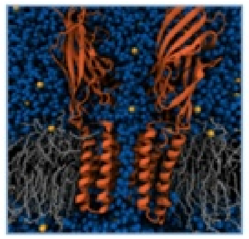
HPC codesign in GROMACS

Szilárd Páll
pszilard@kth.se

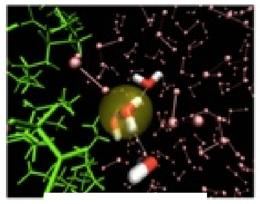
Workshop on Software Co-Design Actions in European Flagship HPC Codes
ISC 2022
June 2, 2022

BioExcel Center of Excellence

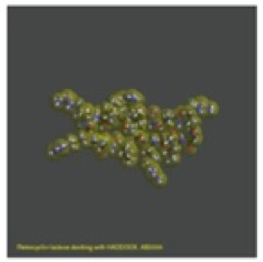
Improving the performance and scalability of key codes towards the Exascale



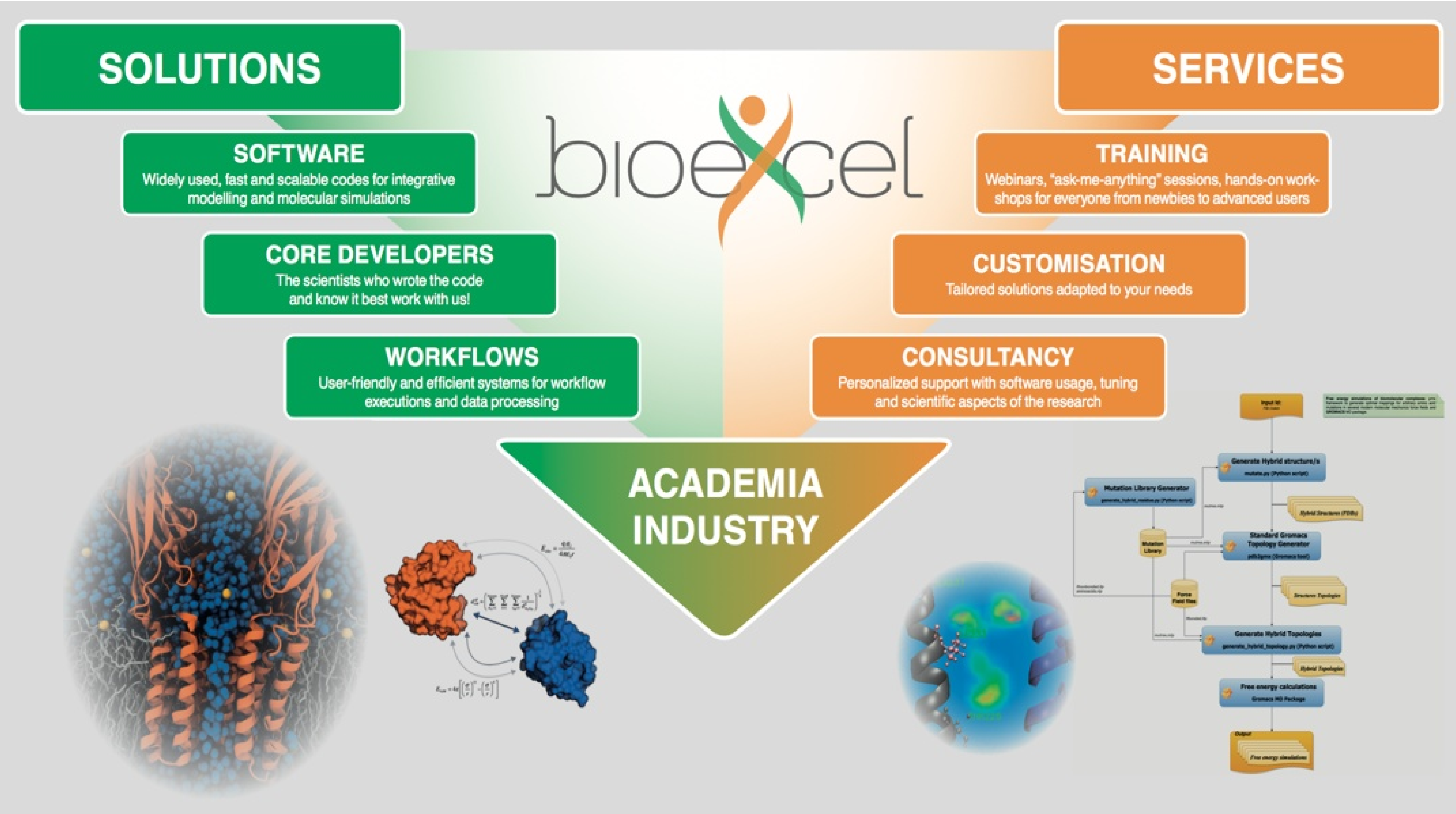
MD simulations /GROMACS/



QMMM /CP2K/



Docking /HADDOCK/



SOLUTIONS

SOFTWARE
Widely used, fast and scalable codes for integrative modelling and molecular simulations

CORE DEVELOPERS
The scientists who wrote the code and know it best work with us!

WORKFLOWS
User-friendly and efficient systems for workflow executions and data processing

bioexcel

SERVICES

TRAINING
Webinars, "ask-me-anything" sessions, hands-on workshops for everyone from newbies to advanced users

CUSTOMISATION
Tailored solutions adapted to your needs

CONSULTANCY
Personalized support with software usage, tuning and scientific aspects of the research

ACADEMIA
INDUSTRY

Devising efficient workflow environments with associated data integration

biobb

COMMON WORKFLOW LANGUAGE

py COMPSs

Open for Innovation KNIME

Galaxy PROJECT

Open PHACTS

Training Program → Competency profiles → Mapped competencies to training → Gap analysis

krc.bioexcel.eu

Provide consultancy and training to end users in academia and industry

Goals:

- Develop key applications (incl. GROMACS) for exascale;
- Develop workflow solutions
- Training/support to academia and industry
- Establish a long-term organizational structure

FAST. FLEXIBLE. FREE.

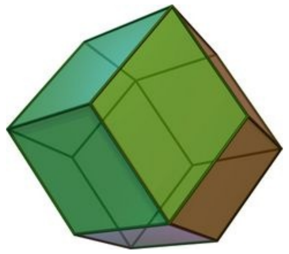
GROMACS

- **Classical MD** code
  - supports all major force-fields
  - broad algorithm support
- **Development**:

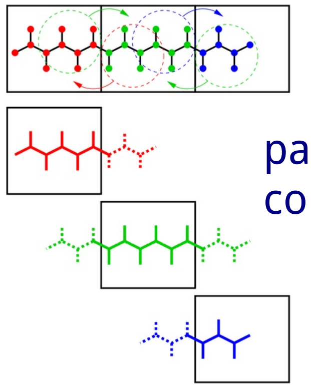
  Stockholm Sweden & partners worldwide
- **Large user base**:
  - One of the top HPC codes worldwide

    deployed on most clusters
  - 10k's academic & industry users

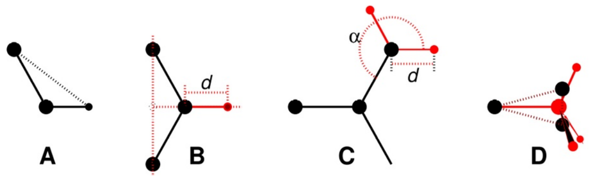
- **Open source**: LGPLv2
- **Open development:**
  - code review & bug-tracker:https://gitlab.com/gromacs



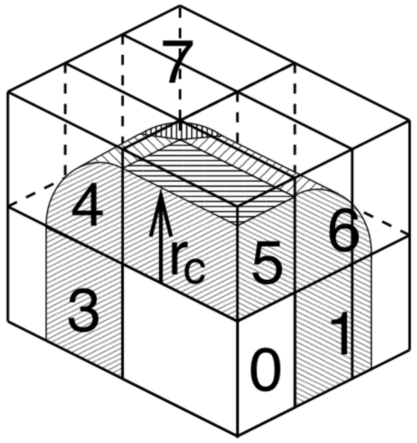
arbitrary
units cells



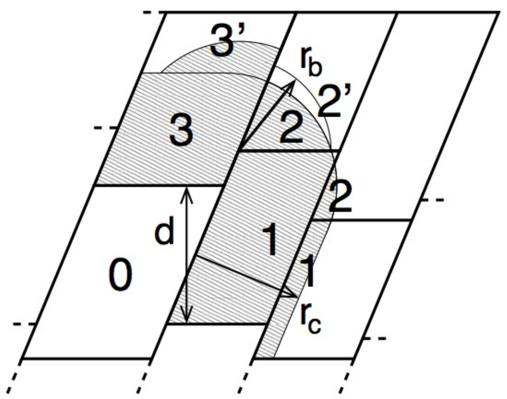
parallel
constraints



virtual interaction sites



Eighth shell
domain
decomposition



Triclinic unit cell with
load balancing and
staggered cell boundaries

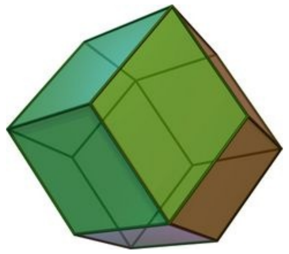
FAST. FLEXIBLE. FREE.

GROMACS

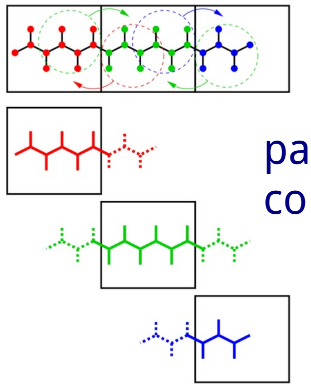
- Focus on **high performance:**

  efficient algorithms & highly-tuned parallel code
- **Bottom-up performance oriented design:**
  - absolute performance over “just scaling”
- Focus on **portability**
  - Linux distro integration and CI
  - regular testing on all HPC arch
  - SIMD portability library, GPU abstraction layer
  - open standards-based languages/APIs

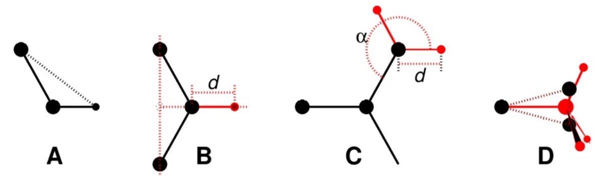
- **Modern development workflow**
  - mandatory open code review for >10 years
  - tiered CI testing / verification



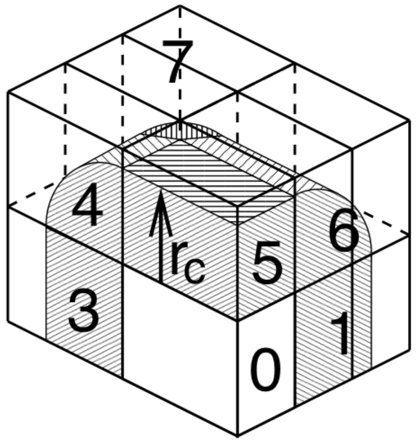
arbitrary units cells



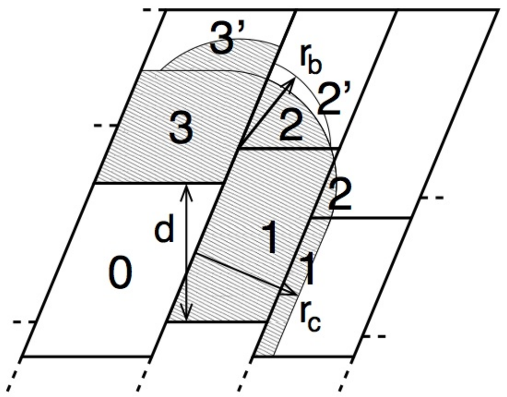
parallel constraints



virtual interaction sites

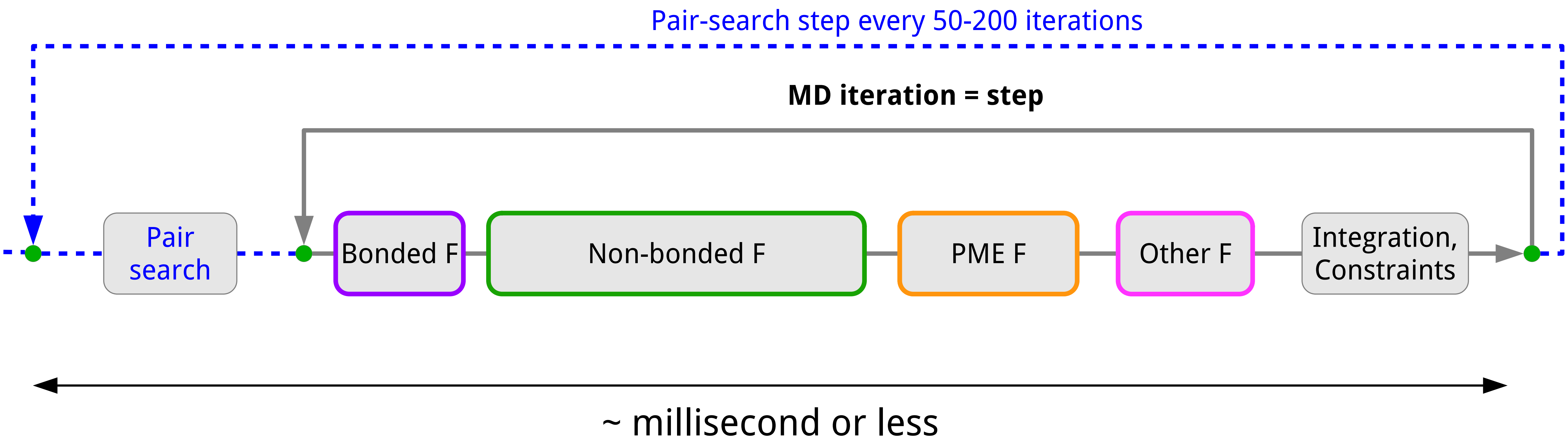


Eighth shell domain decomposition



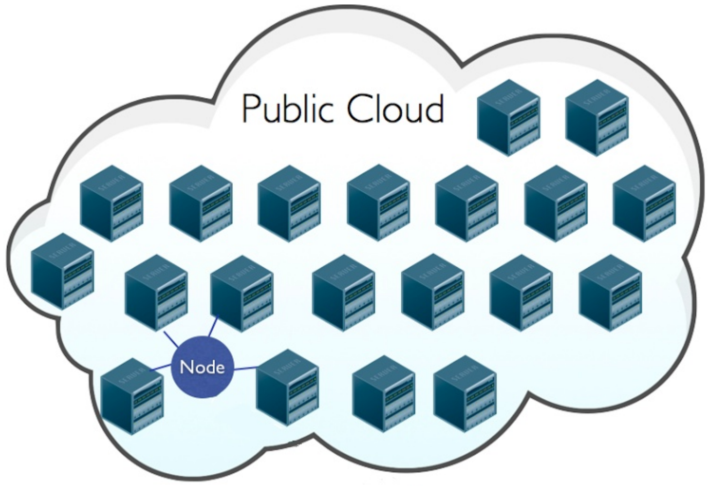
Triclinic unit cell with load balancing and staggered cell boundaries

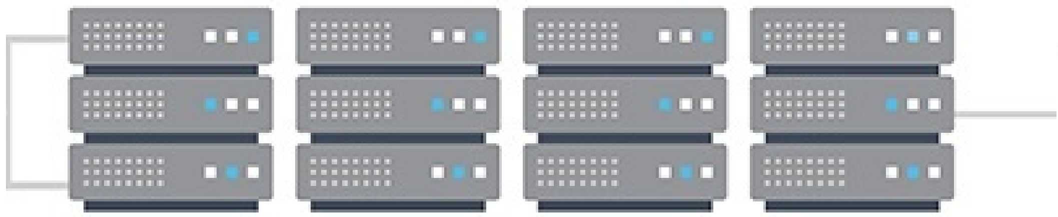
MD: computational challenge



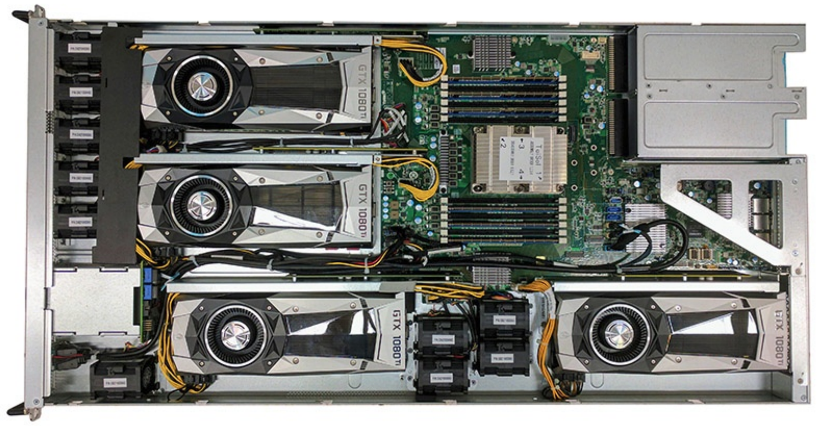
- Simulation vs real-world **time-scale gap**
  - Every simulation: 10^8 –10^{15} steps
  - Every step: 10^6 – 10^9 FLOPs
- Main goal of parallelization:
  - study molecular systems: tackle the time- or length-scale challenge
  - typically requires: **strong scaling**, increasingly **ensemble**
- MD codes at peak: ~**100 μs / step** (on commodity hardware)
  - <100 atoms/core at peak
  - <10000 atoms / GPU

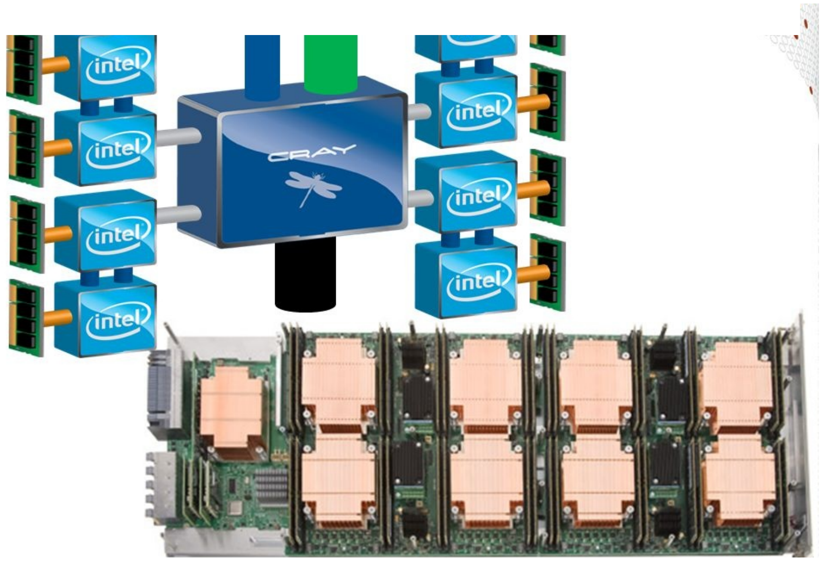
Multiple levels of hardware parallelism

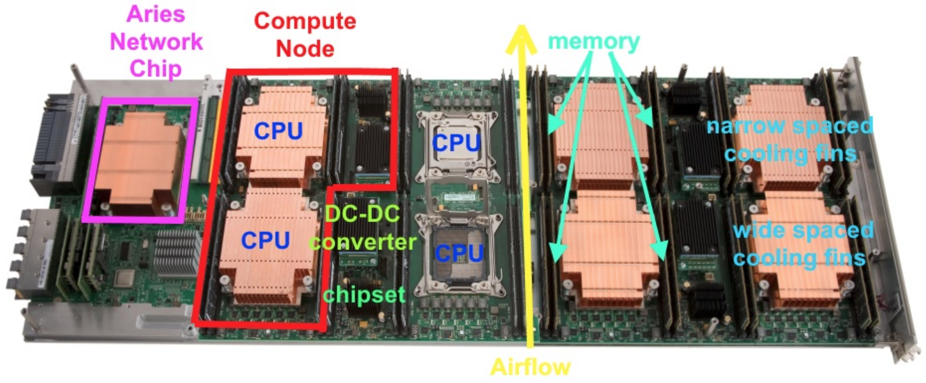




Compute cluster or cloud
Networked computers:
topology, bandwidth, latency

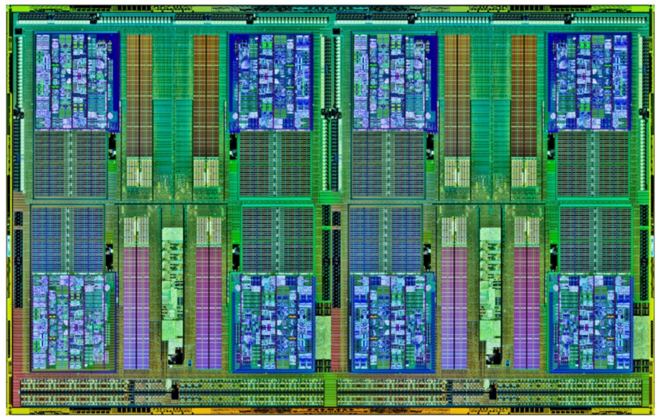




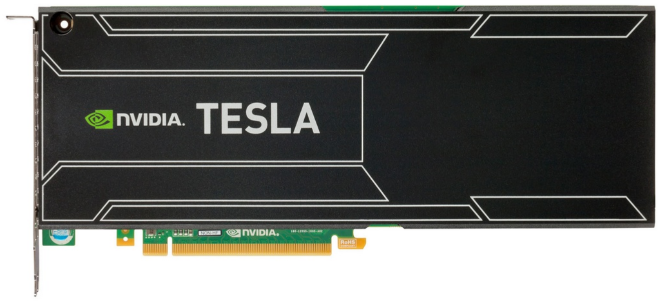


Compute node / workstation

NUMA topology, PCIe

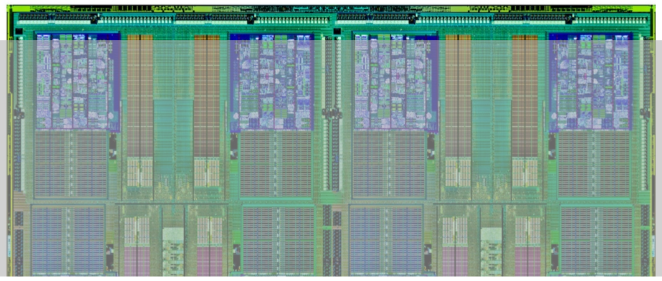


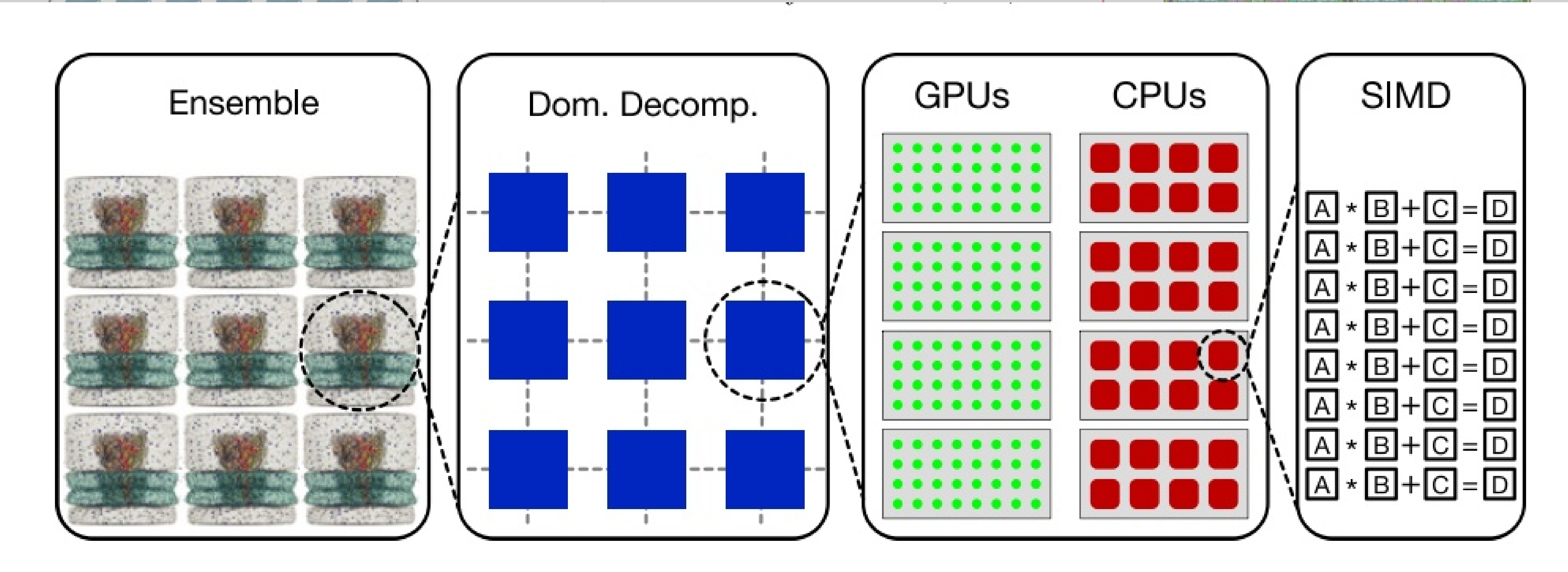
up to 512-bit vector units/core
=>
up to 16 single precision ops/clock



Multicore CPU &
manycore GPU
caches, interconnects

Multiple levels of **hardware parallelism**
Multiple levels of **parallelization**





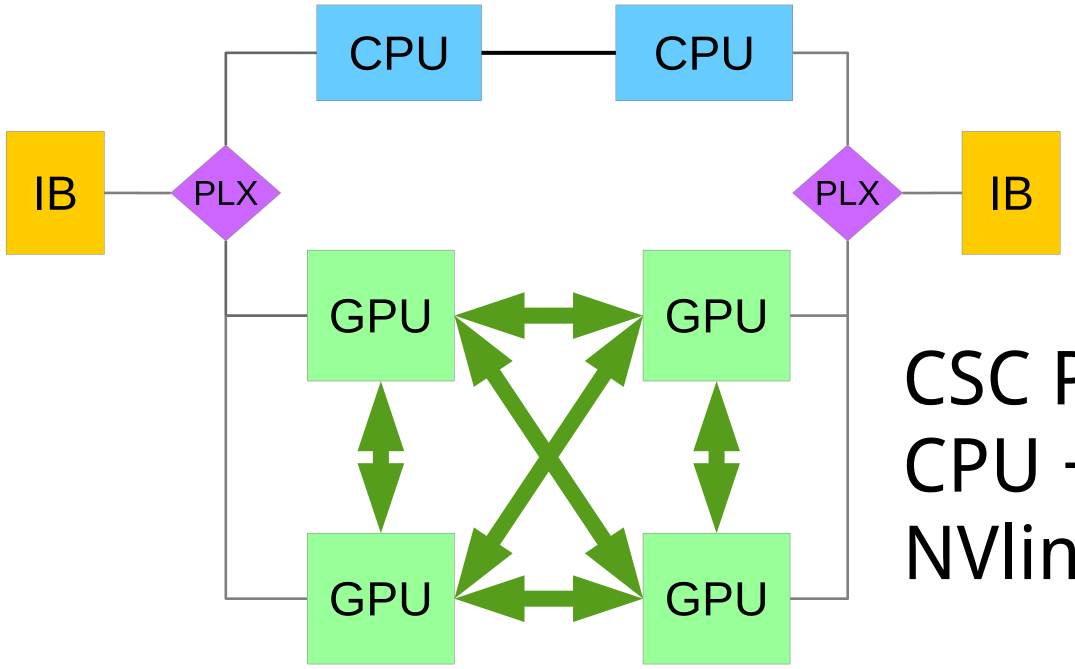
- Mapping the problem to the hardware:

  expose parallelism (algorithms) & **express parallelism** (implementation)

- Need to choose the right:

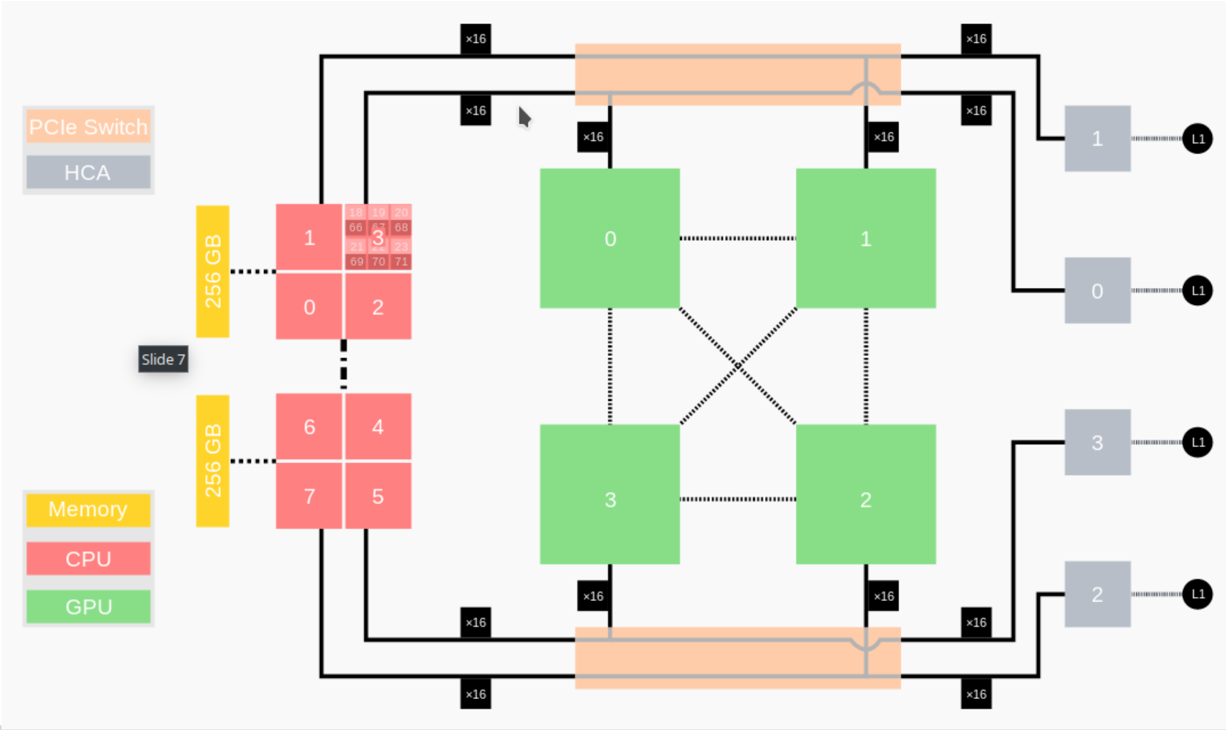
  granularity & **abstraction** (problem & hardware-specific)

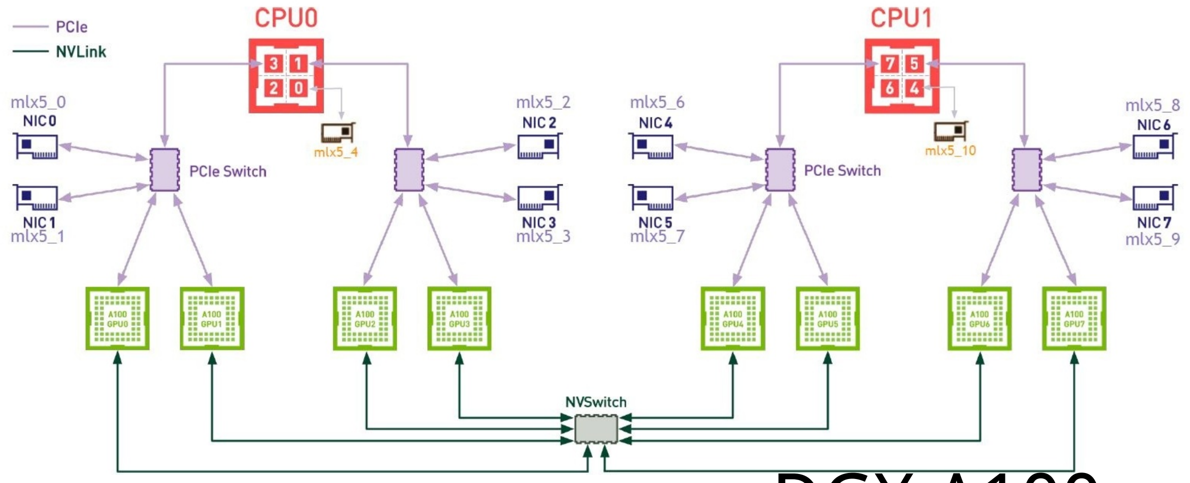
HPC nodes today/soon



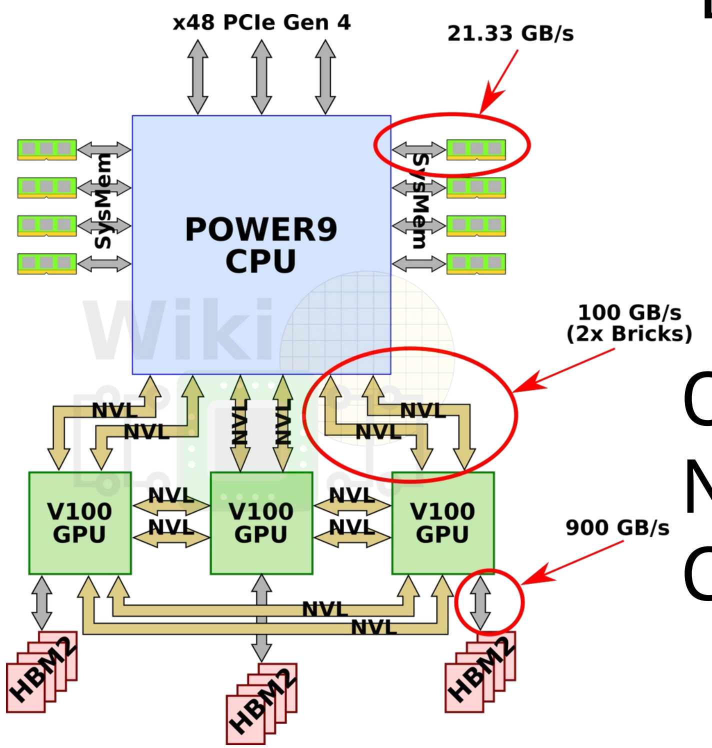
CSC Puhti :2 CPU + 4 GPU+ NVlink, 2 NIC

JUWELS-Booster: 2 CPU + 4 GPU w NVlink + 4 NIC





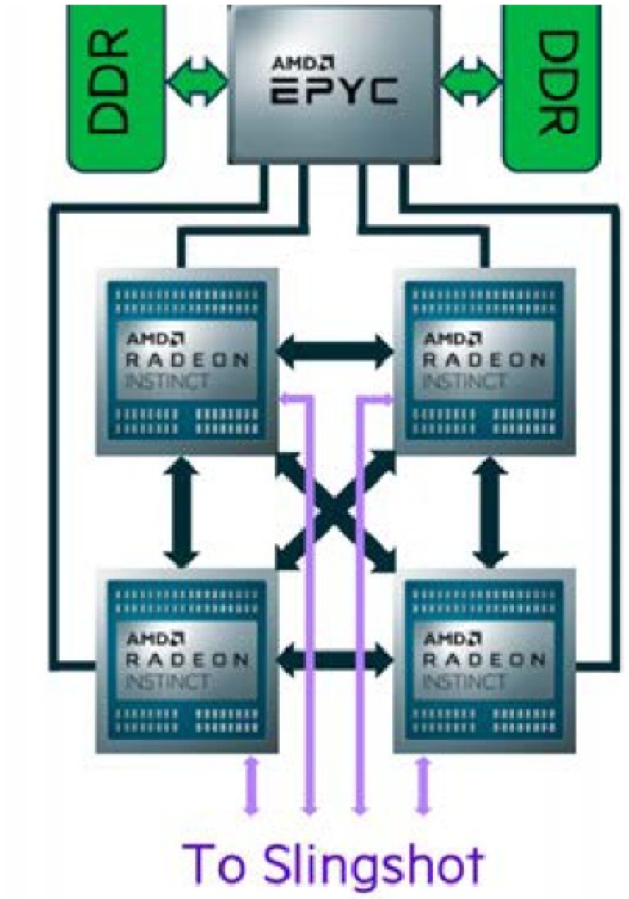
DGX A100



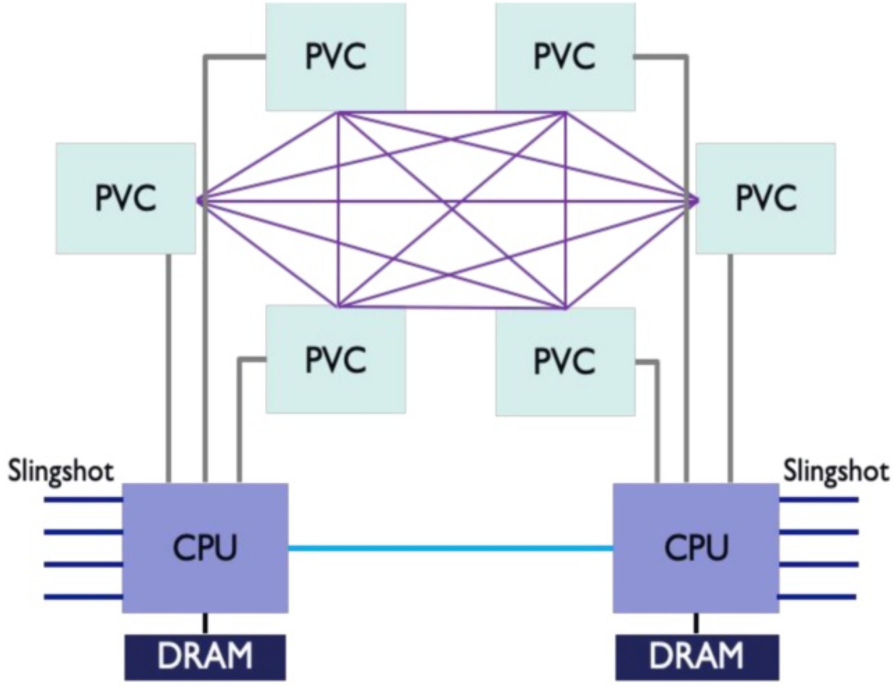
ORNL Summit: NVLink CPU/GPU-GPU

Shared under CC

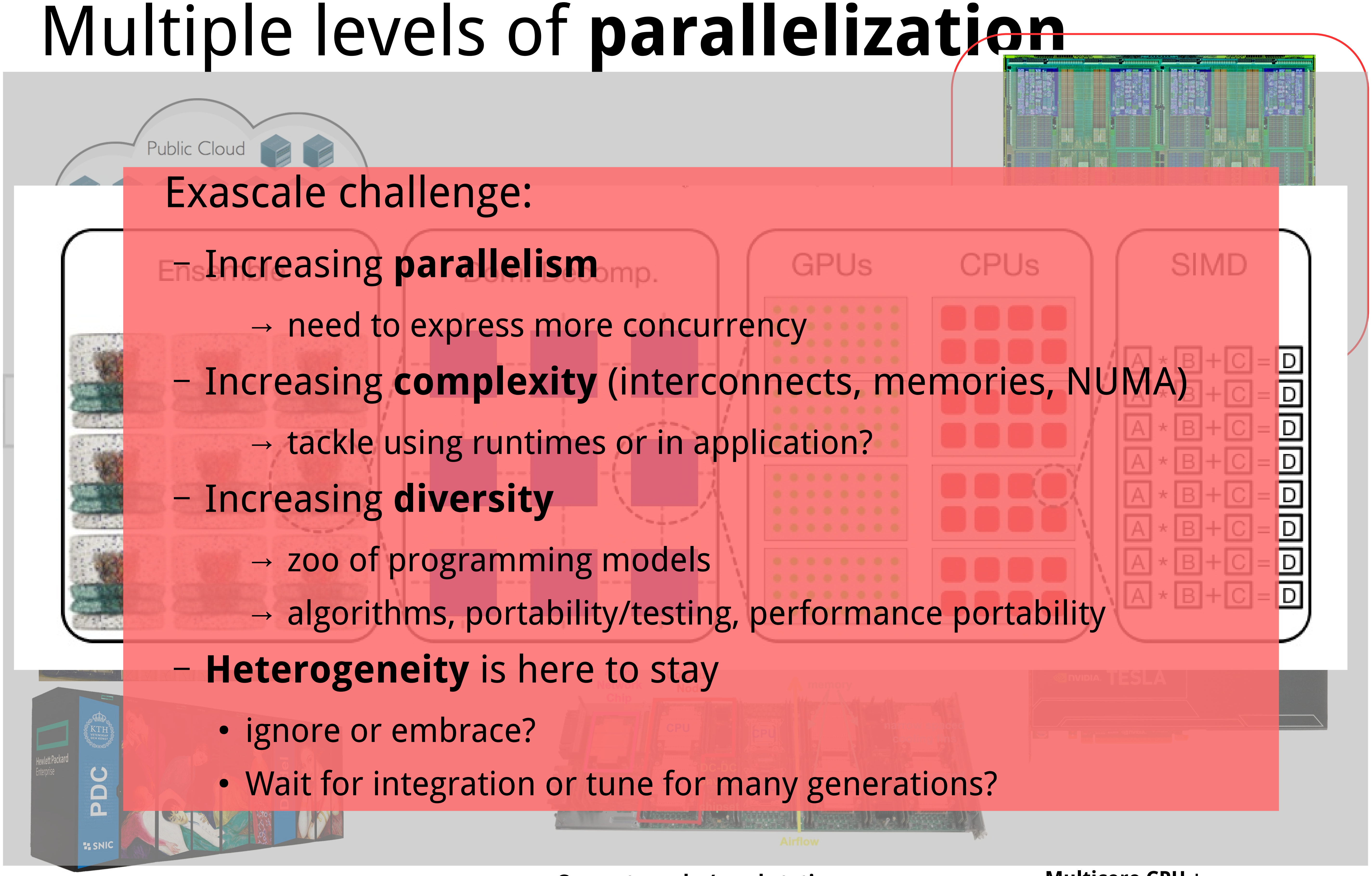
AMD Exascale architecture: LUMI, Frontier, Dardel



Intel Exascale architecture: Aurora



Multiple levels of **hardware parallelism**
Multiple levels of **parallelization**



Exascale challenge:

- Increasing **parallelism**
  - → need to express more concurrency
- Increasing **complexity** (interconnects, memories, NUMA)
  - → tackle using runtimes or in application?
- Increasing **diversity**
  - → zoo of programming models
  - → algorithms, portability/testing, performance portability
- **Heterogeneity** is here to stay
  - ignore or embrace?
  - Wait for integration or tune for many generations?

Compute cluster or cloud
Networked computers:
topology, bandwidth, latency

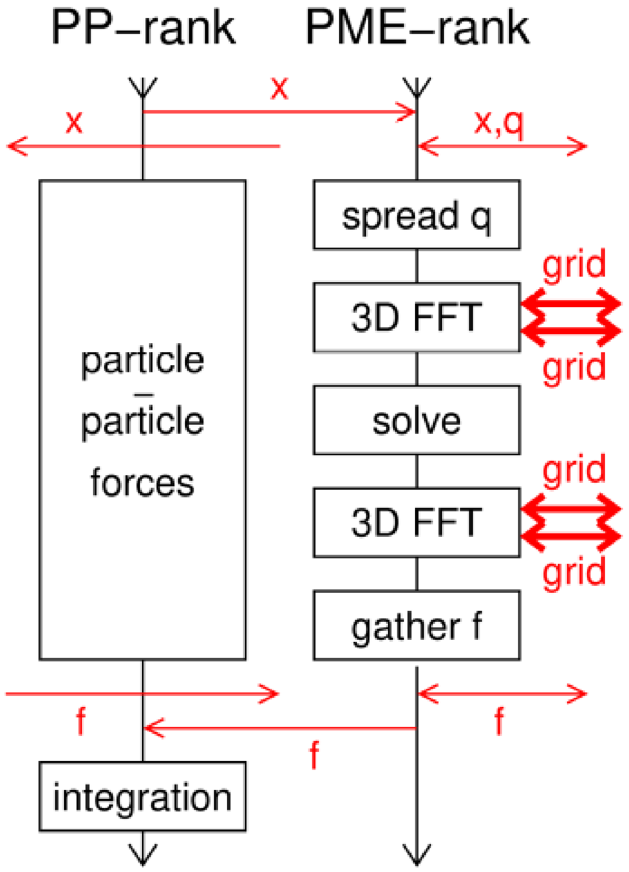
Compute node / workstation

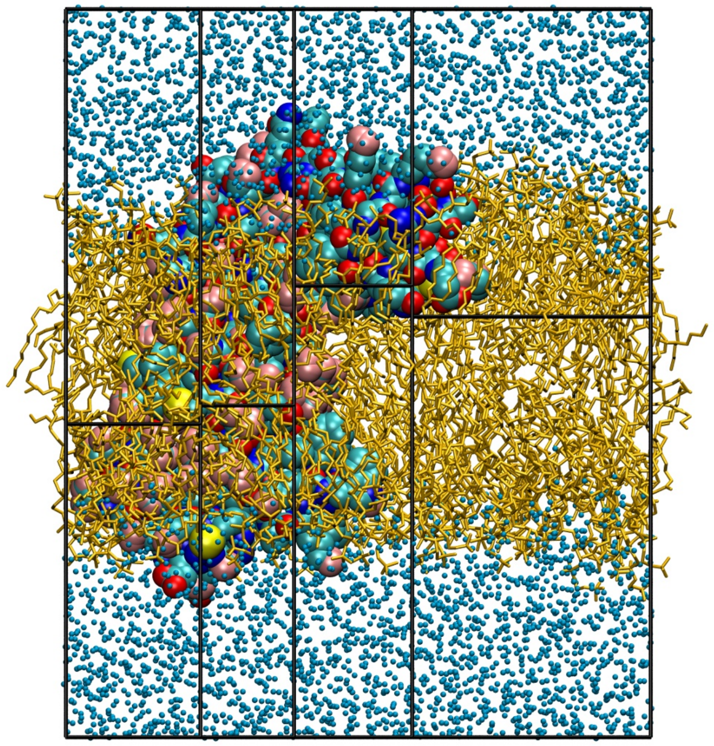
NUMA topology, PCIe

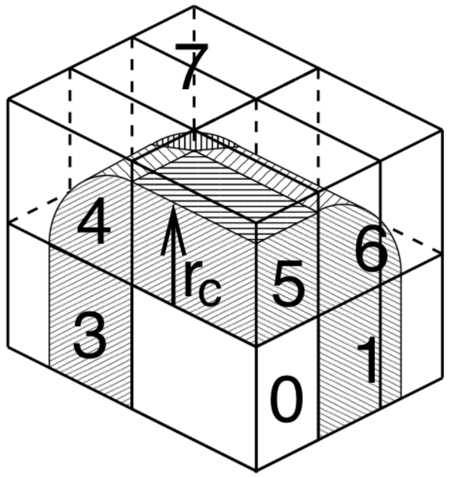
Multicore CPU +
manycore GPU
caches, interconnects

GROMACS parallelization

- **Multi-level hierarchical** parallelization: target each level of hardware parallelism individually
  - Intra-node:
    - OpenMP multi-threading
      - static loop schedule, cache optimized work decomposition layout, sparse reductions
    - SIMD C++ library abstraction:
      - 14 flavors supported
    - GPU abstraction layer
      - CUDA, OpenCL, SYCL
    - thread-MPI: pthreads-based MPI for ease of use
  - Inter-node:
    - MPI: SPMD / MPMD
    - Dynamic load balancing, task balancing







Why codesign?

- interdisciplinarity challenge

  → many hard problems need cross-disciplinary solutions

- MD: need for performance
  - GROMACS: design focus

- portability
  - GROMACS design focus

- hardware evolution...

GROMACS & codesign

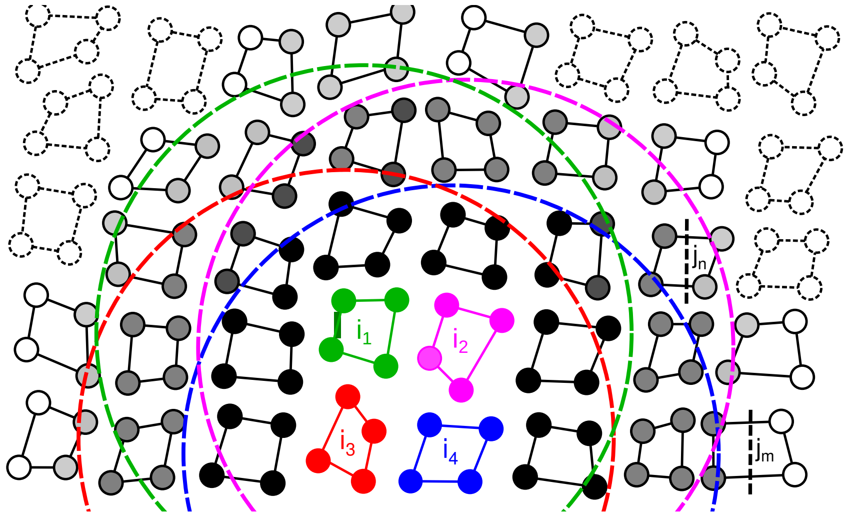
- Petascale → Exascale
  - required algorithm & parallelization redesign
  - Codesign has been & remains core component

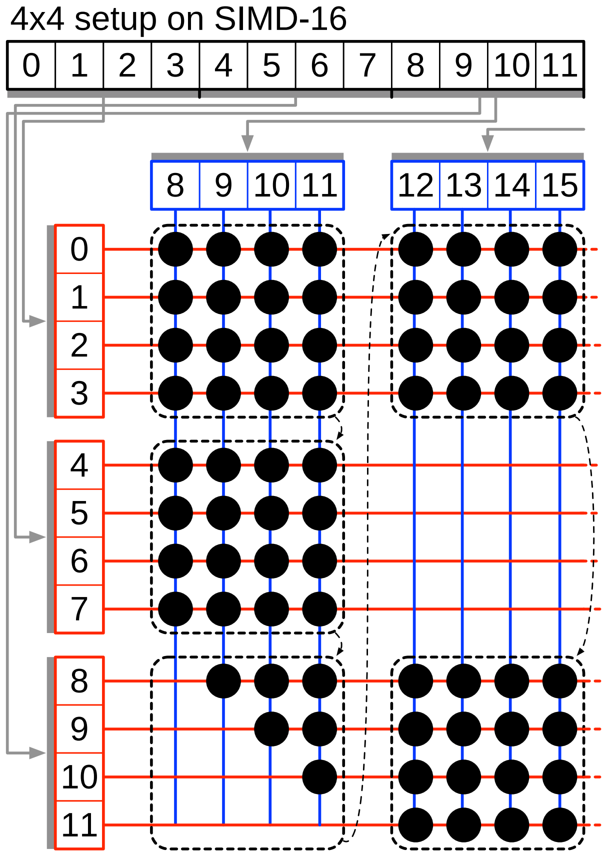
- Physics / math + algorithms + HW
  - mainly intra-team/community
  - innovate (reformulate algorithms, accuracy-based algorithms)
  - enable (domain experts method dev, CS experts micro-bench / port)

- Algorithms + HW + vendors / CS-experts
  - mainly inter-team collaboration
  - **align goals** for collaboration so **benefits both ways**!
  - Long-term: many steps forward and several major successes

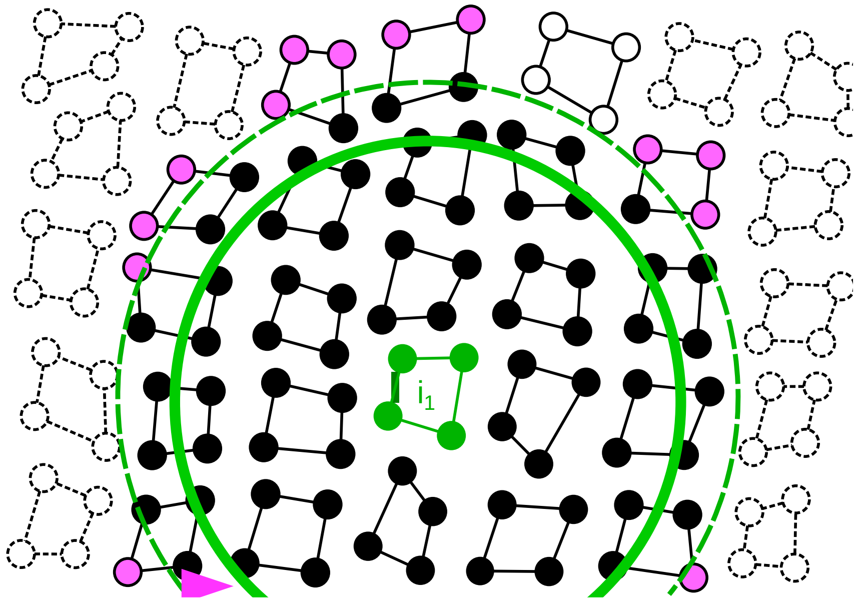
Algorithm redesign for modern architectures

Cluster pair-interaction algorithm for SIMD/SIMT

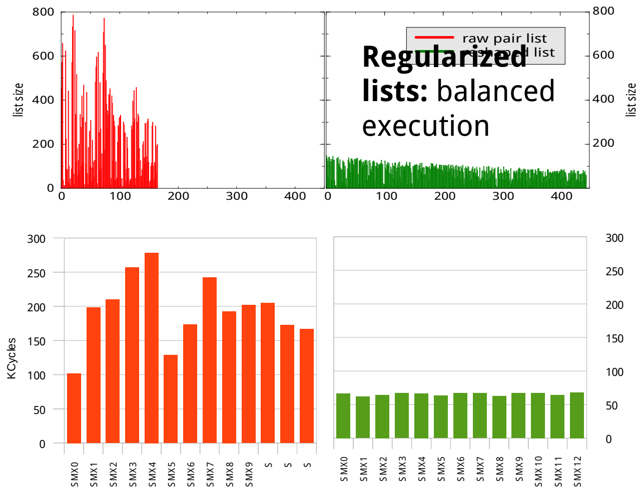


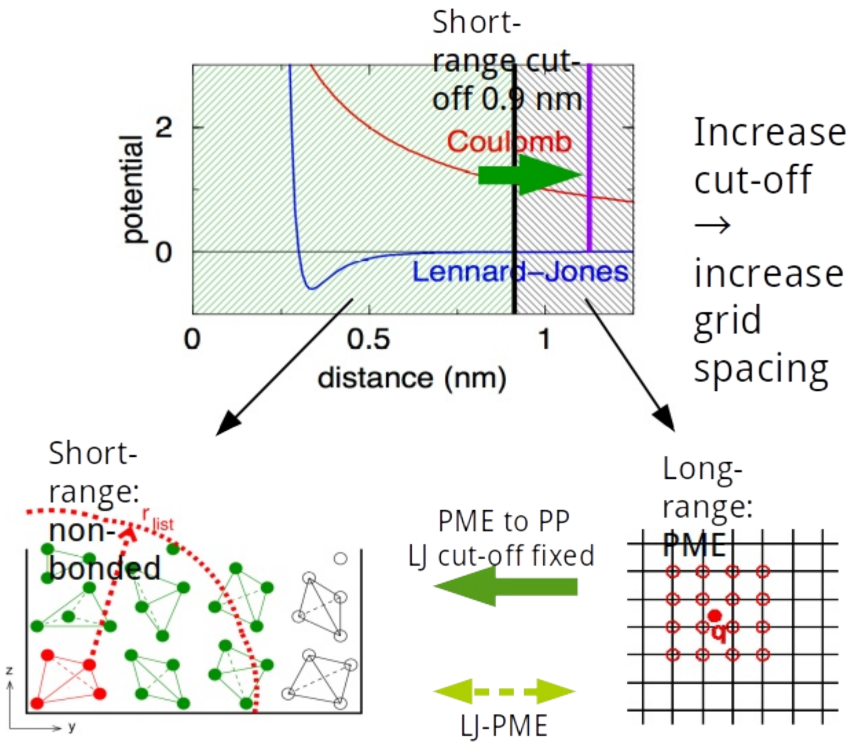


Accuracy-based automated list buffer improves SIMD algorithm parallel efficiency

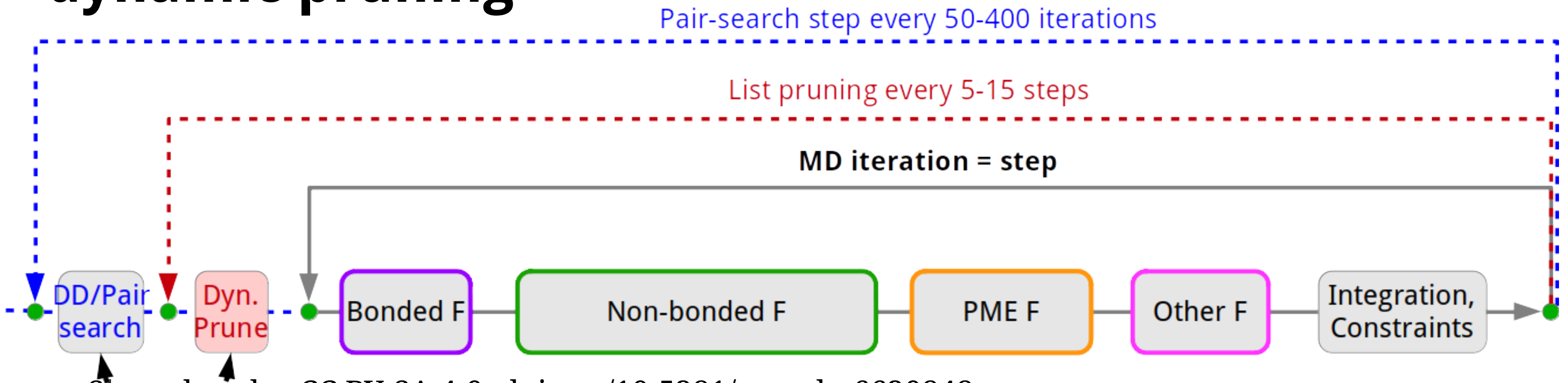


Multi-level heterogeneous data and task load-balancing: intra-GPU, intra-node, inter-node





Dual pair list with dynamic pruning

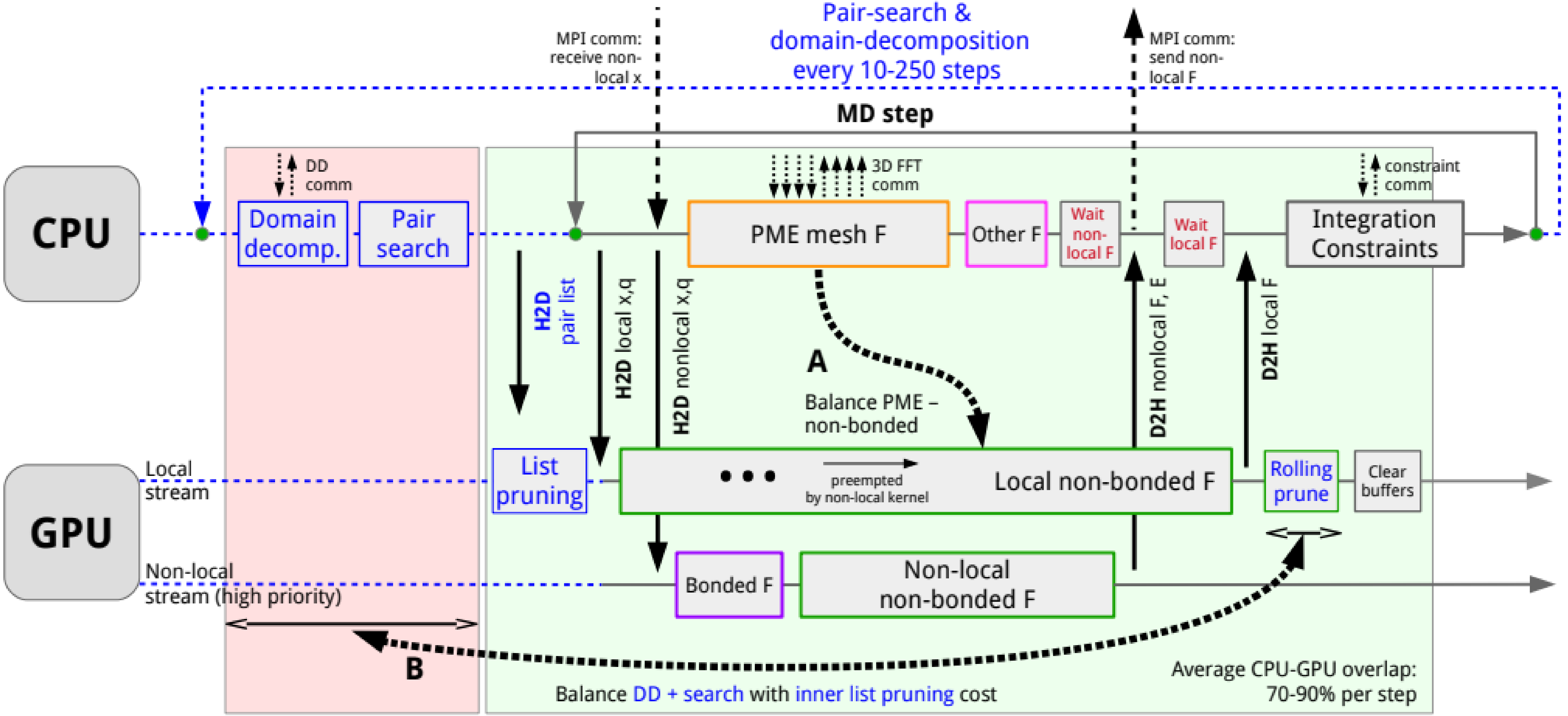


Embracing heterogeneity

- Heterogeneous design at the core:
  - "somewhat" complex schedule.

    → "But there is also always some reason in madness."

    - Heterogeneity for performance & **flexibility:** think of the (sometimes) silent codesign partners, method devs

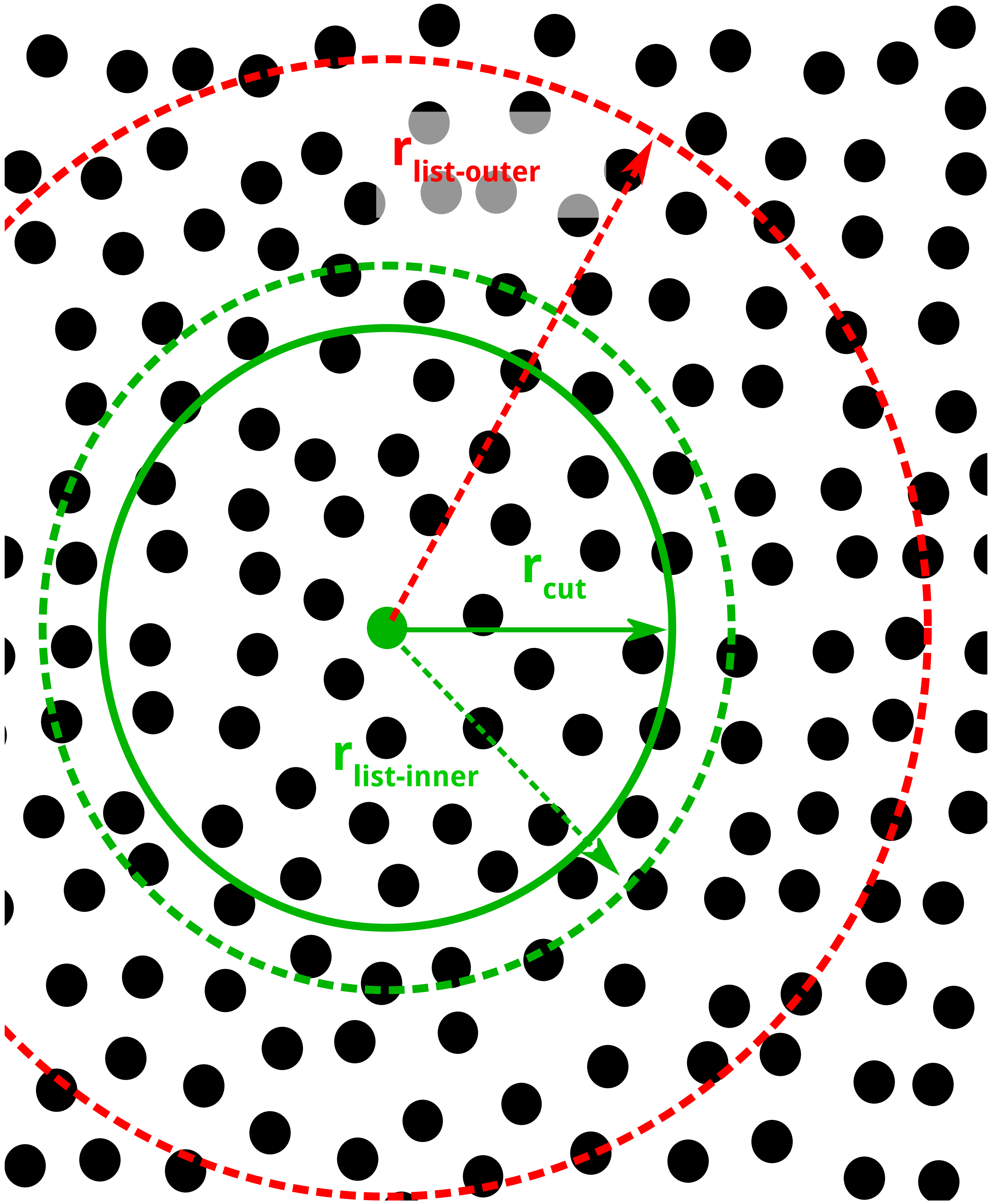


Dual pair list

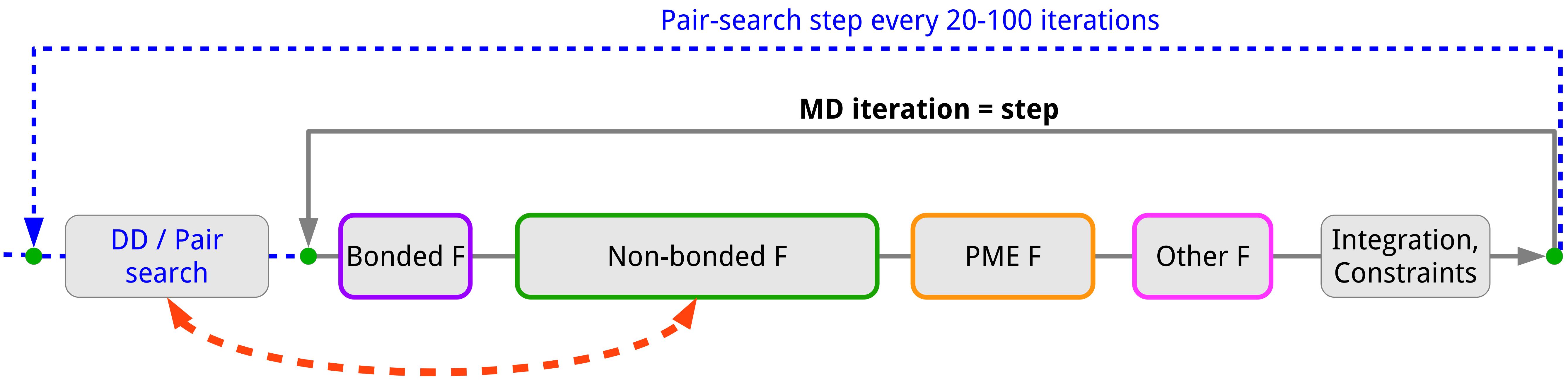
- Trading costly data regularization for force computation not ideal!
- Instead: keep regularized particle data longer, **shift the cost trade-off**
- Use two buffers and lists:

  outer / inner
- Periodically re-prune

  outer → inner
- List lifetime / search frequency:
  - outer list less frequently (costly)
  - inner list more frequently (cheap)



Accuracy-based balancing: **dual pair list** reducing decomposition & search cost



Pair-search step every 20-100 iterations

MD iteration = step

DD / Pair search

Bonded F

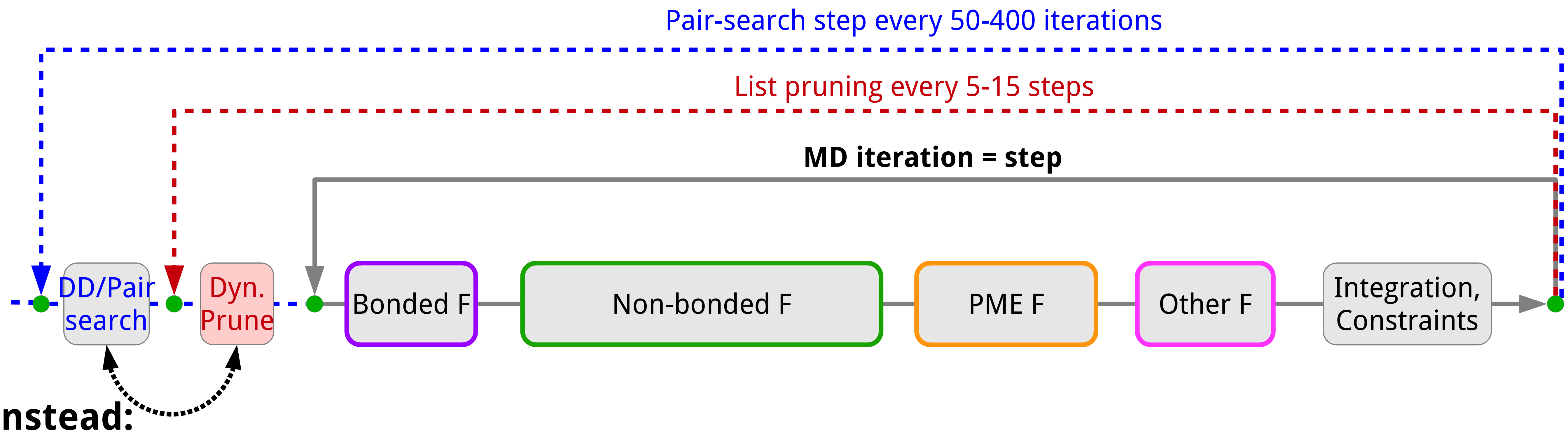
Non-bonded F

PME F

Other F

Integration, Constraints

Trade search/DD cost → non-bonded pair interaction cost



Pair-search step every 50-400 iterations

List pruning every 5-15 steps

MD iteration = step

DD/Pair search

Dyn. Prune

Bonded F

Non-bonded F

PME F

Other F

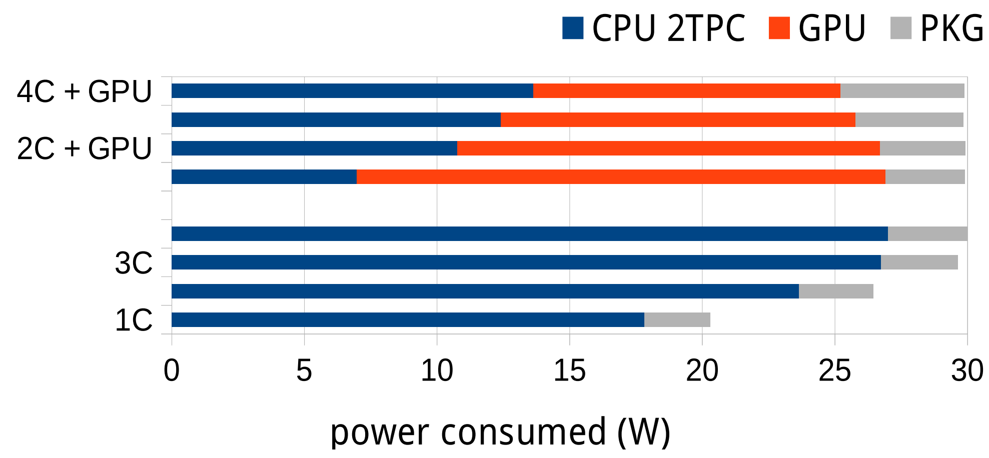
Integration, Constraints

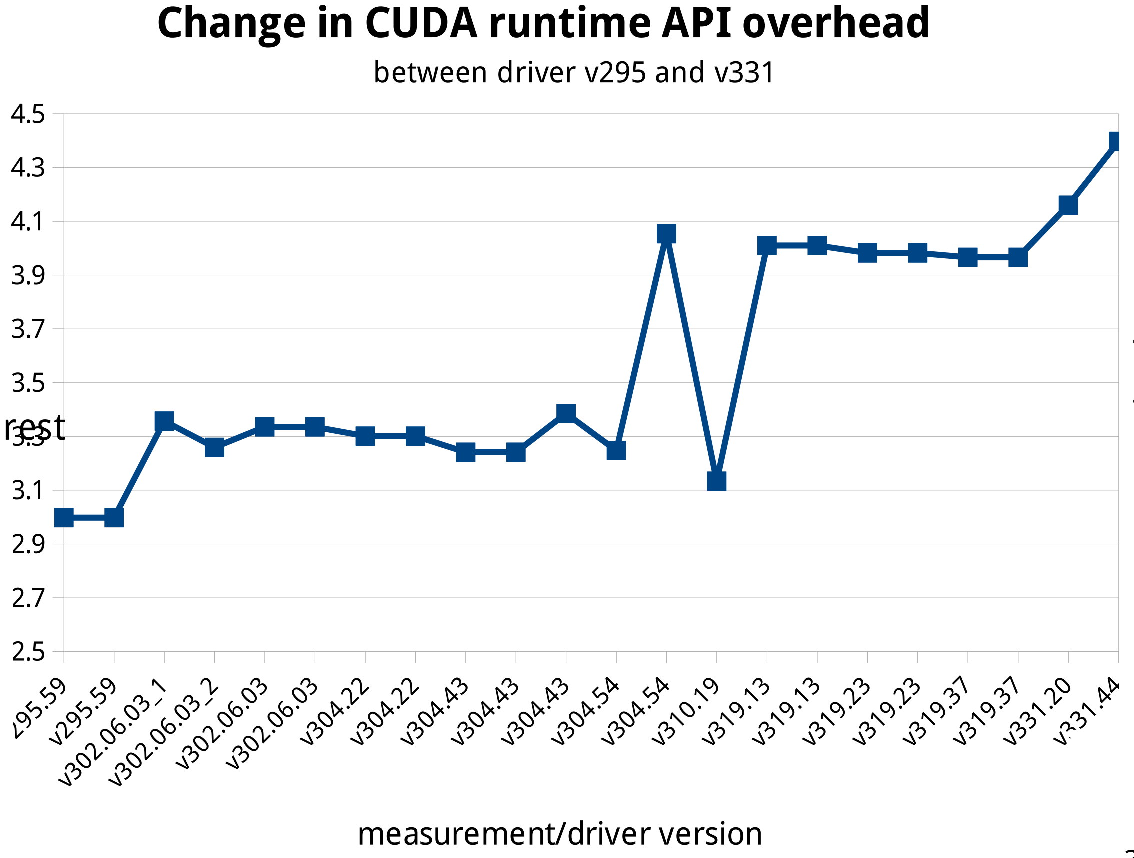
Instead:

Trade search/DD cost → dynamic re-pruning

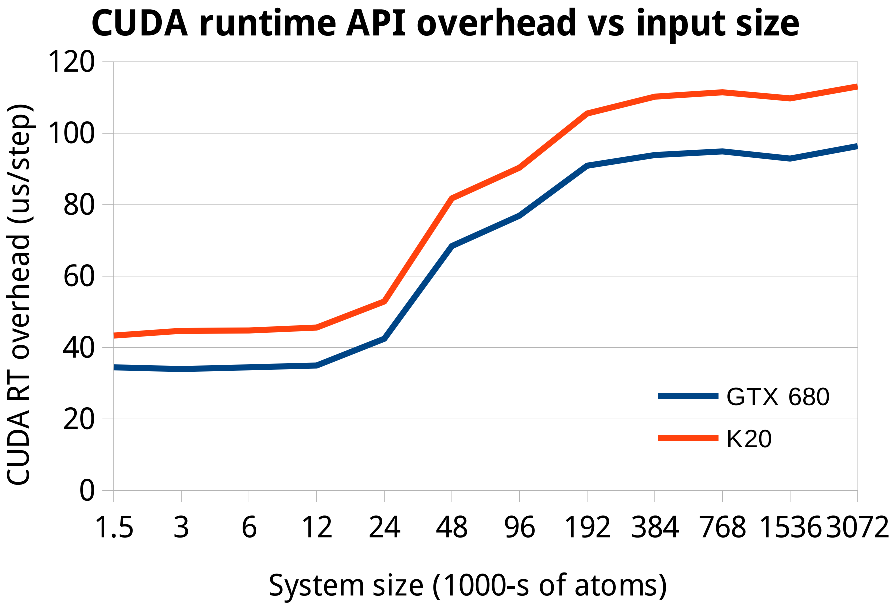
Keeping the non-bonded pair interaction cost (near) constant

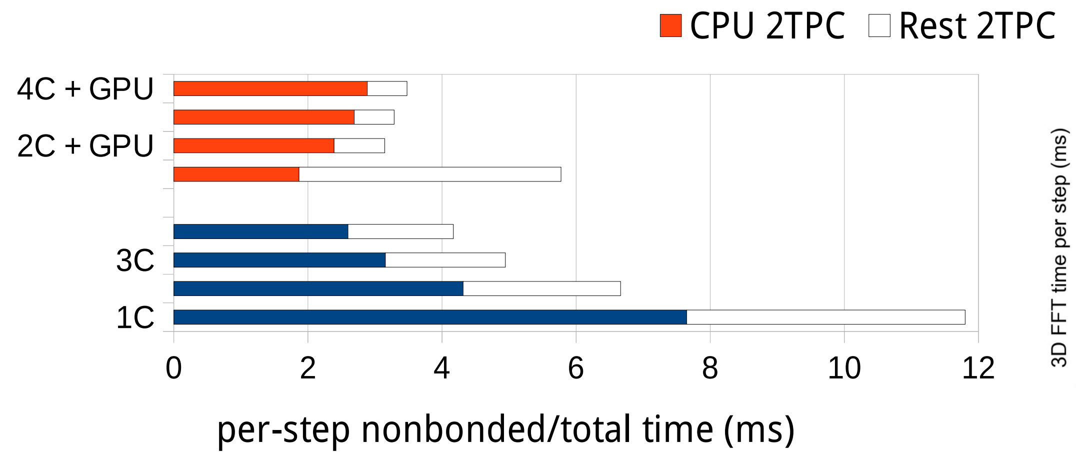
Vendor-collaboration codesign: long-term practice

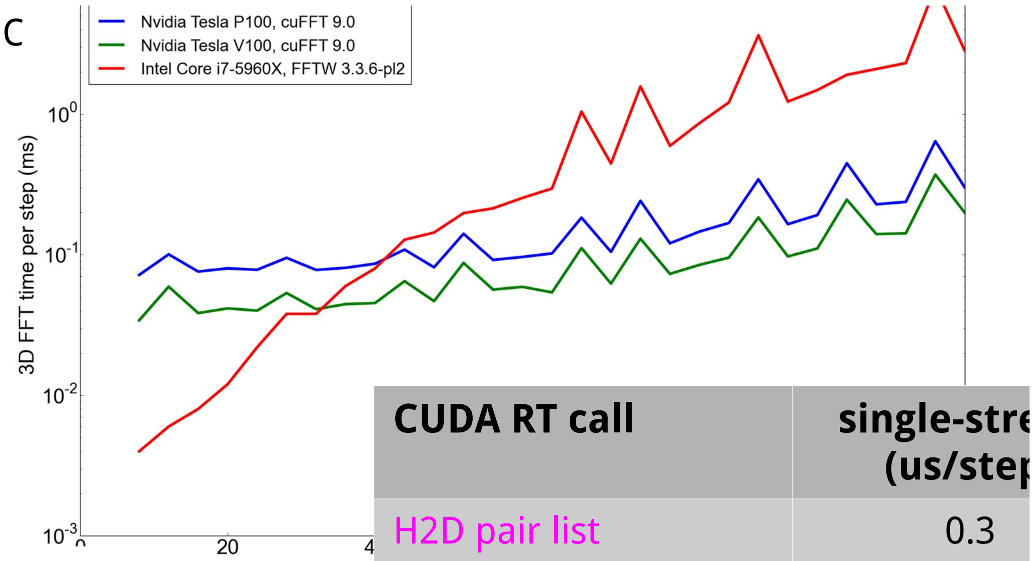


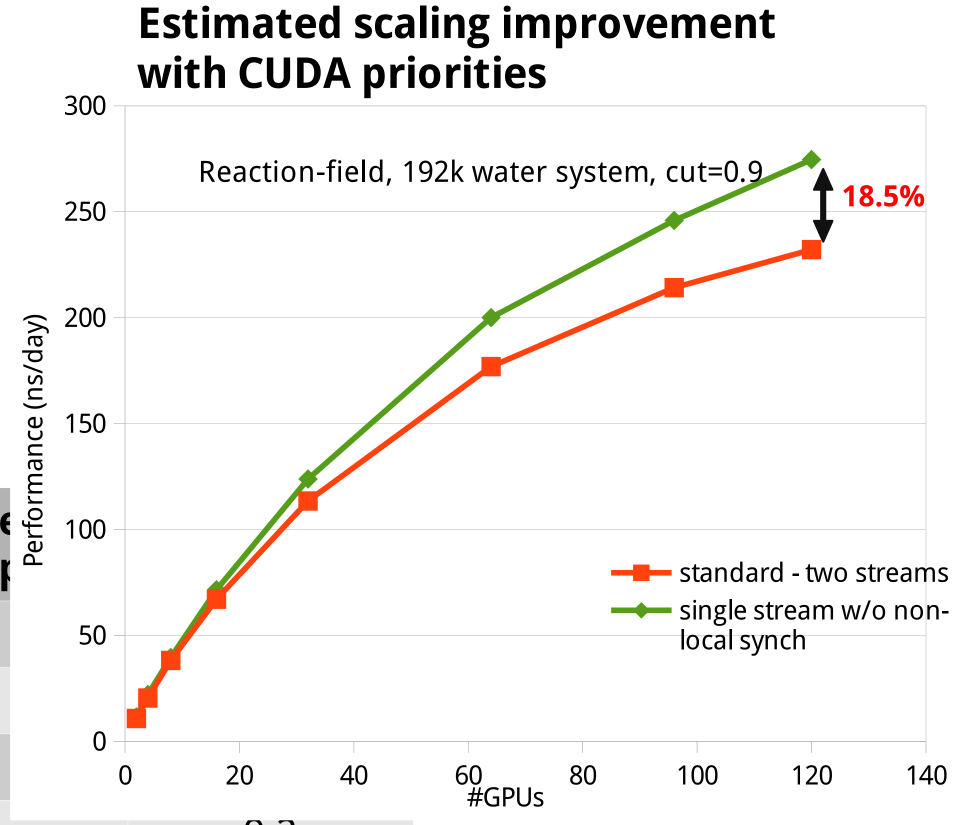

Change in CUDA runtime API overhead

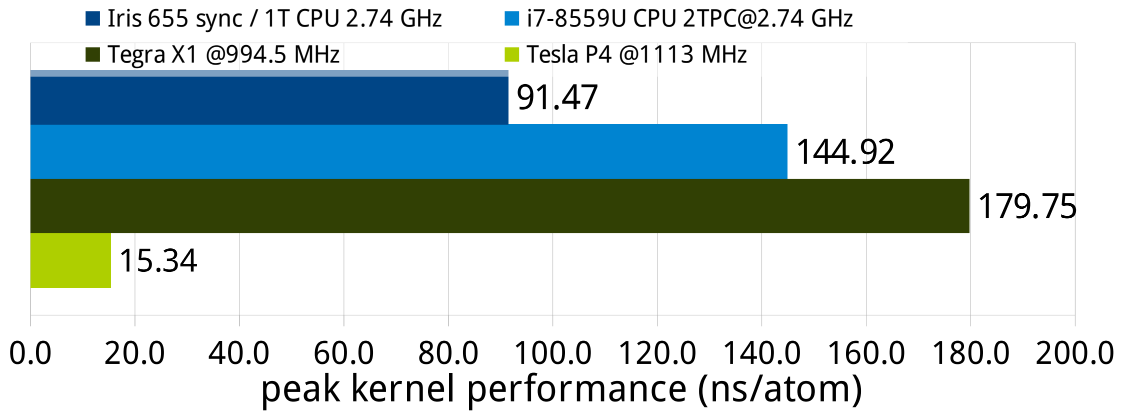
between driver v295 and v331


CUDA runtime API overhead vs input size






Estimated scaling improvement with CUDA priorities



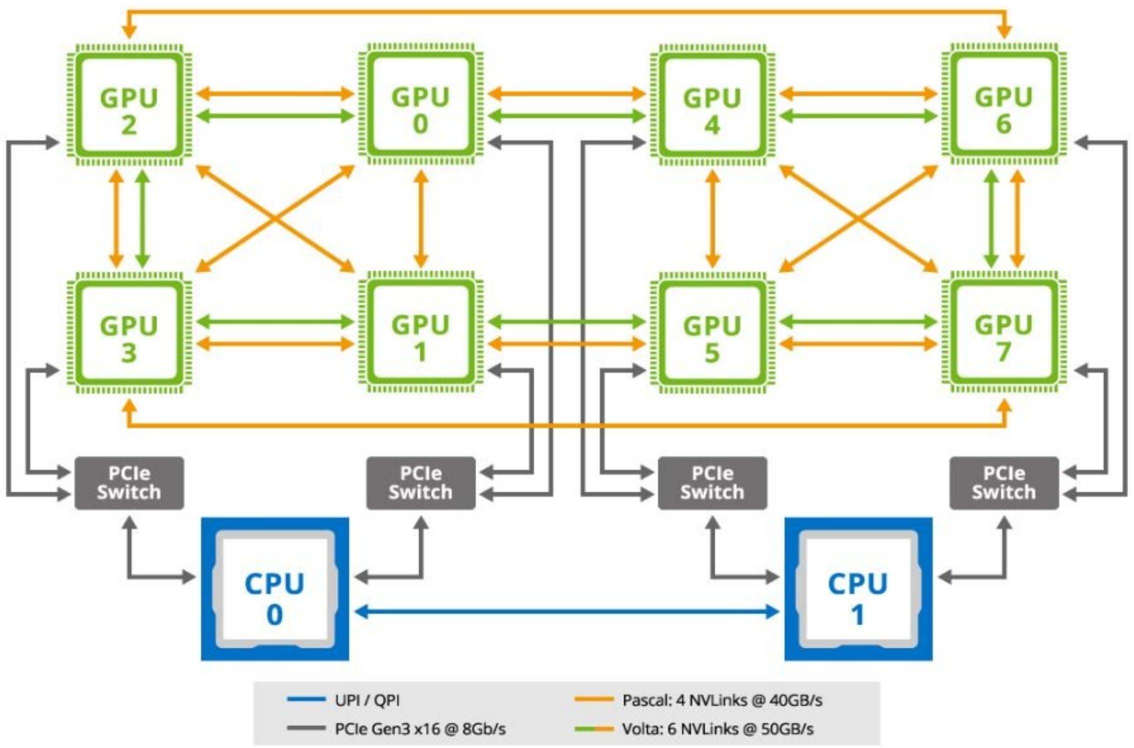
CUDA RT call	single-stre (us/step	
H2D pair list	0.3	
H2D x+q	7.2	
NB kernel	9.8	
D2H forces	5.2	9.2
D2H E+shift F	1.4	1.4
cudaStreamSync	3.4	2.3
Clear kernel	9.2	8.8
Total	**37.2**	**49.5**

Codesign project with NVIDIA

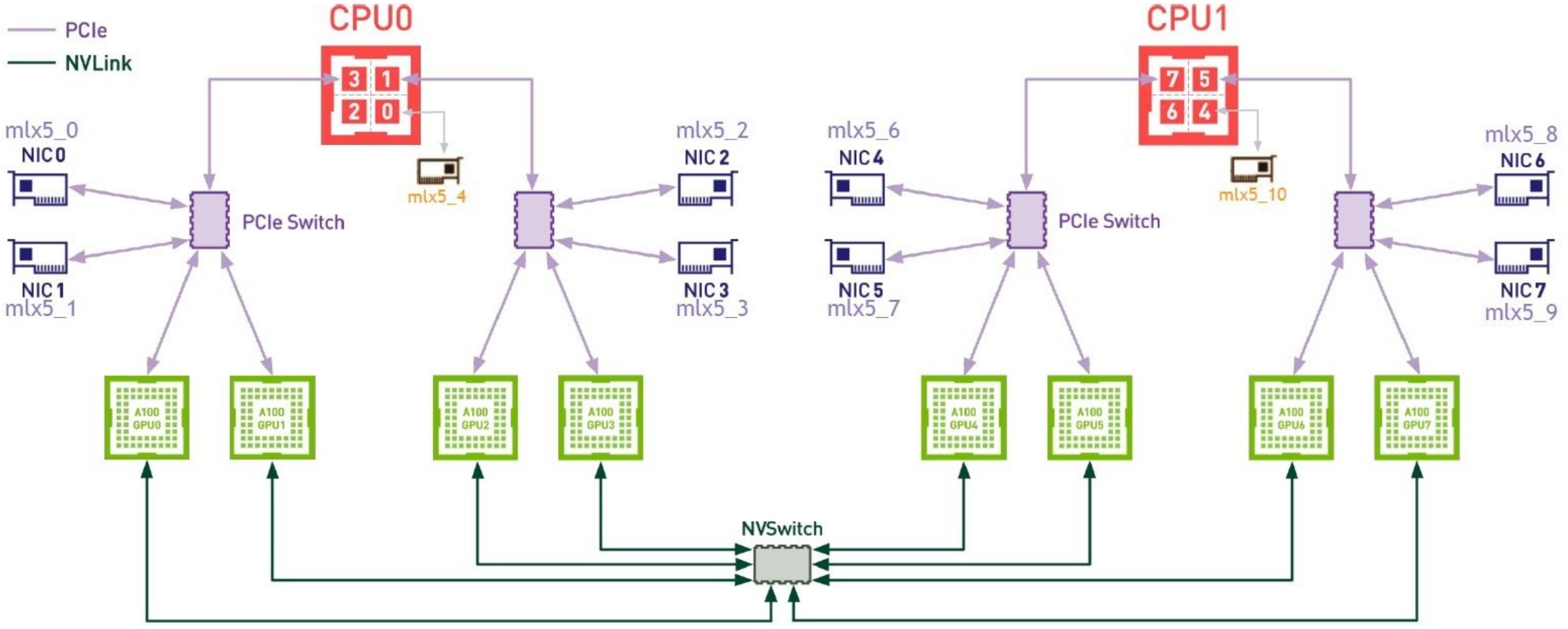
Direct GPU communication

- Alan Gray & Gaurav Garg (NVIDIA)
- Goal:
  - avoid CPU staging, accelerate critical path
  - target intra-node interconnects, e.g. NVLInk
- Two flavors:
  - thread-MPI: single-node (since 2021)
    - P2P copies (put/get), exchange CUDA events allows remote sync
    - Single process + multiple GPUs: bottlenecks required CUDA driver threading optimizations
  - CUDA-aware MPI: multi-node (since 2022)
    - requires host sync before issuing MPI call

DGX 1V

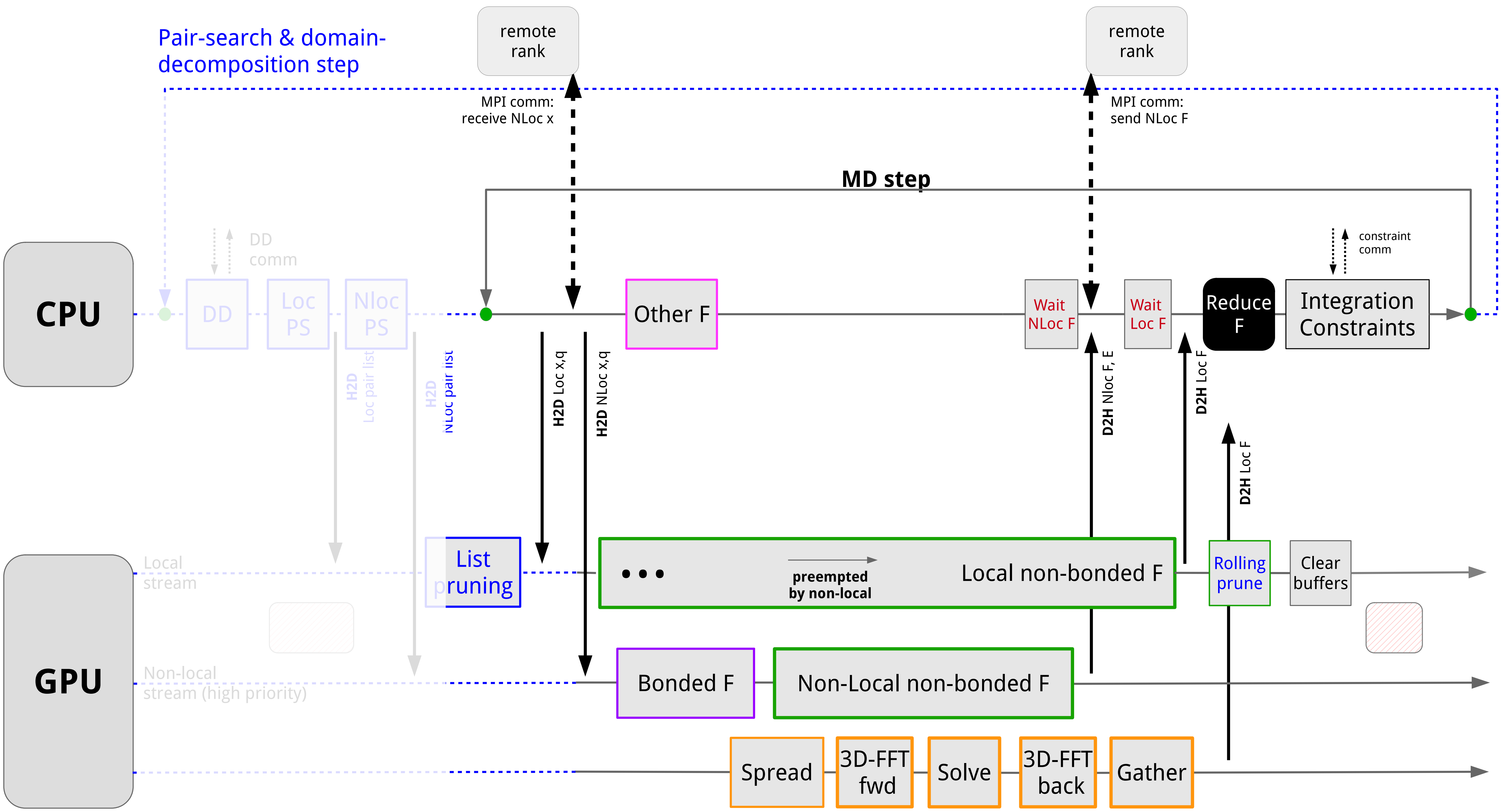


DGX A100

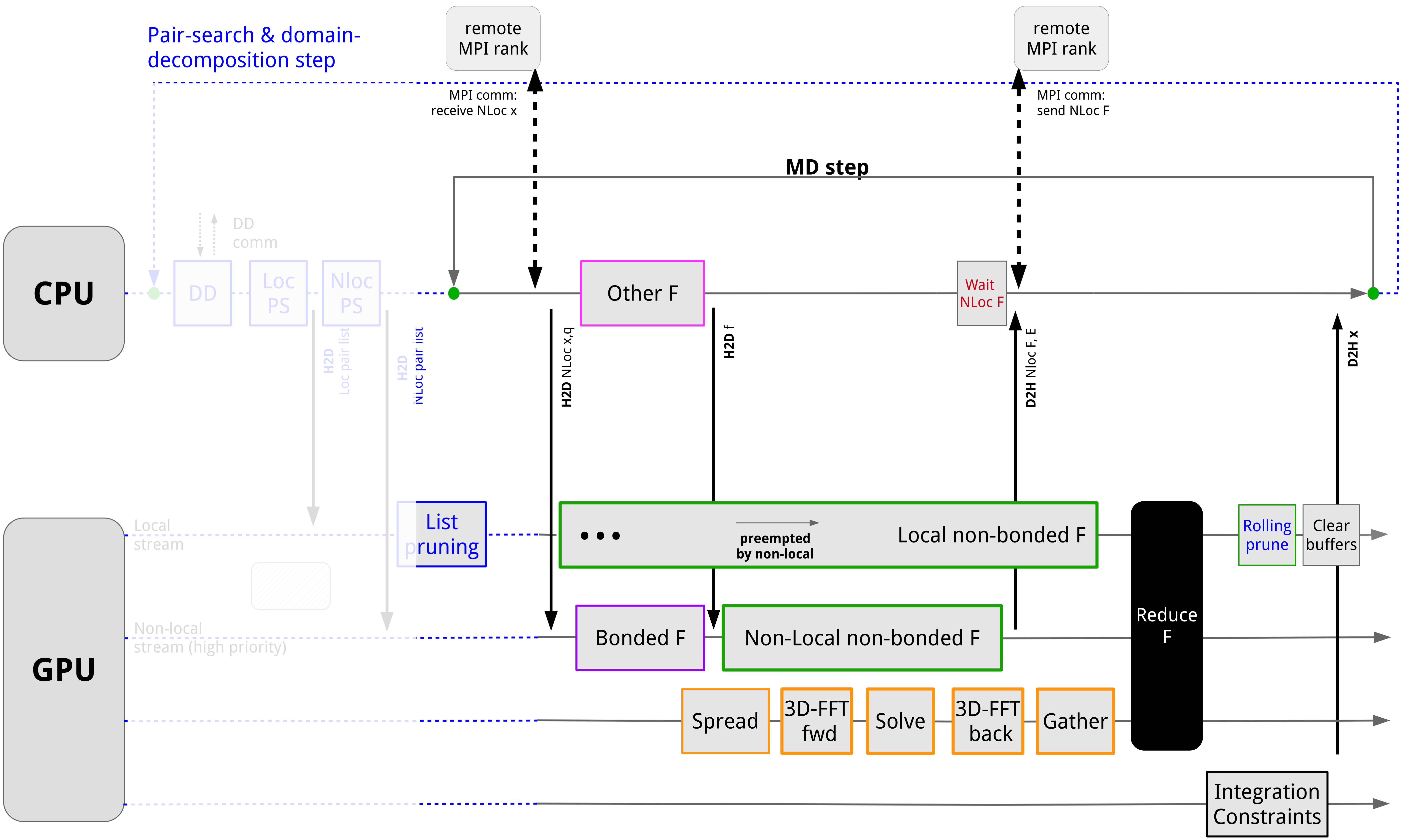


Shared under CC

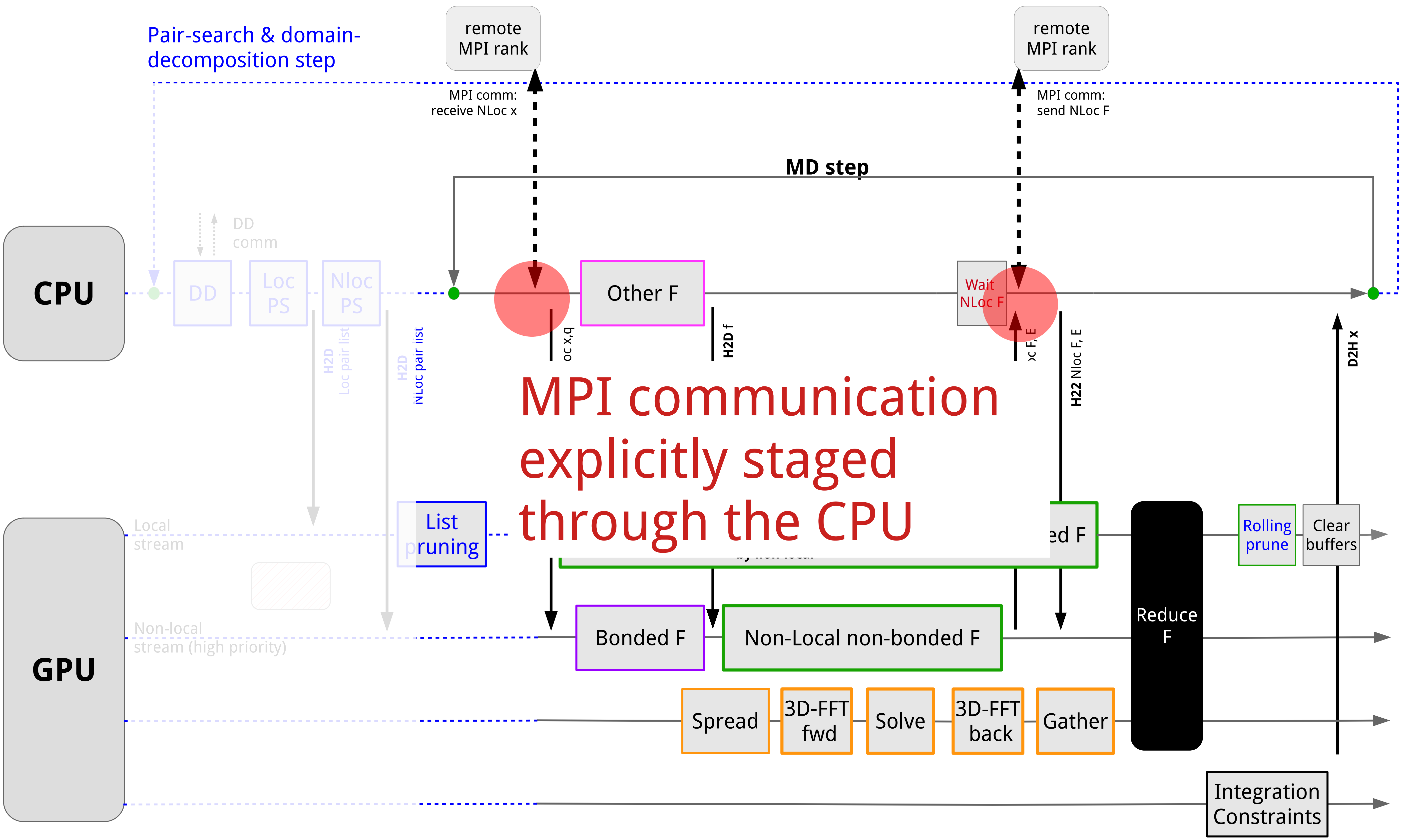
Multi-GPU/rank force offload scheme



Multi-GPU/rank GPU-resident scheme

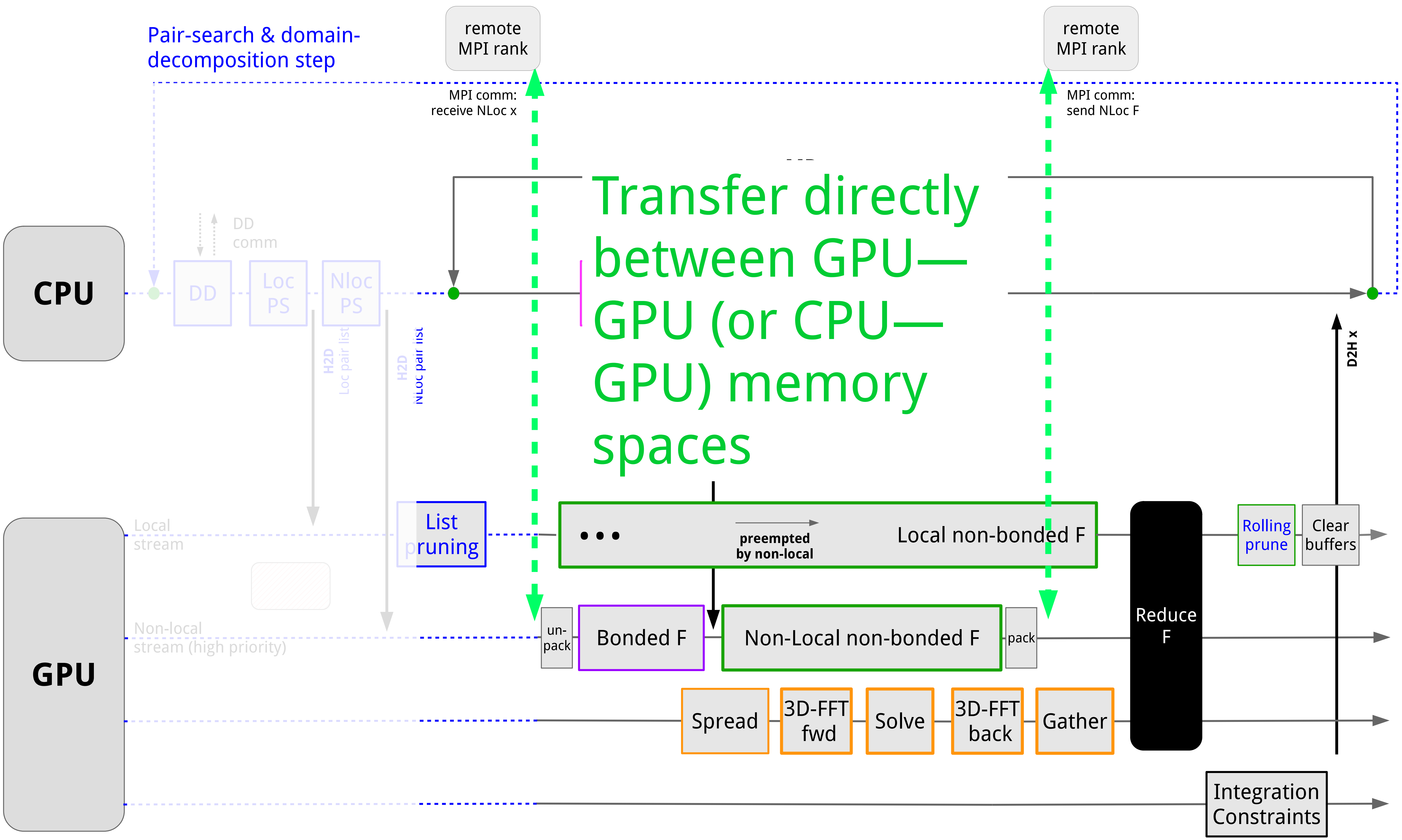


Multi-GPU/rank GPU-resident scheme



MPI communication explicitly staged through the CPU

Multi-node GPU resident & direct GPU communication

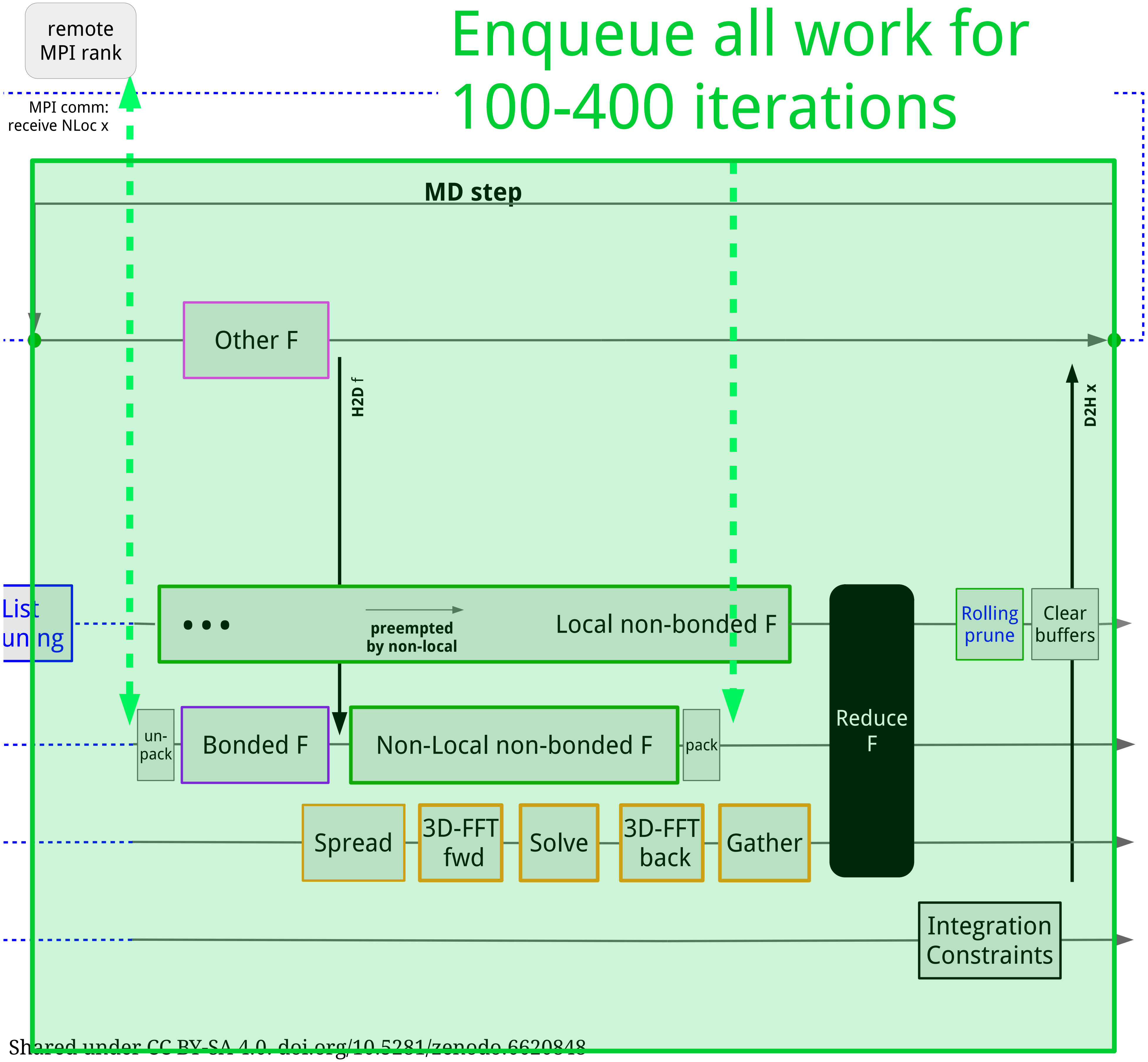


Transfer directly between GPU—GPU (or CPU—GPU) memory spaces

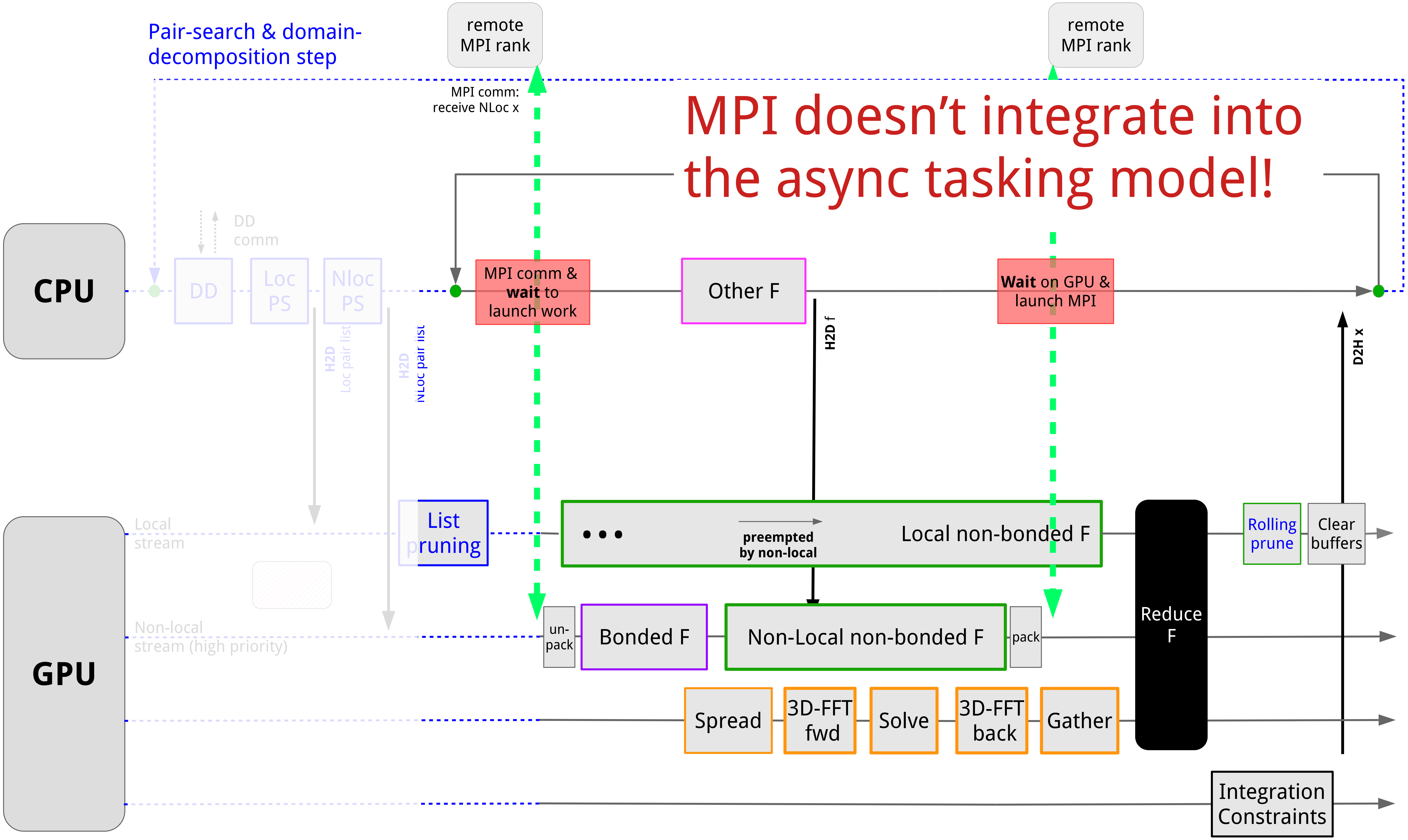
Multi-GPU resident step: single-node P2P direct GPU comm

- The entire inner loop including communication can be enqueued ahead of time
  - if there is no CPU task (Other F)
  - enables more efficient scheduling
  - overlap launch cost with work
  - CUDA graphs

- Challenges:
  - integrating CPU tasks
  - load balancing



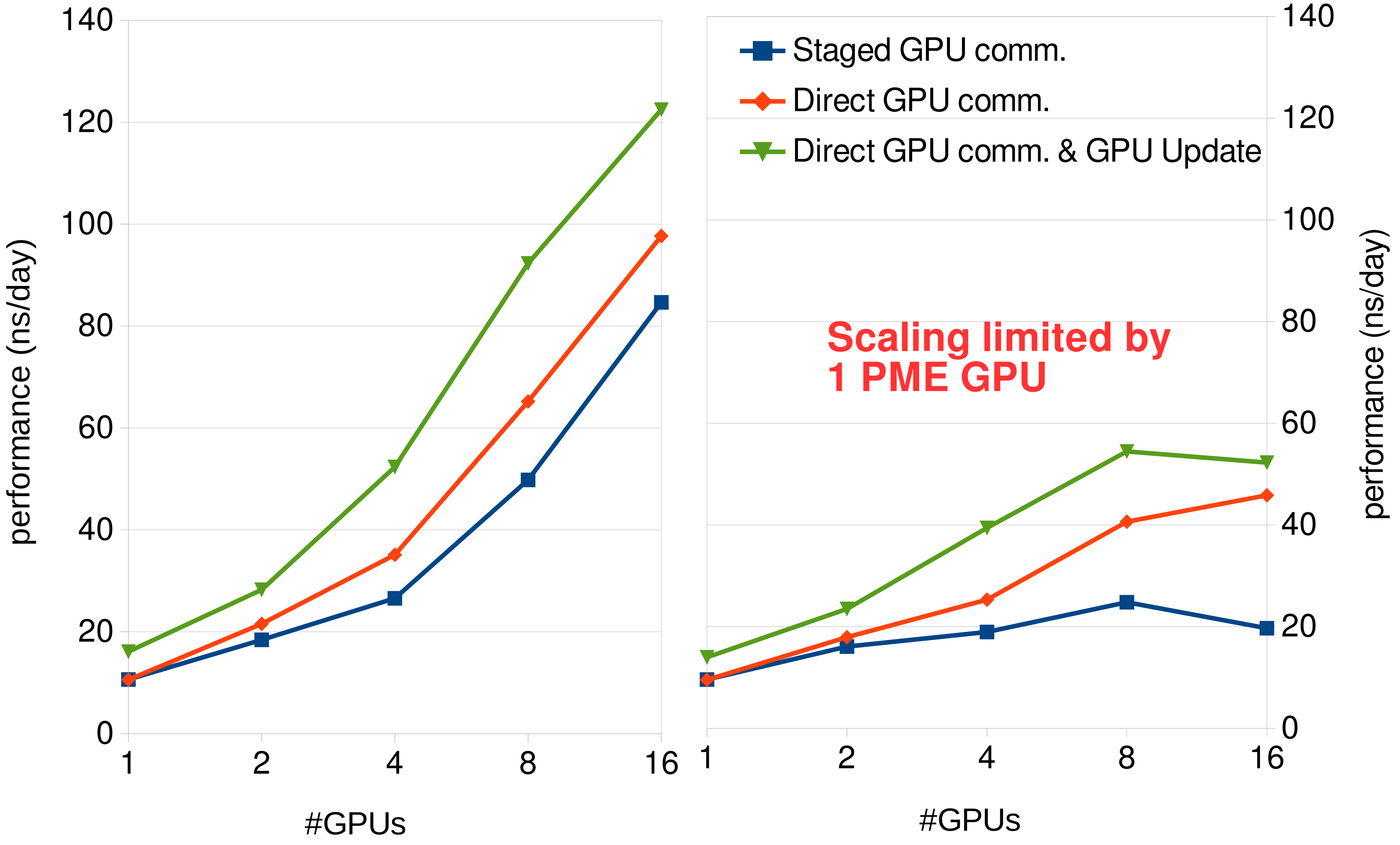
Multi-node GPU resident step & GPU-aware MPI comm



MPI doesn't integrate into the async tasking model!

Direct GPU communication performance

- Major benefit on fast interconnects with GPU-resident steps
- Modest improvements on low-end interconnects



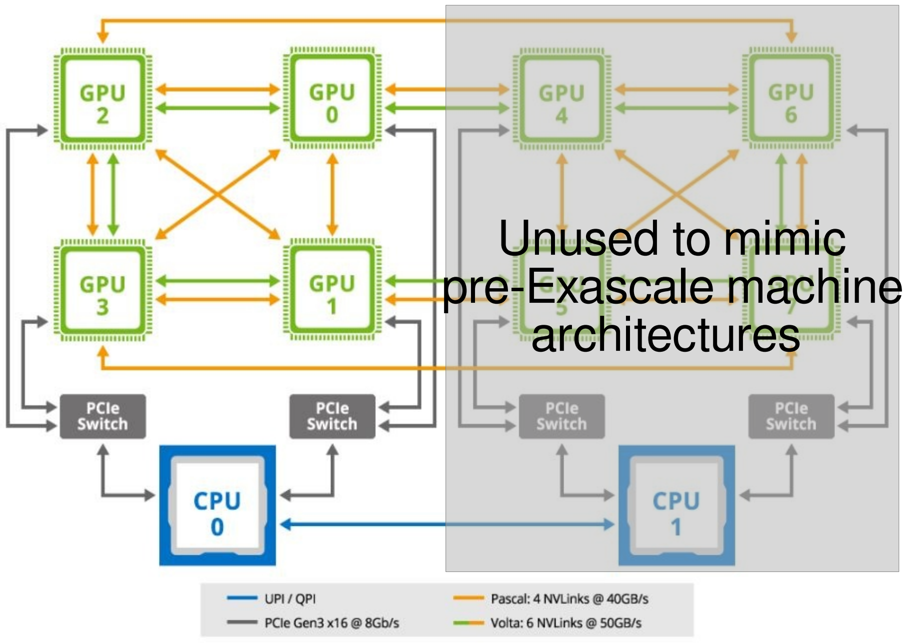
Reaction-field electrostatics: only halo-exchange

PME electrostatics: halo-exchange & PME MPMD communication

Scaling limited by 1 PME GPU

System: STMV 1M atoms

Hardware: DGX-1V

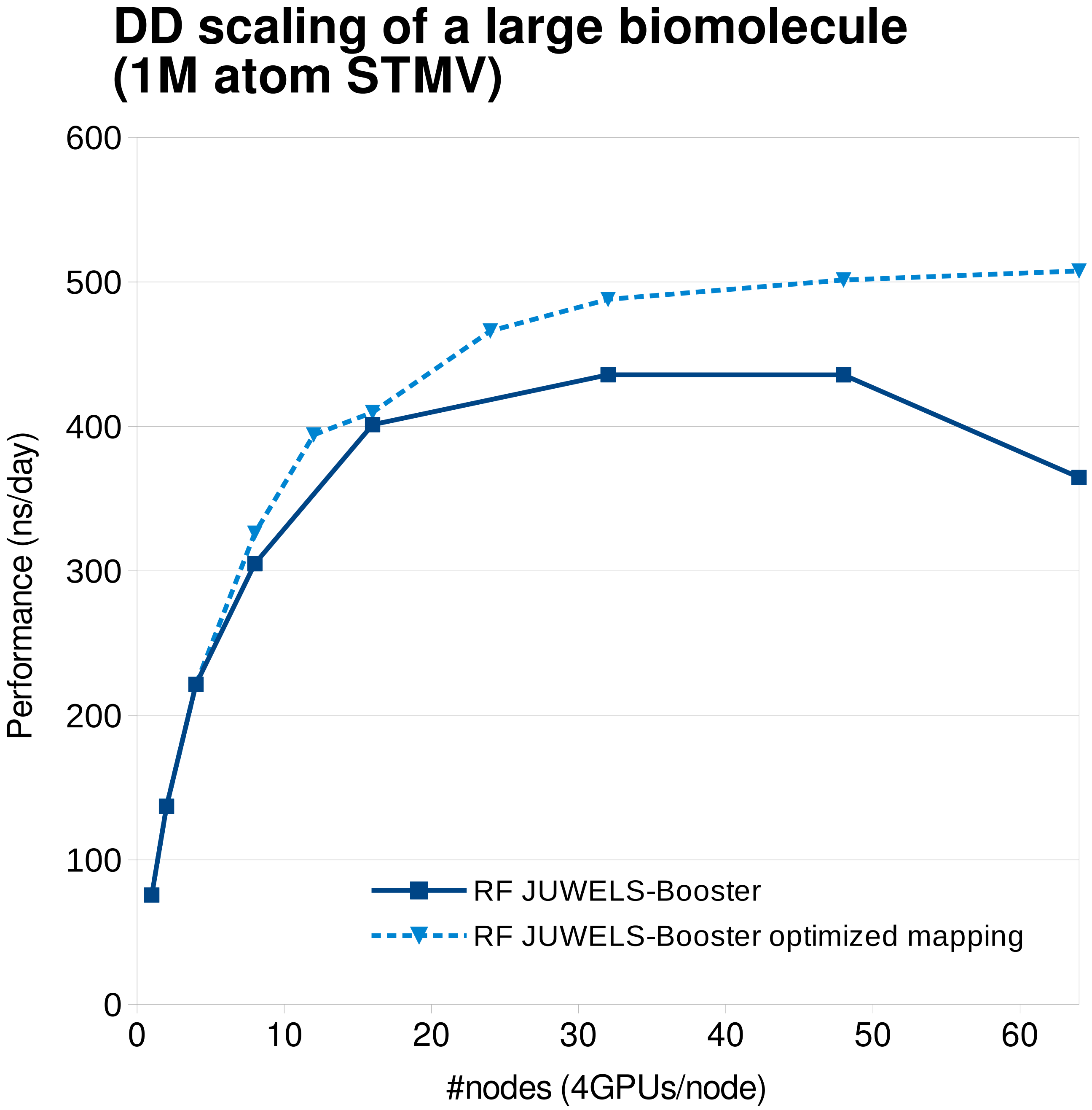


Unused to mimic pre-Exascale machine architectures

DD halo exchange peak strong scaling

- JUWELS-booster:
  - 2x24-core AMD EPYC Rome
  - 4xA100
- ~50% parallel efficiency up to 12 nodes
  - only ~20000 atoms/GPU
- Peak at 48-64 nodes:

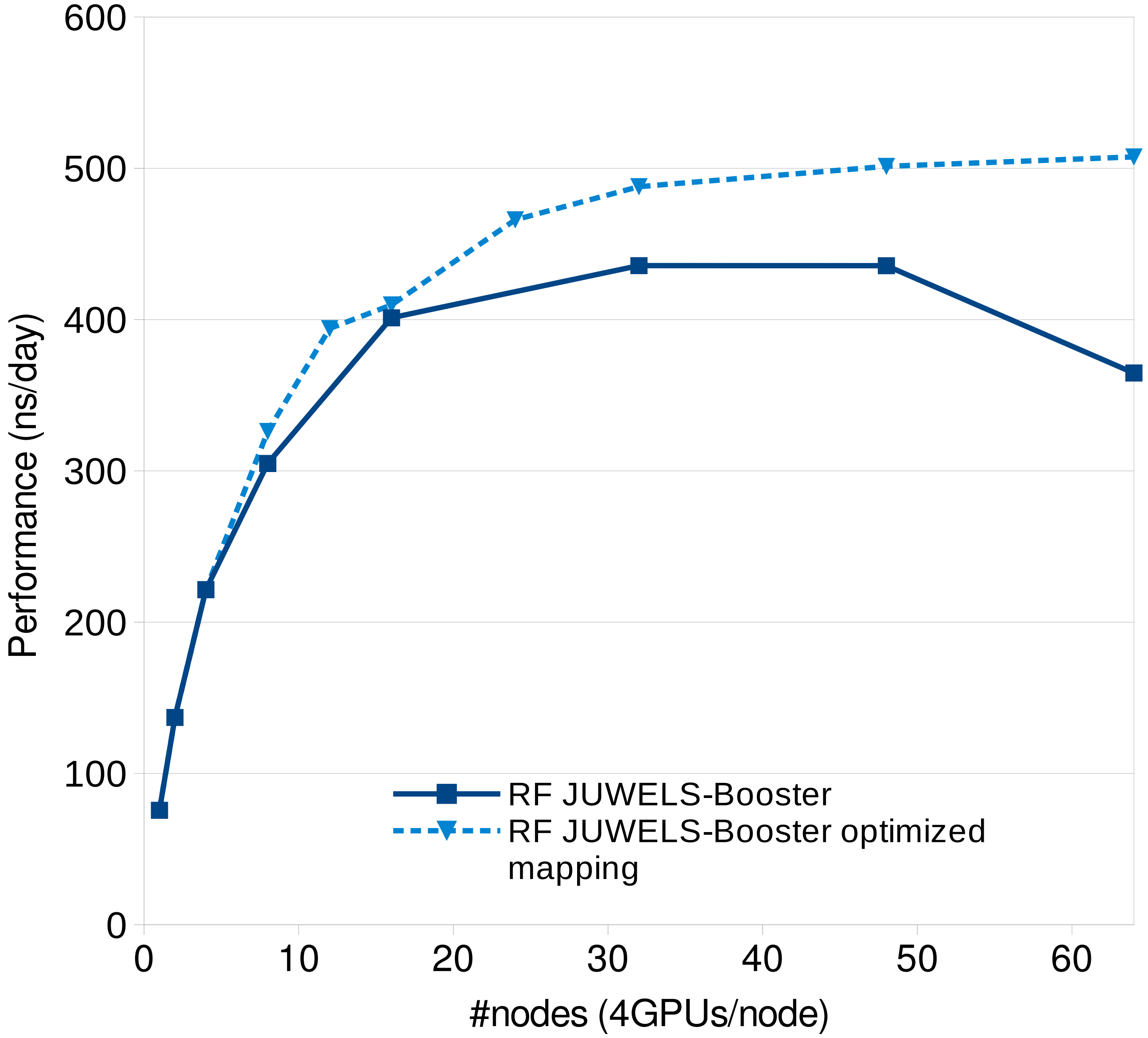
  >500 ns/day



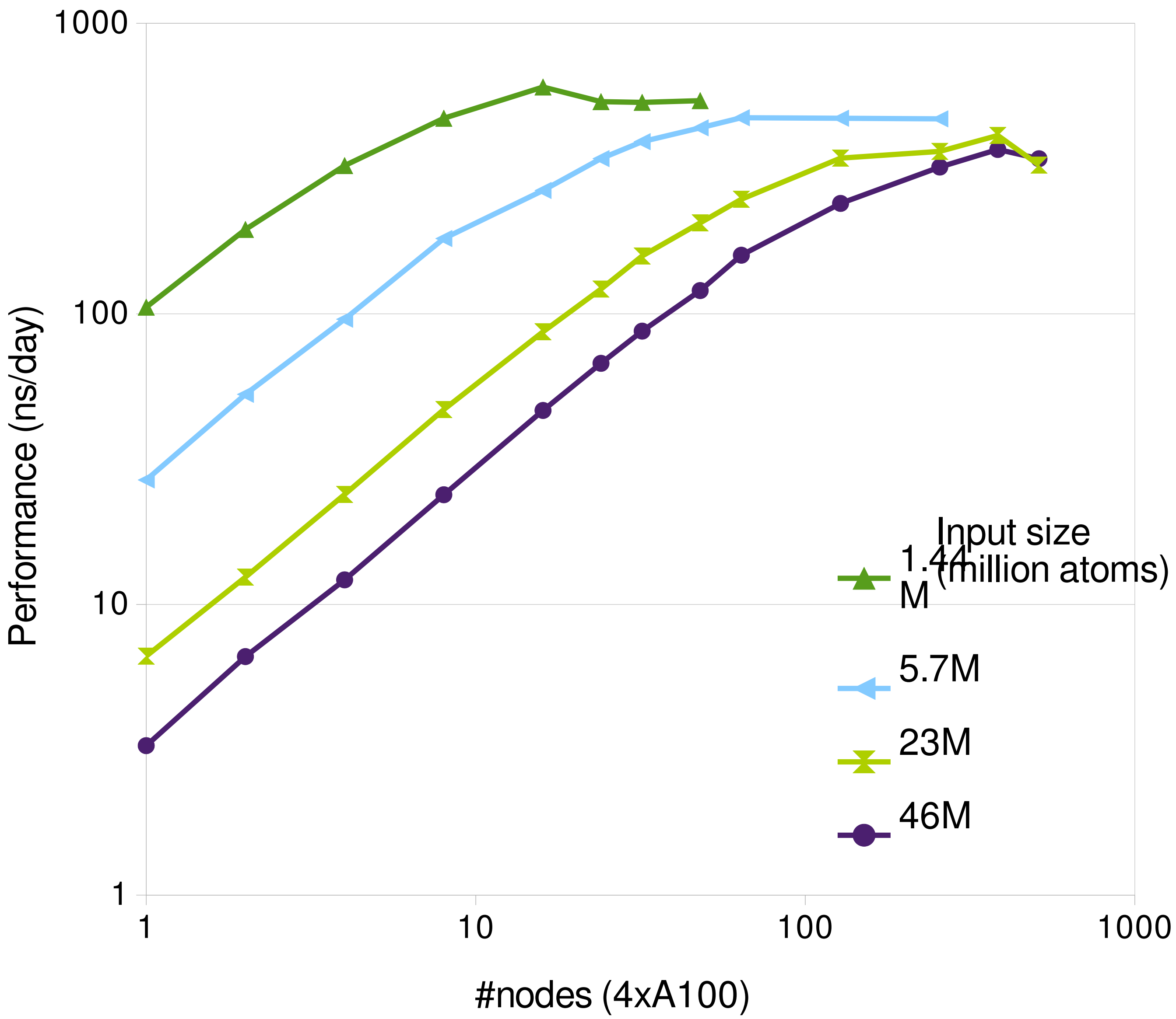
DD halo exchange peak strong scaling

- JUWELS-booster:
  - 2x24-core AMD EPYC Rome
  - 4xA100

DD scaling of a large biomolecule (1M atom STMV)



Scaling large homogeneous systems to ~400 GPUs

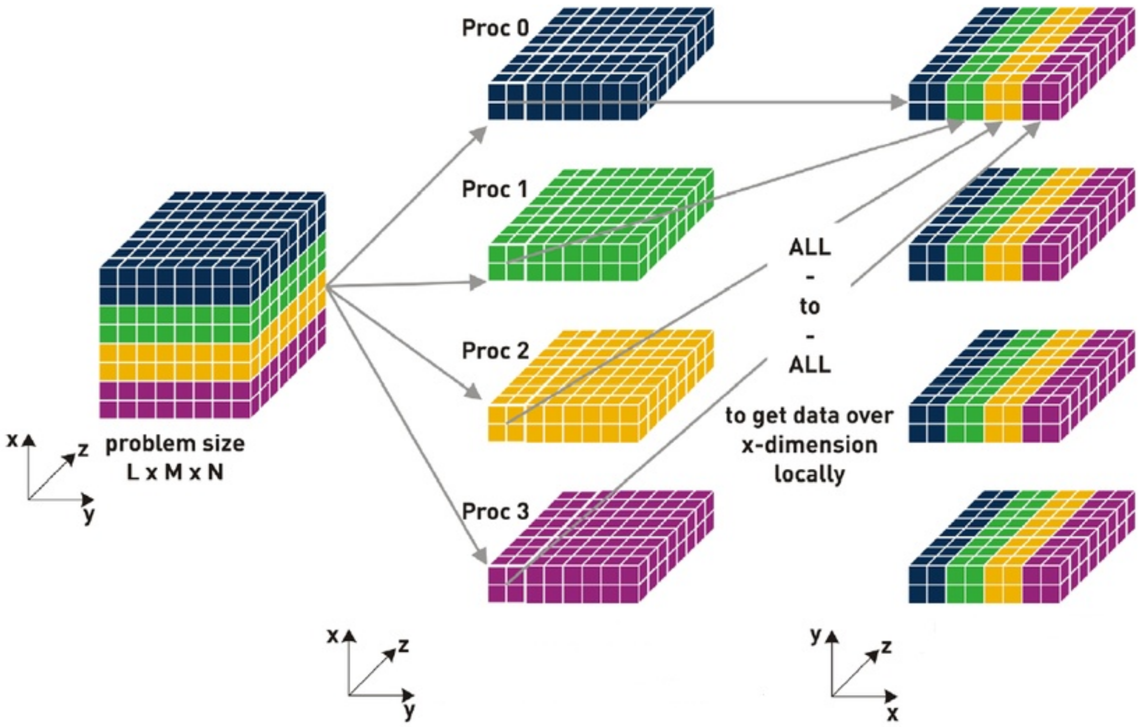


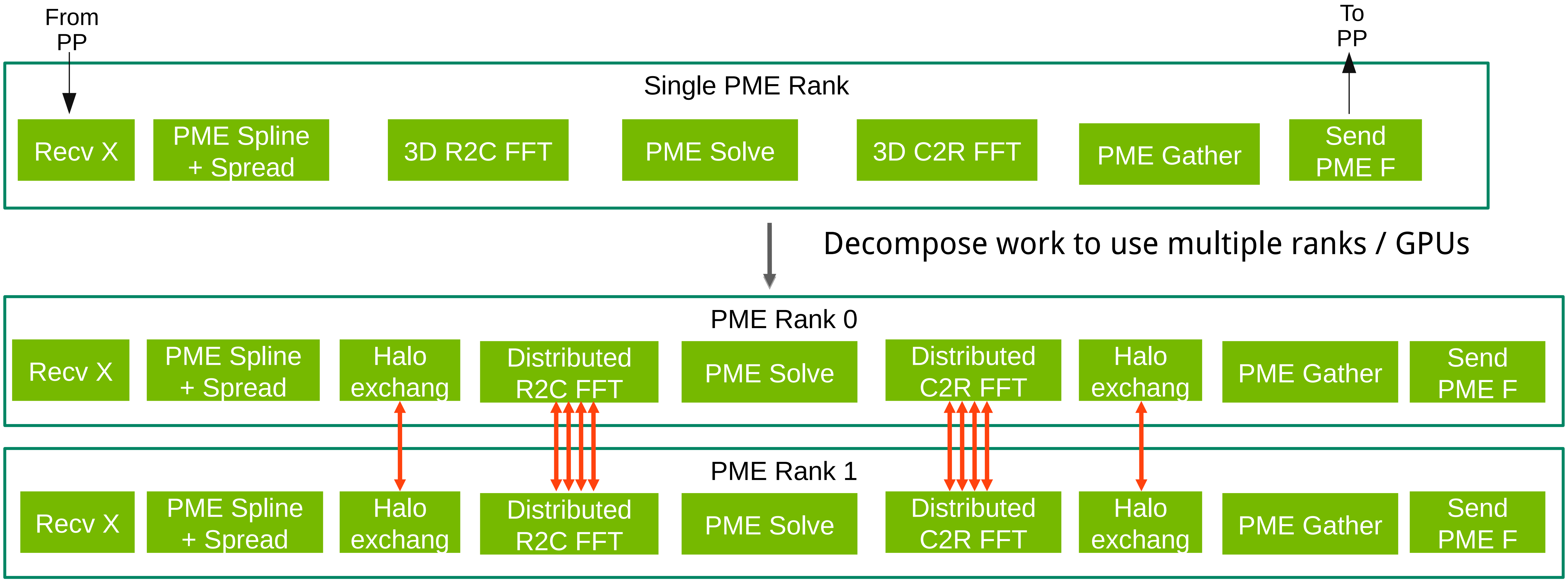
PME decomposition

- **GROMACS team + Gaurav Garg (NVIDIA)**
- remove the limitation of single dedicated PME GPU
- 3D FFTs strong-scaling challenge:

  typical size 32^3-256^3, hardly scale
- Released in 2022:

  Hybrid mode: FFT on CPU
- In development (upstreamed):
  - major algorithmic and parallelization optimizations
  - **HeFFT** and **cuFFTmp** for GPU-resident mode




From PP

Single PME Rank

Recv X → PME Spline + Spread → 3D R2C FFT → PME Solve → 3D C2R FFT → PME Gather → Send PME F

To PP

Decompose work to use multiple ranks / GPUs

PME Rank 0

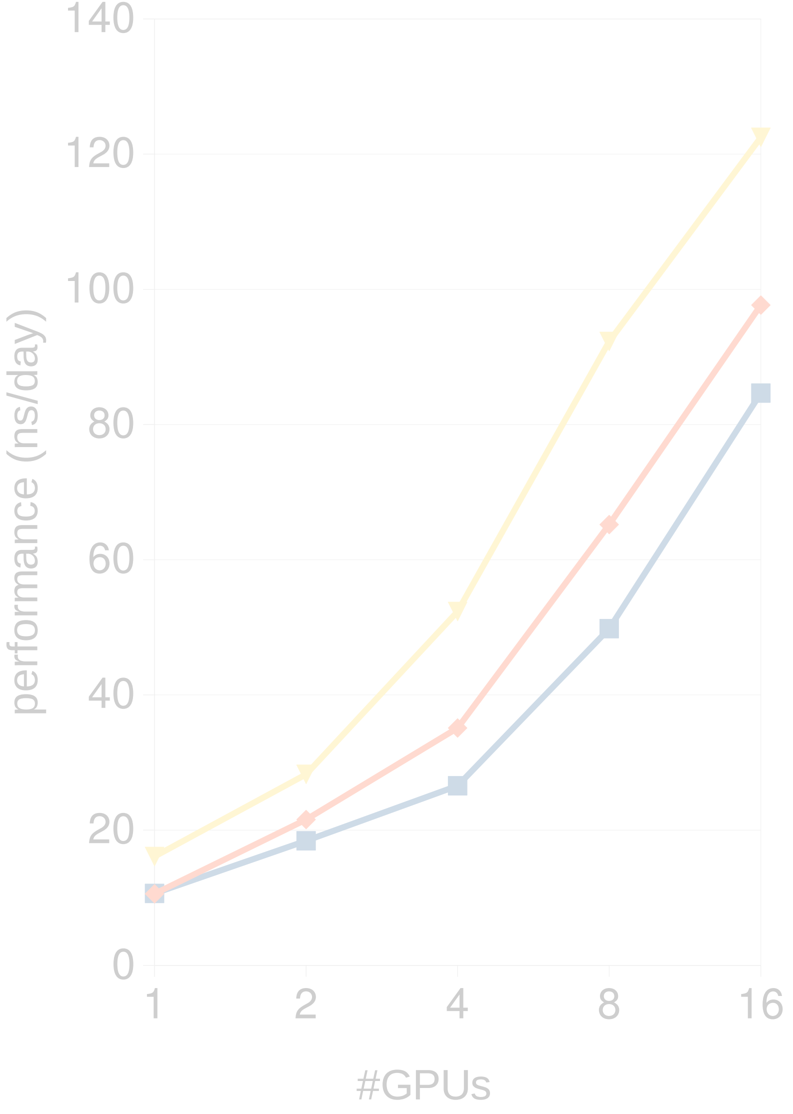
Recv X → PME Spline + Spread → Halo exchang → Distributed R2C FFT → PME Solve → Distributed C2R FFT → Halo exchang → PME Gather → Send PME F

PME Rank 1

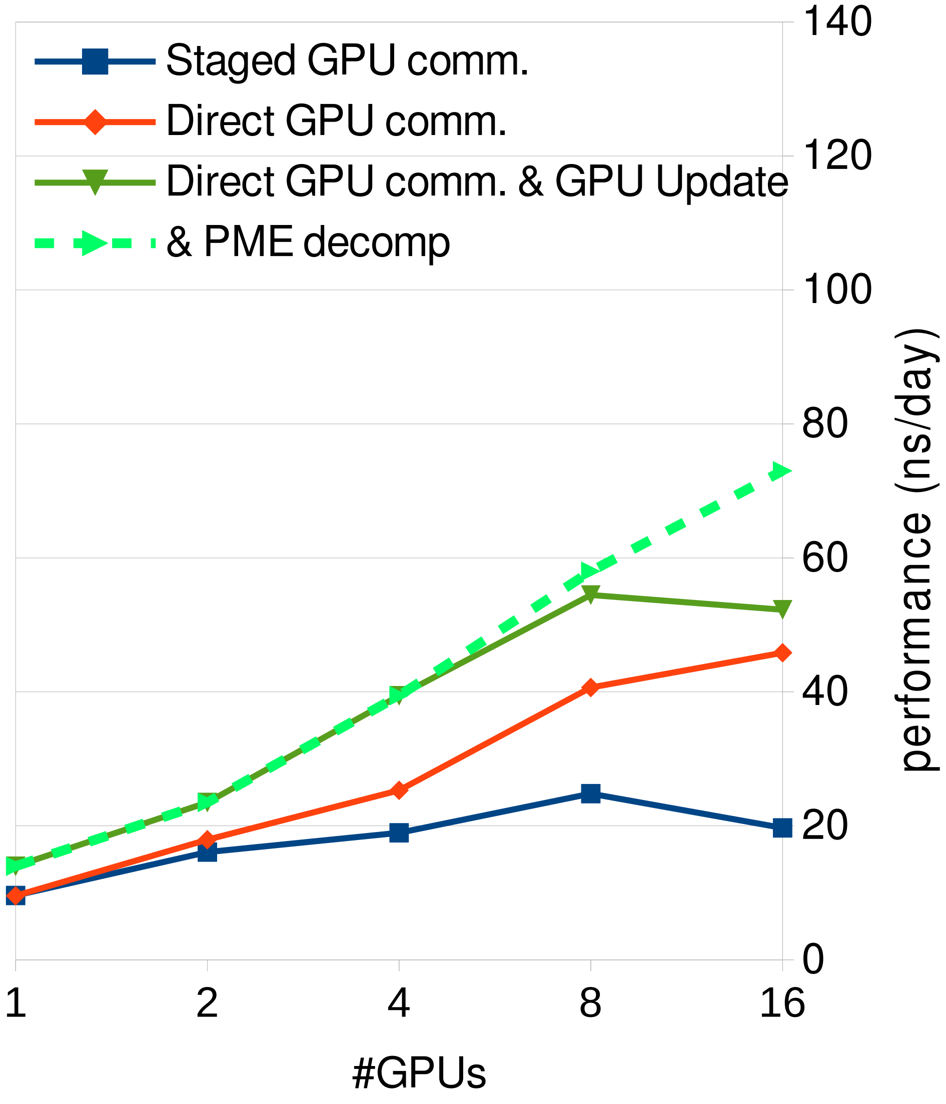
Recv X → PME Spline + Spread → Halo exchang → Distributed R2C FFT → PME Solve → Distributed C2R FFT → Halo exchang → PME Gather → Send PME F

Direct GPU communication with PME decomposition

- Major benefit on fast interconnects with GPU-resident steps
- Modest improvements on low-end interconnects

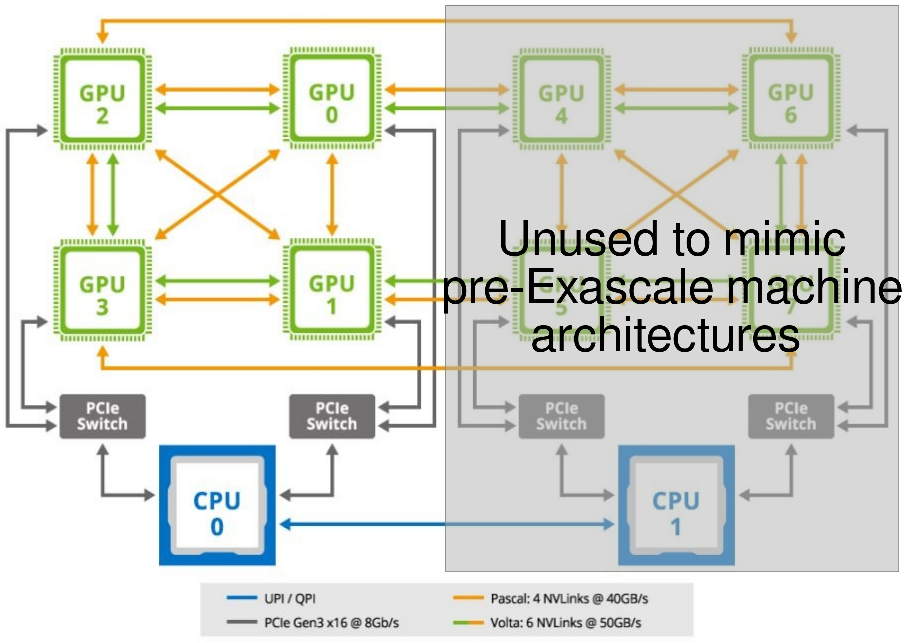


PME electrostatics:
halo-exchange & PME MPMD communication



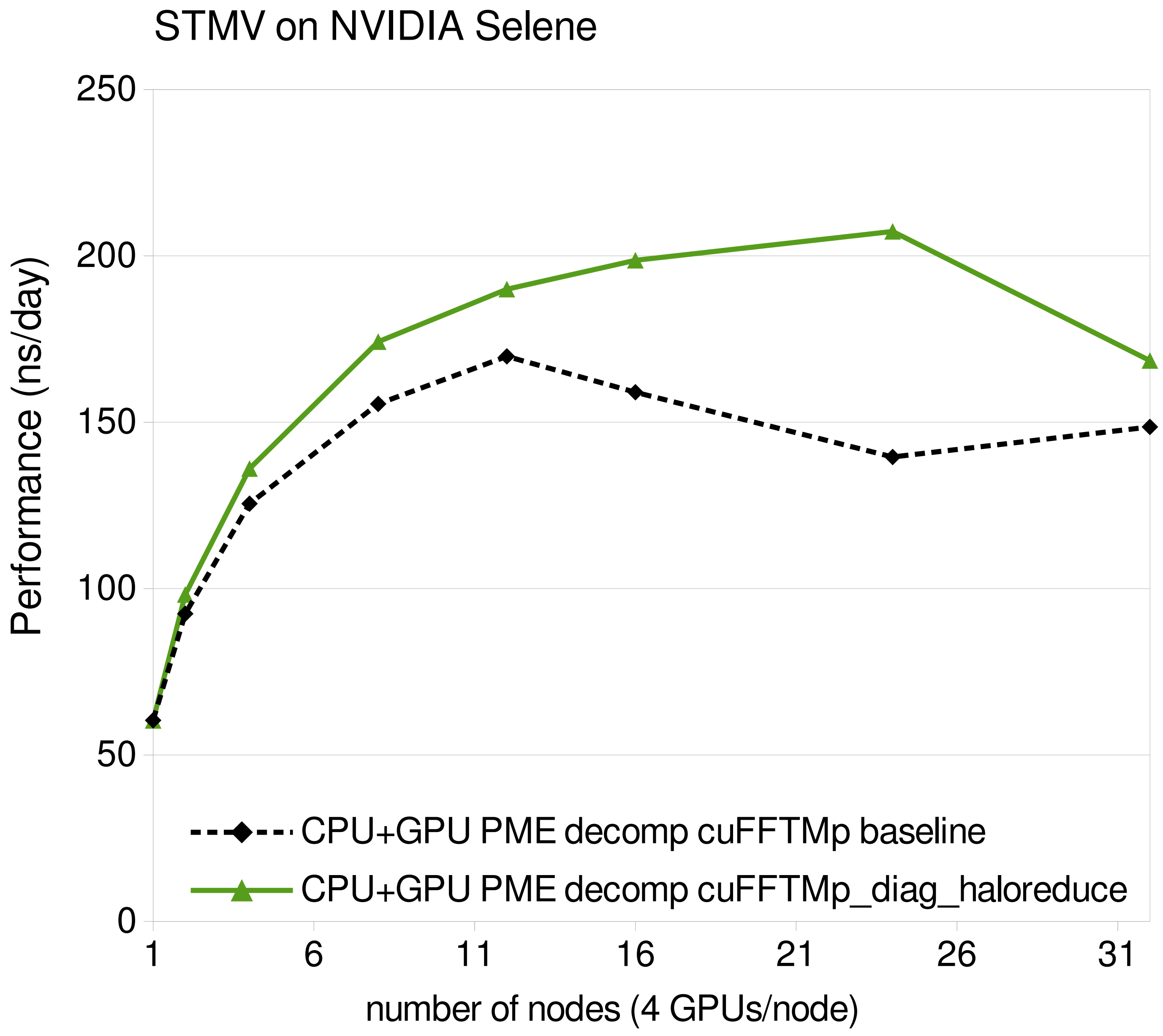
System: STMV 1M atoms

Hardware: DGX-1V



Unused to mimic pre-Exascale machine architectures

PME scaling improvements with cuFFTmp (in development)

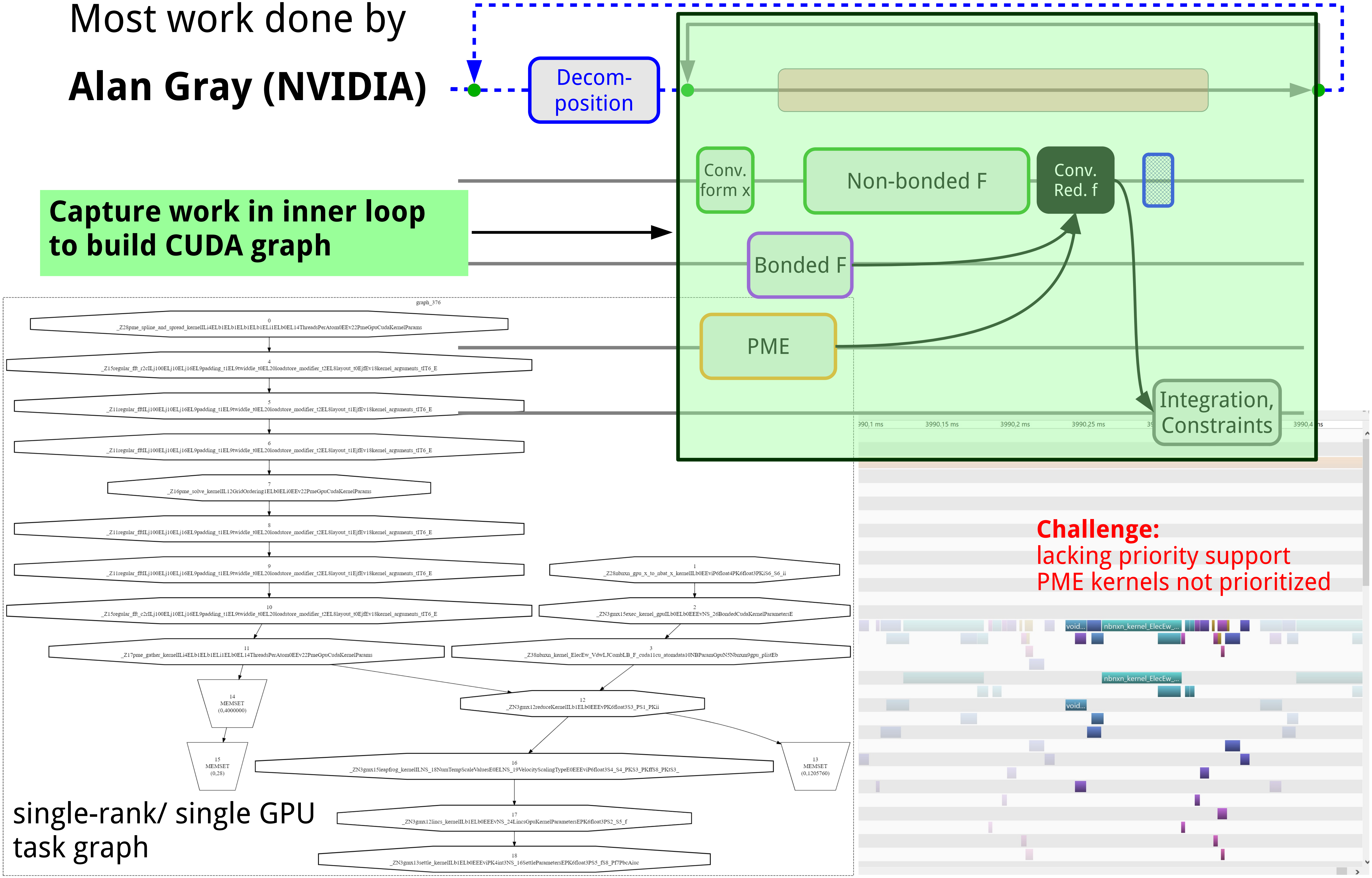


- Strong scales reasonably well to 16-24 nodes:
  - only 10-15k (!) atoms per GPU
  - further improvements planned
- Peak can still be lower than CPU-only machines
  - algorithm improvements needed
  - next-gen hardware expected to help

Asynchronous scheduling: CUDA graphs

Most work done by

Alan Gray (NVIDIA)

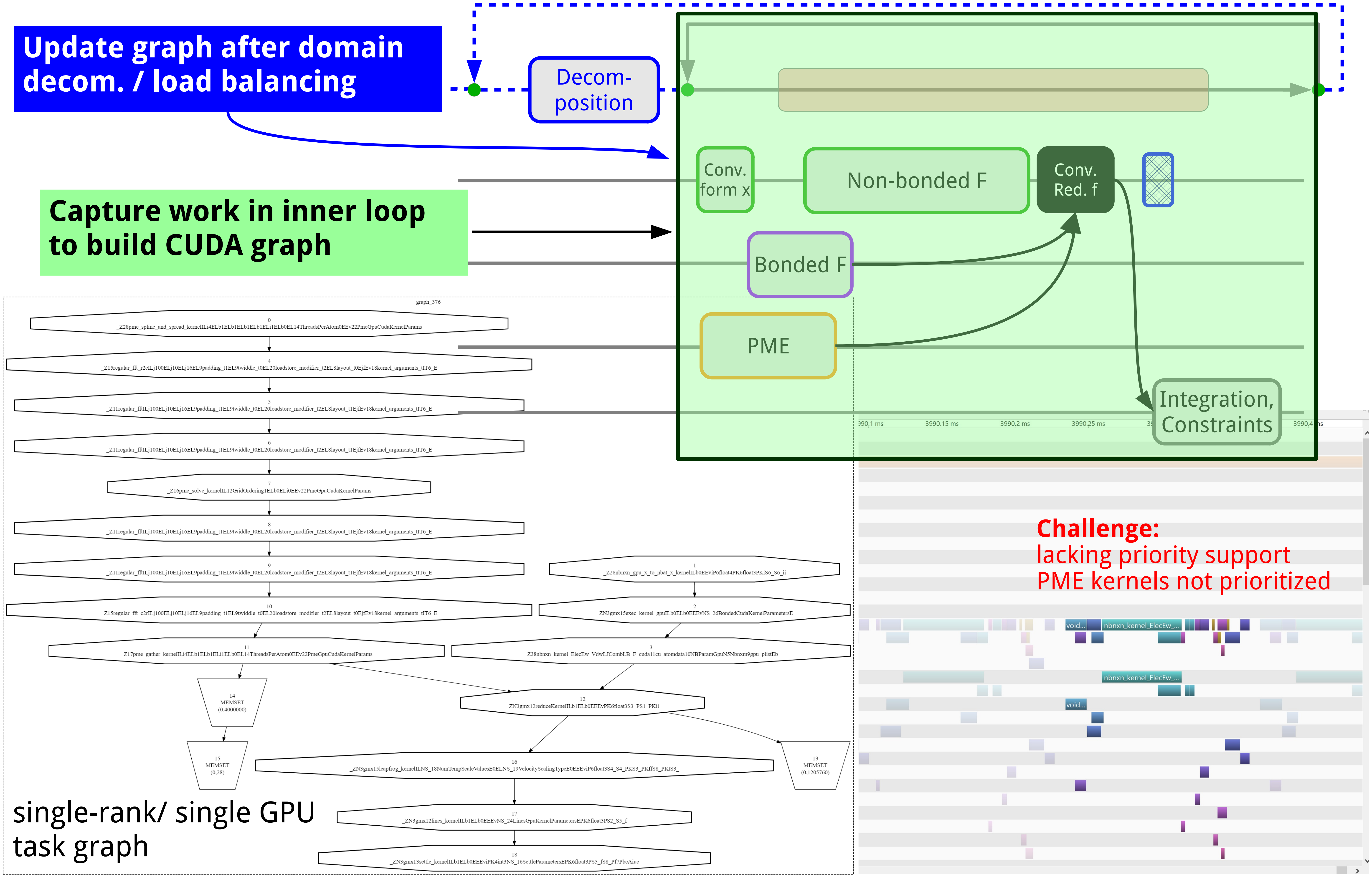

Decom-
position
Conv.
form x
Non-bonded F
Conv.
Red. f
Bonded F
PME
Integration,
Constraints
graph_376
0
_Z28pme_spline_and_spread_kernelILi4ELb1ELb1ELb1ELb1ELi1ELb0EL14ThreadsPerAtom0EEv22PmeGpuCudaKernelParams
4
_Z15regular_fft_r2cILj100ELj10ELj16EL9padding_t1EL9twiddle_t0EL20loadstore_modifier_t2EL8layout_t0EjfEv18kernel_arguments_tIT6_E
5
_Z11regular_fftILj100ELj10ELj16EL9padding_t1EL9twiddle_t0EL20loadstore_modifier_t2EL8layout_t1EjfEv18kernel_arguments_tIT6_E
6
_Z11regular_fftILj100ELj10ELj16EL9padding_t1EL9twiddle_t0EL20loadstore_modifier_t2EL8layout_t1EjfEv18kernel_arguments_tIT6_E
7
_Z16pme_solve_kernelIL12GridOrdering1ELb0ELi0EEv22PmeGpuCudaKernelParams
8
_Z11regular_fftILj100ELj10ELj16EL9padding_t1EL9twiddle_t0EL20loadstore_modifier_t2EL8layout_t1EjfEv18kernel_arguments_tIT6_E
9
_Z11regular_fftILj100ELj10ELj16EL9padding_t1EL9twiddle_t0EL20loadstore_modifier_t2EL8layout_t1EjfEv18kernel_arguments_tIT6_E
10
_Z15regular_fft_c2rILj100ELj10ELj16EL9padding_t1EL9twiddle_t0EL20loadstore_modifier_t2EL8layout_t0EjfEv18kernel_arguments_tIT6_E
11
_Z17pme_gather_kernelILi4ELb1ELb1ELi1ELb0EL14ThreadsPerAtom0EEv22PmeGpuCudaKernelParams
1
_Z28nbnxn_gpu_x_to_nbat_x_kernelILb0EEviP6float4PK6float3PKiS6_S6_ii
2
_ZN3gmx15exec_kernel_gpuILb0ELb0EEEvNS_26BondedCudaKernelParametersE
3
_Z38nbnxn_kernel_ElecEw_VdwLJCombLB_F_cuda11cu_atomdata10NBParamGpuN5Nbnxm9gpu_plistEb
14
MEMSET
(0,4000000)
12
_ZN3gmx12reduceKernelILb1ELb0EEEvPK6float3S3_PS1_PKii
15
MEMSET
(0,28)
16
_ZN3gmx15leapfrog_kernelILNS_18NumTempScaleValuesE0ELNS_19VelocityScalingTypeE0EEEviP6float3S4_S4_PKS3_PKffS8_PKtS3_
13
MEMSET
(0,1205760)
17
_ZN3gmx12lincs_kernelILb1ELb0EEEvNS_24LincsGpuKernelParametersEPK6float3PS2_S5_f
18
_ZN3gmx13settle_kernelILb1ELb0EEEviPK4int3NS_16SettleParametersEPK6float3PS5_fS8_Pf7PbcAiuc
390,1 ms
3990,15 ms
3990,2 ms
3990,25 ms
3990,4 ms
void...
nbnxn_kernel_ElecEw_...
nbnxn_kernel_ElecEw_...
void...

Capture work in inner loop to build CUDA graph

Challenge:
lacking priority support
PME kernels not prioritized

single-rank/ single GPU task graph

Asynchronous scheduling: CUDA graphs


Decom-
position
Conv.
form x
Non-bonded F
Conv.
Red. f
Bonded F
PME
Integration,
Constraints
graph_376
0
_Z28pme_spline_and_spread_kernelILi4ELb1ELb1ELb1ELb1ELi1ELb0EL14ThreadsPerAtom0EEv22PmeGpuCudaKernelParams
4
_Z15regular_fft_r2cILj100ELj10ELj16EL9padding_t1EL9twiddle_t0EL20loadstore_modifier_t2EL8layout_t0EjfEv18kernel_arguments_tIT6_E
5
_Z11regular_fftILj100ELj10ELj16EL9padding_t1EL9twiddle_t0EL20loadstore_modifier_t2EL8layout_t1EjfEv18kernel_arguments_tIT6_E
6
_Z11regular_fftILj100ELj10ELj16EL9padding_t1EL9twiddle_t0EL20loadstore_modifier_t2EL8layout_t1EjfEv18kernel_arguments_tIT6_E
7
_Z16pme_solve_kernelIL12GridOrdering1ELb0ELi0EEv22PmeGpuCudaKernelParams
8
_Z11regular_fftILj100ELj10ELj16EL9padding_t1EL9twiddle_t0EL20loadstore_modifier_t2EL8layout_t1EjfEv18kernel_arguments_tIT6_E
9
_Z11regular_fftILj100ELj10ELj16EL9padding_t1EL9twiddle_t0EL20loadstore_modifier_t2EL8layout_t1EjfEv18kernel_arguments_tIT6_E
1
_Z28nbnxn_gpu_x_to_nbat_x_kernelILb0EEviP6float4PK6float3PKiS6_S6_ii
10
_Z15regular_fft_c2rILj100ELj10ELj16EL9padding_t1EL9twiddle_t0EL20loadstore_modifier_t2EL8layout_t0EjfEv18kernel_arguments_tIT6_E
2
_ZN3gmx15exec_kernel_gpuILb0ELb0EEEvNS_26BondedCudaKernelParametersE
11
_Z17pme_gather_kernelILi4ELb1ELb1ELi1ELb0EL14ThreadsPerAtom0EEv22PmeGpuCudaKernelParams
3
_Z38nbnxn_kernel_ElecEw_VdwLJCombLB_F_cuda11cu_atomdata10NBParamGpuN5Nbnxm9gpu_plistEb
14
MEMSET
(0,4000000)
12
_ZN3gmx12reduceKernelILb1ELb0EEEvPK6float3S3_PS1_PKii
15
MEMSET
(0,28)
16
_ZN3gmx15leapfrog_kernelILNS_18NumTempScaleValuesE0ELNS_19VelocityScalingTypeE0EEEviP6float3S4_S4_PKS3_PKfS8_PKtS3_
13
MEMSET
(0,1205760)
17
_ZN3gmx12lincs_kernelILb1ELb0EEEvNS_24LincsGpuKernelParametersEPK6float3PS2_S5_f
18
_ZN3gmx13settle_kernelILb1ELb0EEEviPK4int3NS_16SettleParametersEPK6float3PS5_fS8_Pf7PbcAiuc
390.1 ms
3990.15 ms
3990.2 ms
3990.25 ms
3990.4 ms
void...
nbnxn_kernel_ElecEw_...
nbnxn_kernel_ElecEw_...
void...

Update graph after domain decom. / load balancing

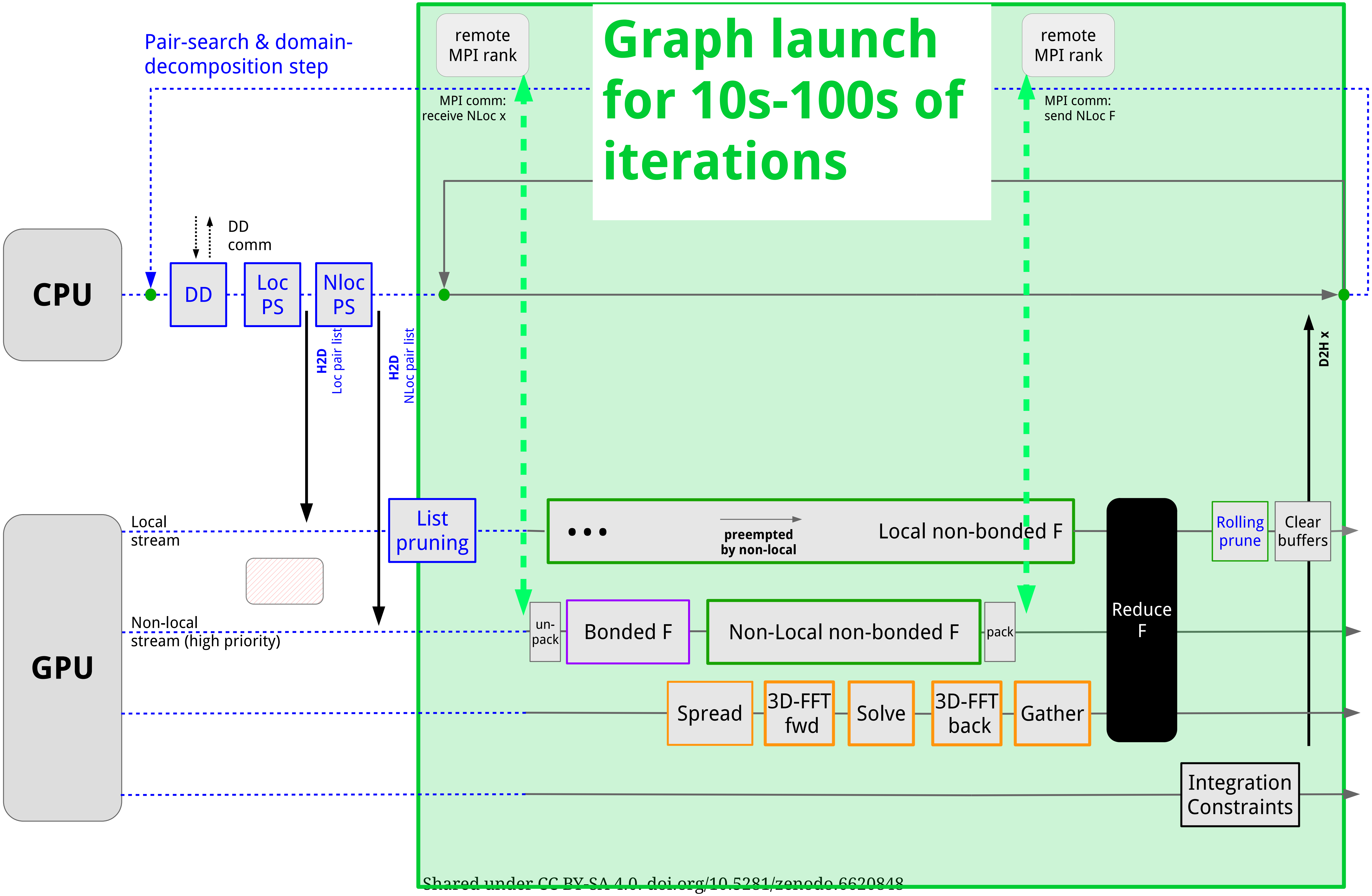
Capture work in inner loop to build CUDA graph

single-rank/ single GPU task graph

Challenge:
lacking priority support
PME kernels not prioritized

Codesign project with NVIDIA

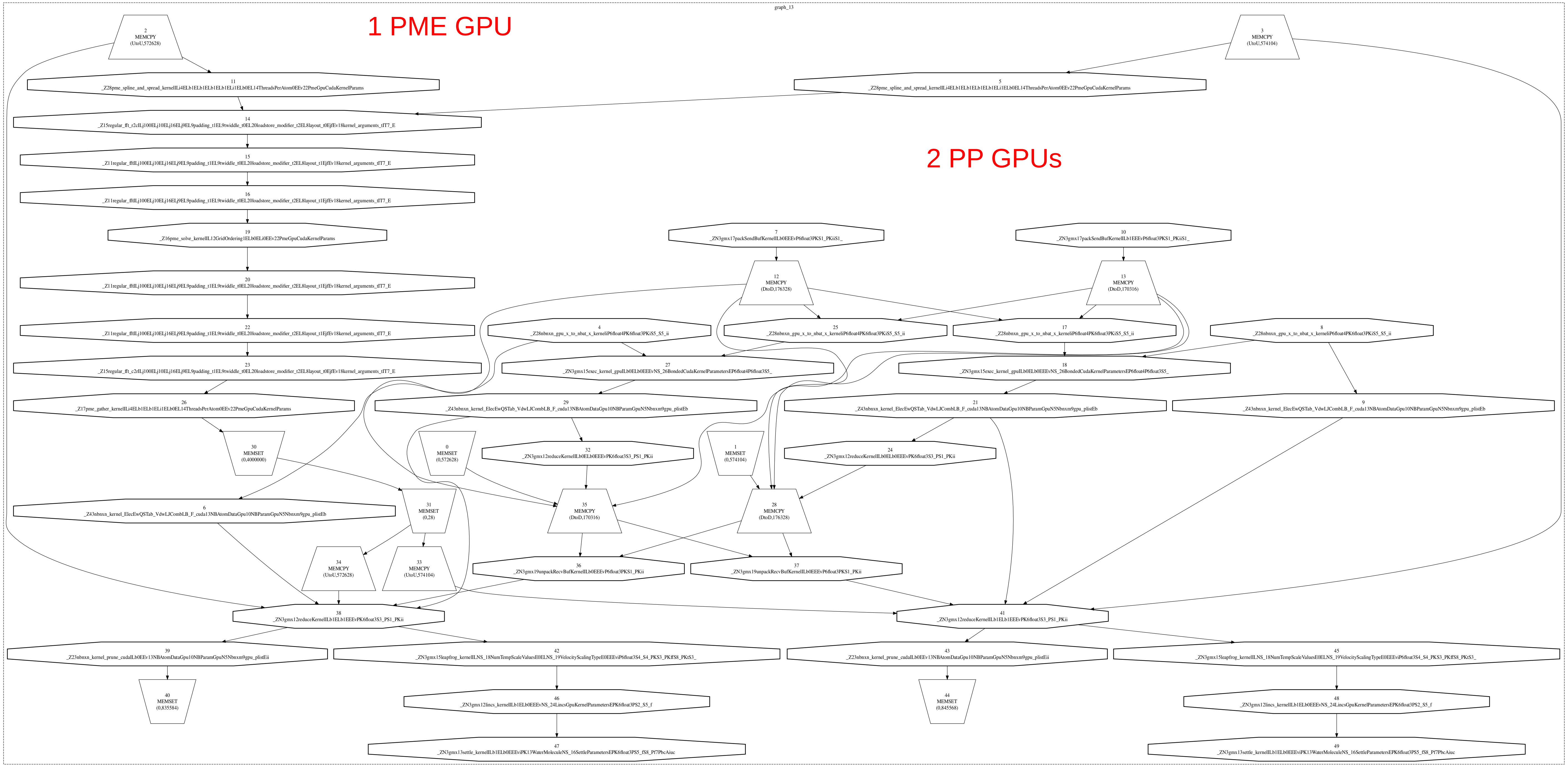
Multi-GPU graph scheduling



Graph launch for 10s-100s of iterations

Asynchronous scheduling: CUDA graphs

- multi-rank: leverage thread-MPI using pthreads for UVA direct async copies



1 PME GPU

2 PP GPUs

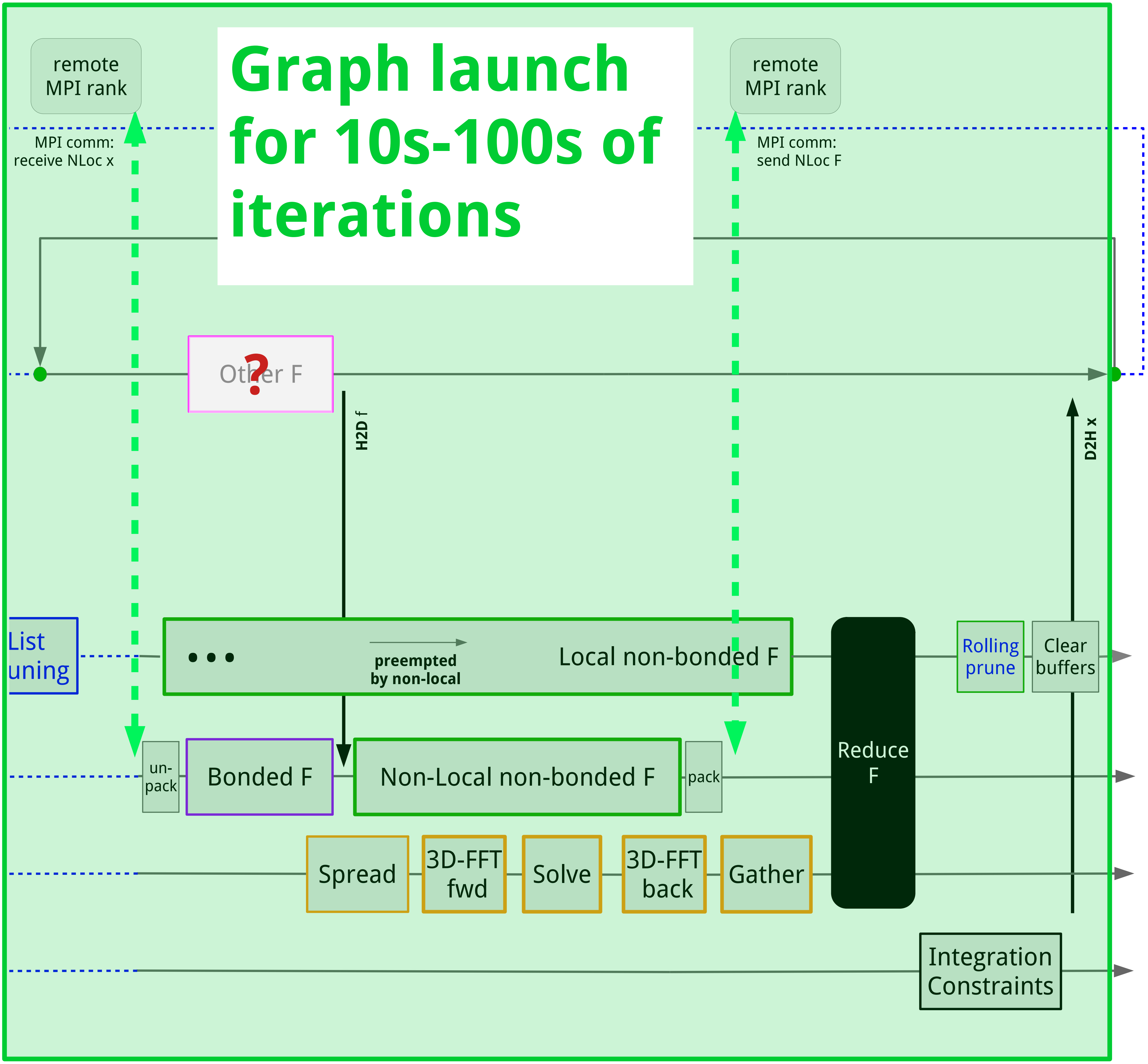
graph_13
2 MEMCPY (UtoU,572628)
3 MEMCPY (UtoU,574104)
11 _Z28pme_spline_and_spread_kernelILi4ELb1ELb1ELb1ELb1ELi1ELb0EL14ThreadsPerAtom0EEv22PmeGpuCudaKernelParams
5 _Z28pme_spline_and_spread_kernelILi4ELb1ELb1ELb1ELb1ELi1ELb0EL14ThreadsPerAtom0EEv22PmeGpuCudaKernelParams
14 _Z15regular_fft_r2cILj100ELj10ELj16ELj9EL9padding_t1EL9twiddle_t0EL20loadstore_modifier_t2EL8layout_t0EjfEv18kernel_arguments_tIT7_E
15 _Z11regular_fftILj100ELj10ELj16ELj9EL9padding_t1EL9twiddle_t0EL20loadstore_modifier_t2EL8layout_t1EjfEv18kernel_arguments_tIT7_E
16 _Z11regular_fftILj100ELj10ELj16ELj9EL9padding_t1EL9twiddle_t0EL20loadstore_modifier_t2EL8layout_t1EjfEv18kernel_arguments_tIT7_E
19 _Z16pme_solve_kernelIL12GridOrdering1ELb0ELi0EEv22PmeGpuCudaKernelParams
7 _ZN3gmx17packSendBufKernelILb0EEEvP6float3PKS1_PKiiS1_
10 _ZN3gmx17packSendBufKernelILb1EEEvP6float3PKS1_PKiiS1_
20 _Z11regular_fftILj100ELj10ELj16ELj9EL9padding_t1EL9twiddle_t0EL20loadstore_modifier_t2EL8layout_t1EjfEv18kernel_arguments_tIT7_E
12 MEMCPY (DtoD,176328)
13 MEMCPY (DtoD,170316)
22 _Z11regular_fftILj100ELj10ELj16ELj9EL9padding_t1EL9twiddle_t0EL20loadstore_modifier_t2EL8layout_t1EjfEv18kernel_arguments_tIT7_E
4 _Z28nbnxn_gpu_x_to_nbat_x_kerneliP6float4PK6float3PKiS5_S5_ii
25 _Z28nbnxn_gpu_x_to_nbat_x_kerneliP6float4PK6float3PKiS5_S5_ii
17 _Z28nbnxn_gpu_x_to_nbat_x_kerneliP6float4PK6float3PKiS5_S5_ii
8 _Z28nbnxn_gpu_x_to_nbat_x_kerneliP6float4PK6float3PKiS5_S5_ii
23 _Z15regular_fft_c2rILj100ELj10ELj16ELj9EL9padding_t1EL9twiddle_t0EL20loadstore_modifier_t2EL8layout_t0EjfEv18kernel_arguments_tIT7_E
27 _ZN3gmx15exec_kernel_gpuILb0ELb0EEEvNS_26BondedCudaKernelParametersEP6float4P6float3S5_
18 _ZN3gmx15exec_kernel_gpuILb0ELb0EEEvNS_26BondedCudaKernelParametersEP6float4P6float3S5_
26 _Z17pme_gather_kernelILi4ELb1ELb1ELi1ELb0EL14ThreadsPerAtom0EEv22PmeGpuCudaKernelParams
29 _Z43nbnxn_kernel_ElecEwQSTab_VdwLJCombLB_F_cuda13NBAtomDataGpu10NBParamGpuN5Nbnxm9gpu_plistEb
21 _Z43nbnxn_kernel_ElecEwQSTab_VdwLJCombLB_F_cuda13NBAtomDataGpu10NBParamGpuN5Nbnxm9gpu_plistEb
9 _Z43nbnxn_kernel_ElecEwQSTab_VdwLJCombLB_F_cuda13NBAtomDataGpu10NBParamGpuN5Nbnxm9gpu_plistEb
30 MEMSET (0,4000000)
0 MEMSET (0,572628)
32 _ZN3gmx12reduceKernelILb0ELb0EEEvPK6float3S3_PS1_PKii
1 MEMSET (0,574104)
24 _ZN3gmx12reduceKernelILb0ELb0EEEvPK6float3S3_PS1_PKii
6 _Z43nbnxn_kernel_ElecEwQSTab_VdwLJCombLB_F_cuda13NBAtomDataGpu10NBParamGpuN5Nbnxm9gpu_plistEb
31 MEMSET (0,28)
35 MEMCPY (DtoD,170316)
28 MEMCPY (DtoD,176328)
34 MEMCPY (UtoU,572628)
33 MEMCPY (UtoU,574104)
36 _ZN3gmx19unpackRecvBufKernelILb0EEEvP6float3PKS1_PKii
37 _ZN3gmx19unpackRecvBufKernelILb0EEEvP6float3PKS1_PKii
38 _ZN3gmx12reduceKernelILb1ELb1EEEvPK6float3S3_PS1_PKii
41 _ZN3gmx12reduceKernelILb1ELb1EEEvPK6float3S3_PS1_PKii
39 _Z23nbnxn_kernel_prune_cudaILb0EEv13NBAtomDataGpu10NBParamGpuN5Nbnxm9gpu_plistEii
42 _ZN3gmx15leapfrog_kernelILNS_18NumTempScaleValuesE0ELNS_19VelocityScalingTypeE0EEEviP6float3S4_S4_PKS3_PKffS8_PKtS3_
43 _Z23nbnxn_kernel_prune_cudaILb0EEv13NBAtomDataGpu10NBParamGpuN5Nbnxm9gpu_plistEii
45 _ZN3gmx15leapfrog_kernelILNS_18NumTempScaleValuesE0ELNS_19VelocityScalingTypeE0EEEviP6float3S4_S4_PKS3_PKffS8_PKtS3_
40 MEMSET (0,835584)
46 _ZN3gmx12lincs_kernelILb1ELb0EEEvNS_24LincsGpuKernelParametersEPK6float3PS2_S5_f
44 MEMSET (0,845568)
48 _ZN3gmx12lincs_kernelILb1ELb0EEEvNS_24LincsGpuKernelParametersEPK6float3PS2_S5_f
47 _ZN3gmx13settle_kernelILb1ELb0EEEviPK13WaterMoleculeNS_16SettleParametersEPK6float3PS5_fS8_Pf7PbcAiuc
49 _ZN3gmx13settle_kernelILb1ELb0EEEviPK13WaterMoleculeNS_16SettleParametersEPK6float3PS5_fS8_Pf7PbcAiuc

Generated with **cudaGraphDebugDotPrint()**

Codesign project with NVIDIA

Multi-GPU graph scheduling

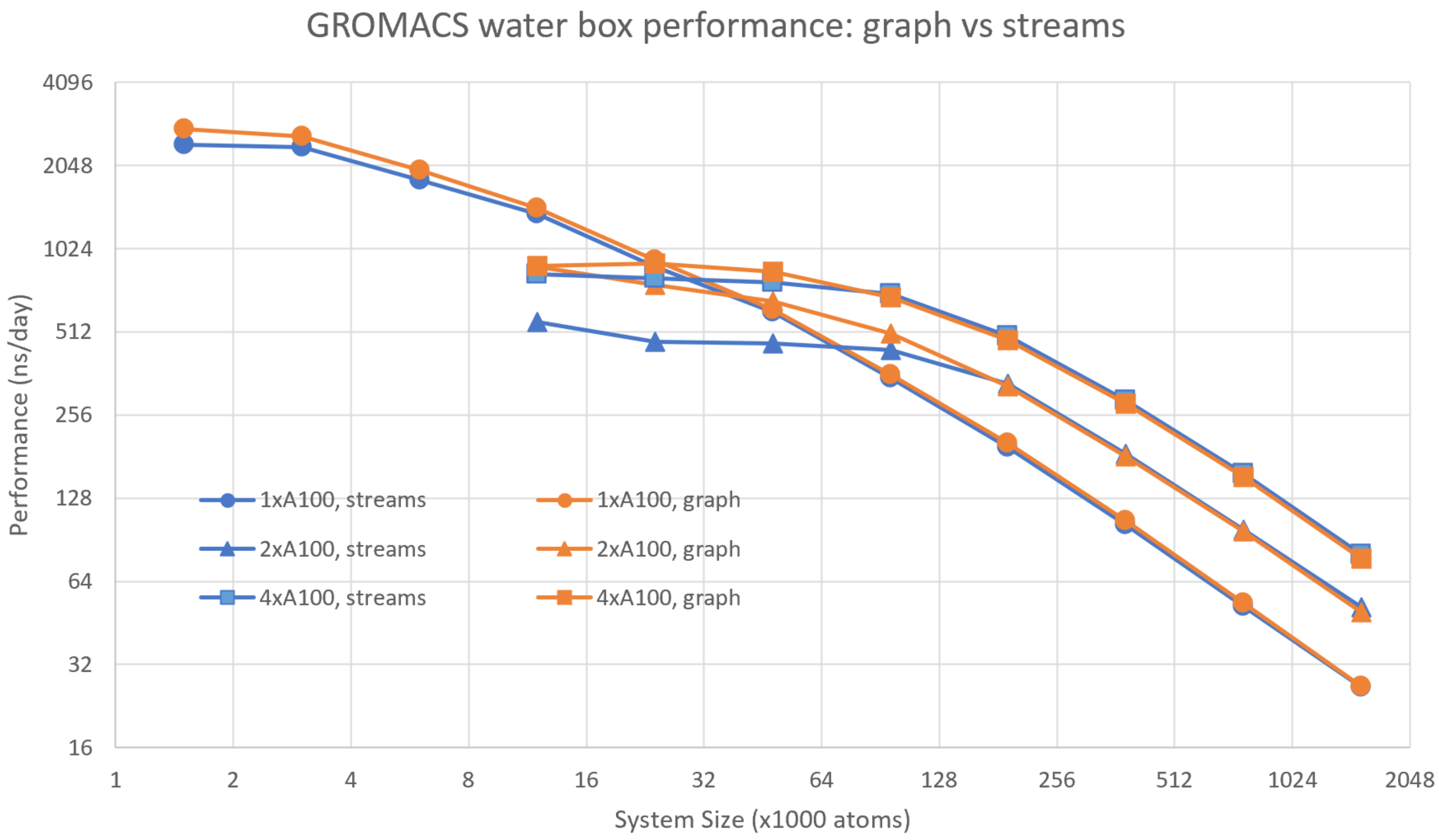
- Inner loop compute + all intra-node communication **with a single graph launch**
  - Reduced overhead
  - Allow the runtime to optimize schedule
- Challenges / WIP:
  - integrating CPU tasks
  - heterogeneous iterations
  - inter-node communication



Graph launch for 10s-100s of iterations

Codesign project with NVIDIA

CUDA graph scheduling performance



Codesign project with Intel

SYCL for portability & performance

- SeRC & Intel: OneAPI CoE: Andrey Alekseenko (KTH)
- **1st GPU backend with DPC++**
  - early prototype released in GROMACS 2021
- added hipSYCL support as portability check first

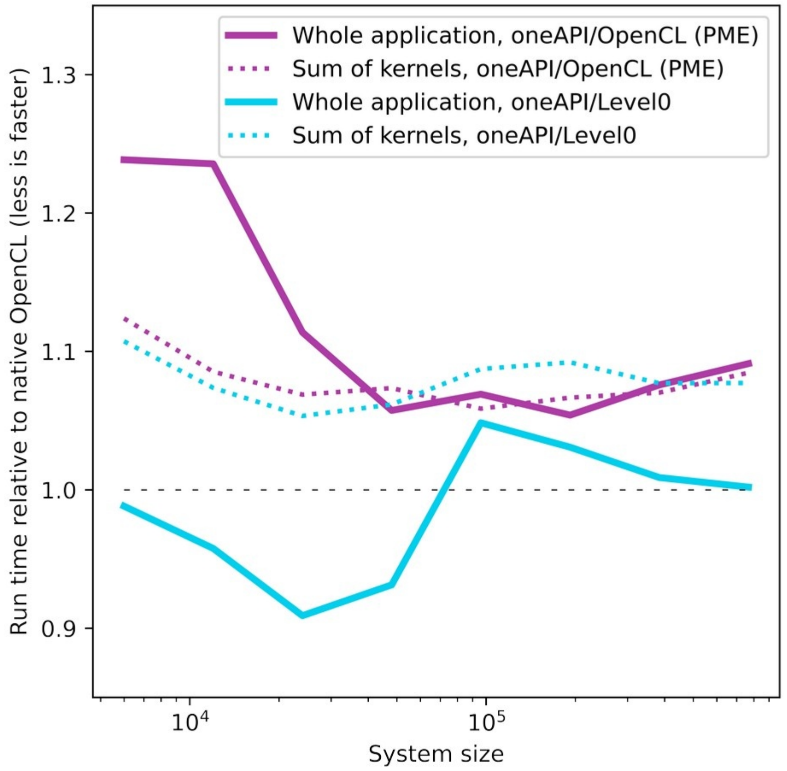
- starting with the 2022 release:

  GROMACS adopted **hipSYCL for production AMD support**

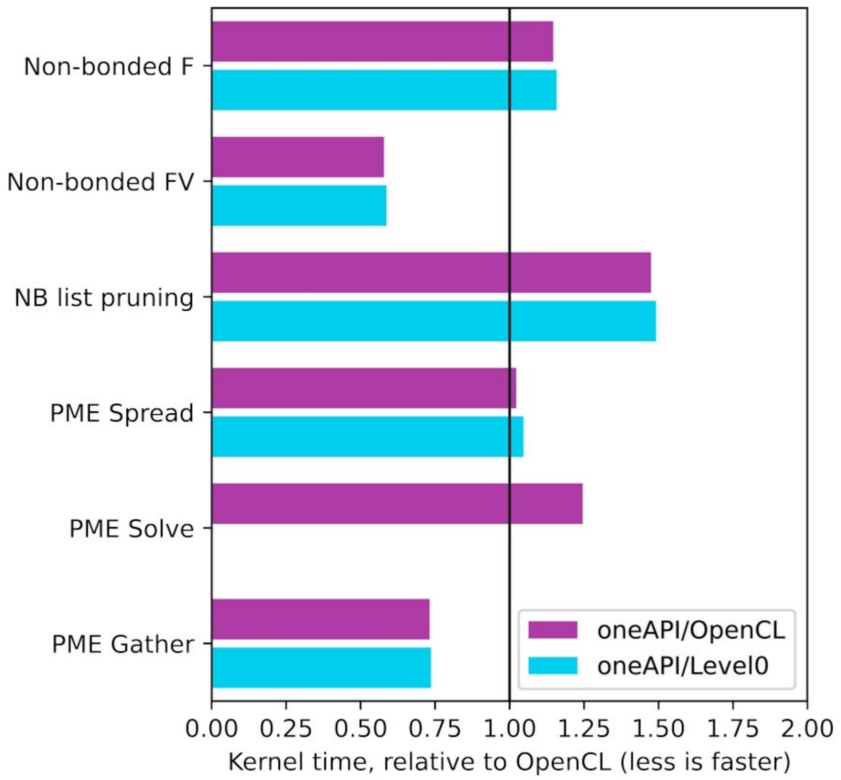
- SYCL to replace OpenCL as portability GPU backend
  - already broader feature set coverage
  - broad vendor support: AMD, NVIDIA, Intel

SYCL in GROMACS relative to native

Intel DG1

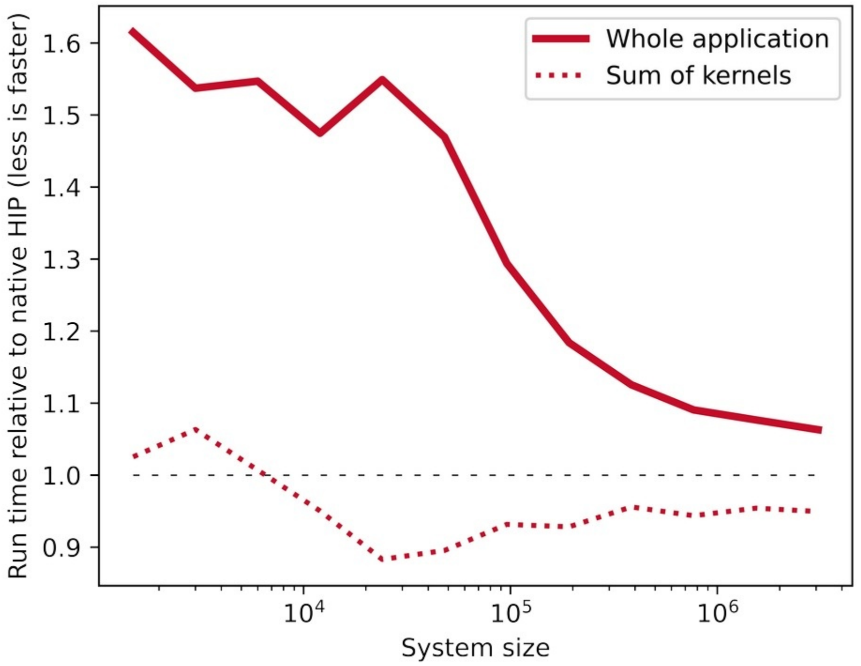


PME electrostatics, 384k system size

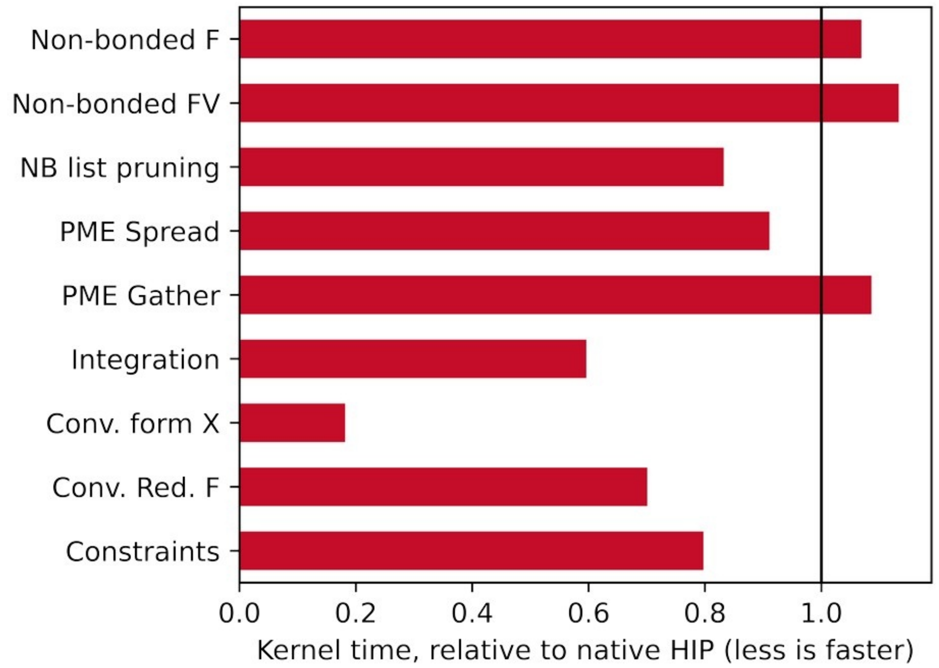


oneAPI/DPC++ on OpenCL/L0 vs native OpenCL
(oneAPI 2022.0 except L0 2021.4)

AMD MI100



PME electrostatics, 384k system size



hipSYCL vs native HIP on ROCm 4.5.2

Relative application perf and rel GPU kernels perf

Perf of individual GPU kernels

Take-aways

- Codesign key for algorithm reformulation / redesign
  - enabled to keep up with hardware evolution
  - need to be forward-looking
  - disruptive vs constructive
- Long-term investment
- Interdisciplinarity helps but challenges too
- Collaboration needs long-term alignment
  - much easier intra-team/community
  - harder and often challenging cross-team
- Plan for the progress but allow for the incidental collaboration
  - accommodating SW design will allow new domain-science contribution

Acknowledgments

GROMACS

Andrey Alekseenko

Artem Zhmurov

Berk Hess

Erik Lindahl

Magnus Lundborg

Paul Bauer

Mark Abraham (Intel)

Roland Schulz (Intel)

Alan Gray (NVIDIA)

Gaurav Garg (NVIDIA)

HW / code contrib

NVIDIA

intel

AMD

CSCS
Swiss National Supercomputing Centre

CSC

STREAM
High Performance Computing

RIST

Funding

Swedish Foundation for Strategic Research

SeRC
Swedish e-Science Research Centre

erc
European Research Council

Vetenskapsrådet

bioexcel