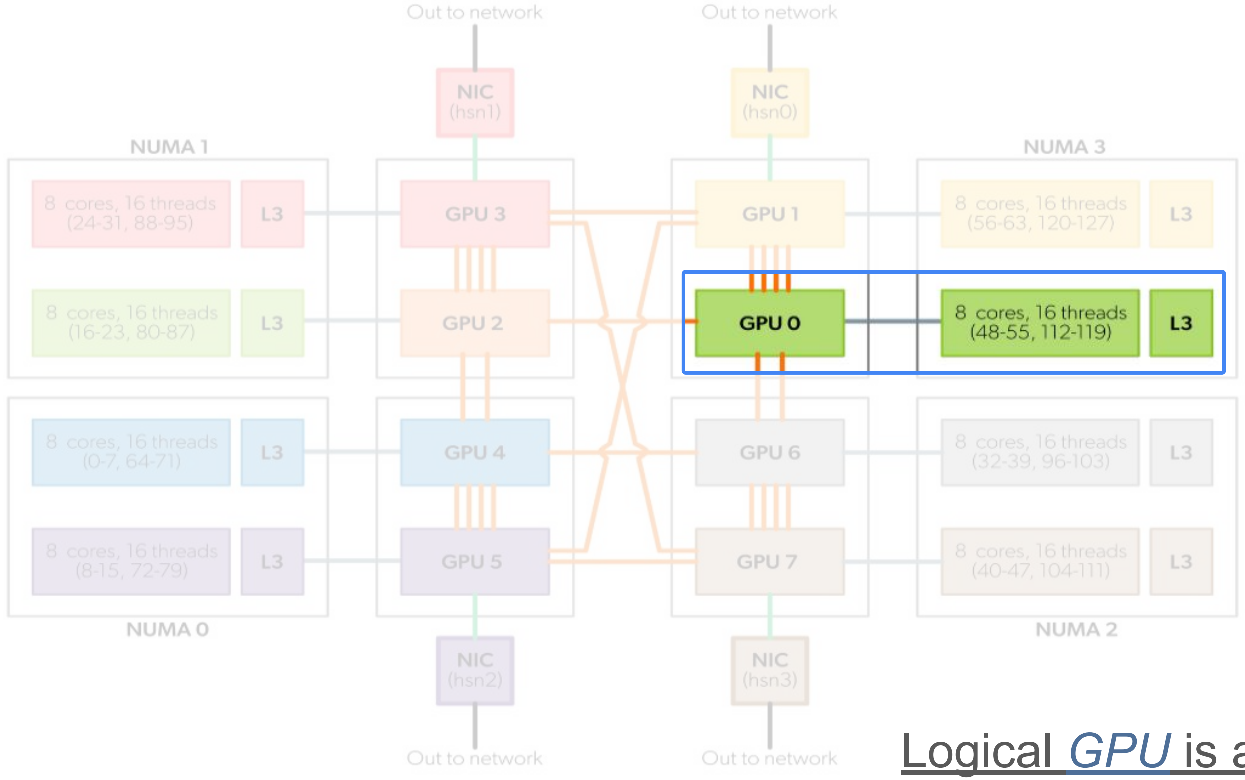
Latency-bound case

- AdaptiveCpp uses extra threads to submit work to GPU
- Theoretically, allows more flexibility
- In practice, incurs overhead
    - Especially with few CPU cores

- Typically, 7 threads per GPU is fine
    - But watch out when approaching the scaling limit!



LUMI-G node architecture



Out to network
Out to network
NIC (hsn1)
NIC (hsn0)
NUMA 1
NUMA 3
8 cores, 16 threads (24-31, 88-95)
L3
GPU 3
GPU 1
8 cores, 16 threads (56-63, 120-127)
L3
8 cores, 16 threads (16-23, 80-87)
L3
GPU 2
GPU 0
8 cores, 16 threads (48-55, 112-119)
L3
8 cores, 16 threads (0-7, 64-71)
L3
GPU 4
GPU 6
8 cores, 16 threads (32-39, 96-103)
L3
8 cores, 16 threads (8-15, 72-79)
L3
GPU 5
GPU 7
8 cores, 16 threads (40-47, 104-111)
L3
NUMA 0
NUMA 2
NIC (hsn2)
NIC (hsn3)
Out to network
Out to network

8 GPUs/GCDs per node
8 CCDs per node
GCD ↔ CCD mapping

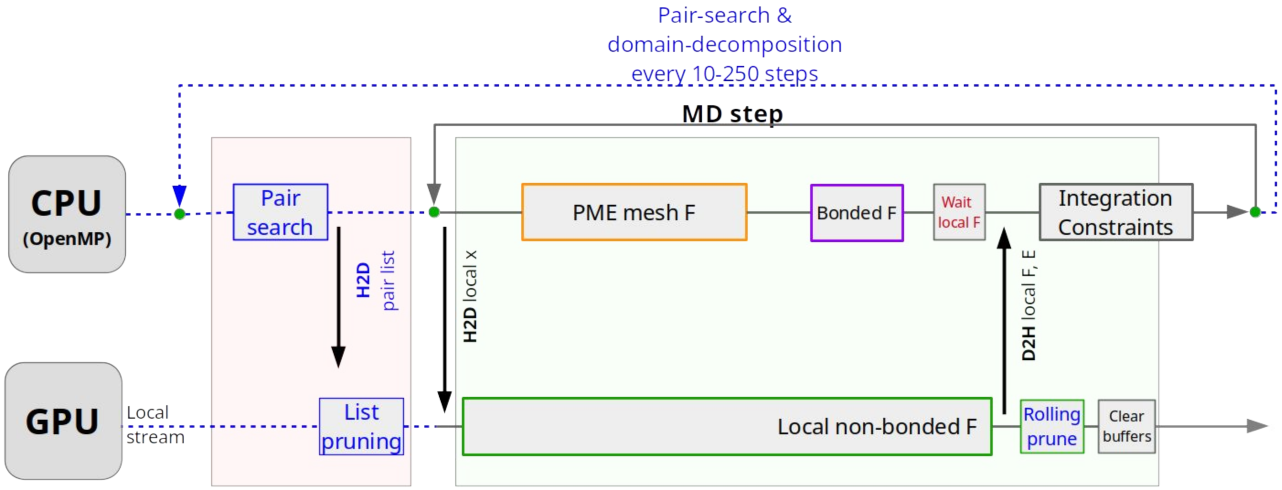
7 cores per CCD
(+1 reserved core)

Logical *GPU* is a single hardware *GCD*

Parallelization overview: single GPU

1 MPI rank:

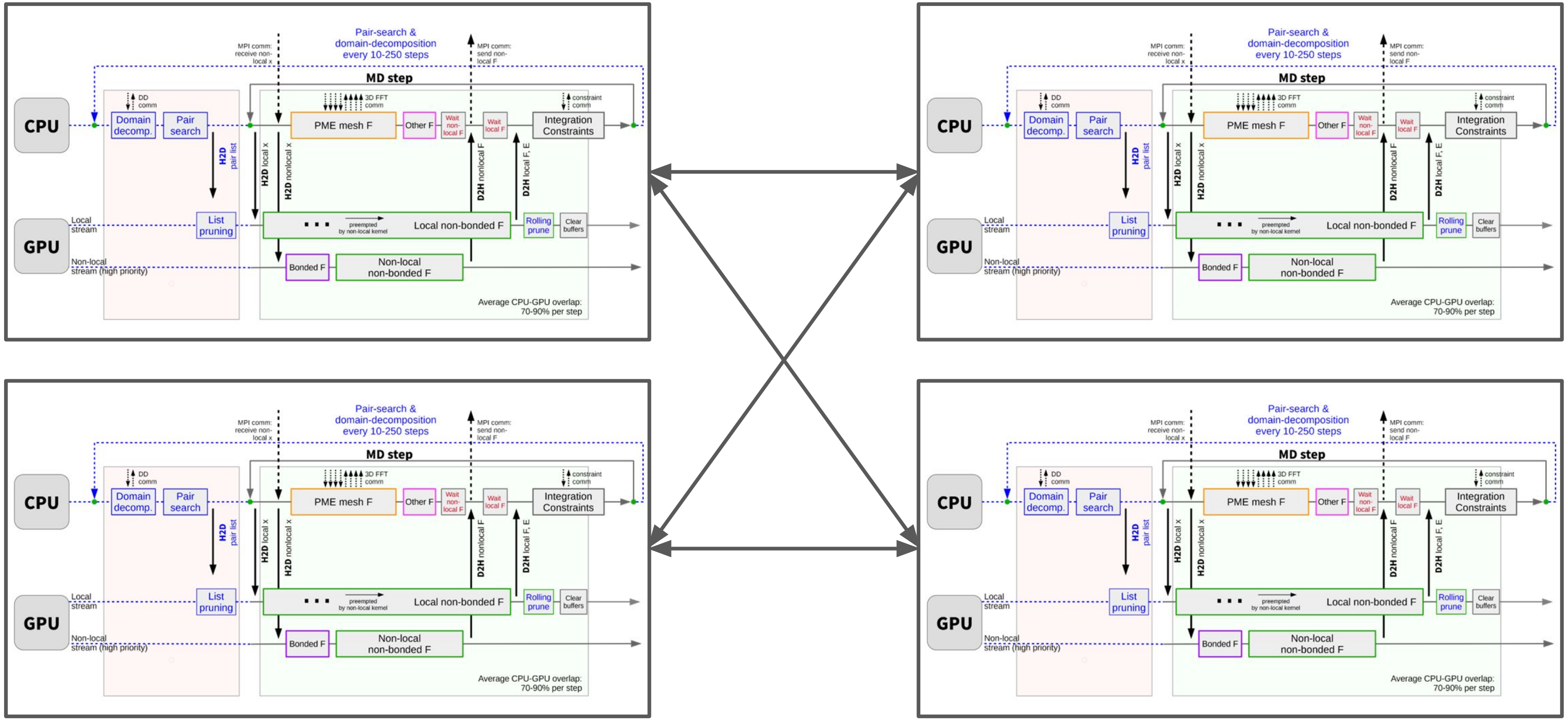
- 1 GPU
- 1-7 CPU cores



Resource control:

- `-ntomp / OMP_NUM_THREADS`
- `-nb`, `-pme`, `-bonded`, `-update`
- `-nstlist`

Parallelization overview: multiple GPUs



Parallelization overview: multiple GPUs

Multiple MPI ranks:

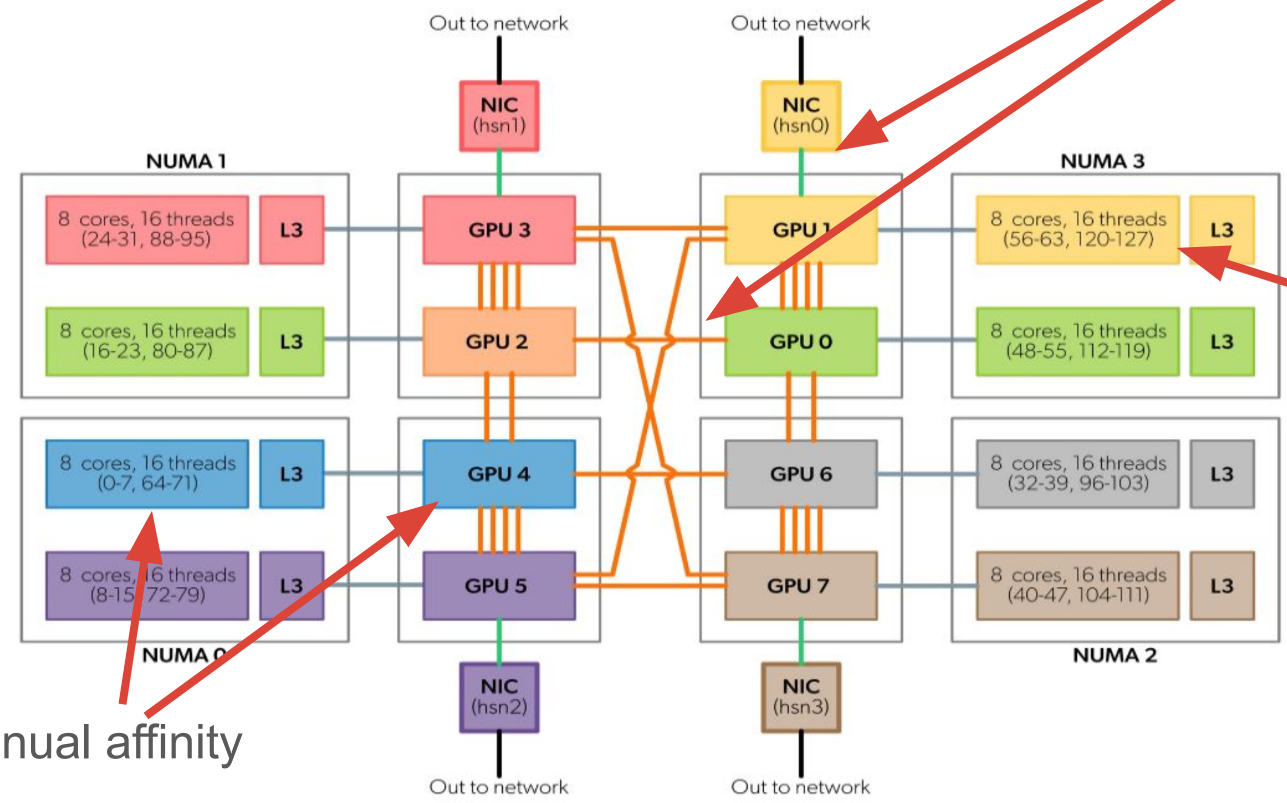
- 1 GPU **per rank**
- 1-7 cores **per rank**

8 GPUs per node, can use multiple nodes

- Better resource control when allocating full node
- Cray MPI is GPU-aware if you tell it to be

Ranks can share the GPU

LUMI-G node architecture



Out to network
Out to network
NIC (hsn1)
NIC (hsn0)
NUMA 1
NUMA 3
8 cores, 16 threads (24-31, 88-95)
L3
GPU 3
GPU 1
8 cores, 16 threads (56-63, 120-127)
8 cores, 16 threads (16-23, 80-87)
GPU 2
GPU 0
8 cores, 16 threads (48-55, 112-119)
8 cores, 16 threads (0-7, 64-71)
GPU 4
GPU 6
8 cores, 16 threads (32-39, 96-103)
8 cores, 16 threads (8-15, 72-79)
GPU 5
GPU 7
8 cores, 16 threads (40-47, 104-111)
NUMA 0
NUMA 2
NIC (hsn2)
NIC (hsn3)
Out to network
Out to network

Direct GPU comm.

8c/16t per GPU
(1 core reserved)

Manual affinity