Advances in the OpenCL Offload Support in GROMACS

Szilárd Páll
pszilard@kth.se

Roland Schulz
roland.schulz@intel.com

May 14, 2019

IWOCL

GROMACS

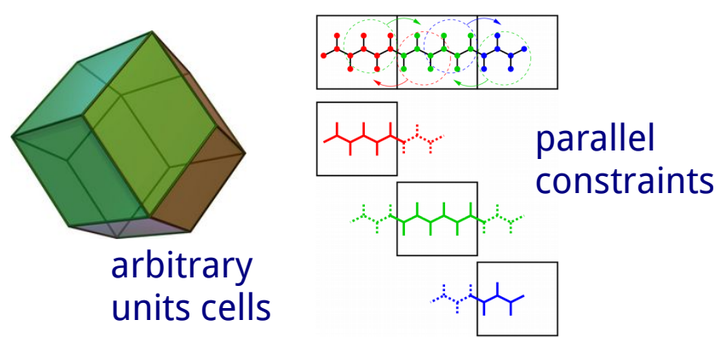
FAST. FLEXIBLE. FREE.

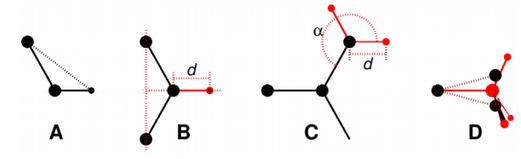
- **Classical molecular dynamics** (MD) code
  - all major force-fields
  - broad algorithm support
- **Development**:

  Stockholm SE & partners worldwide

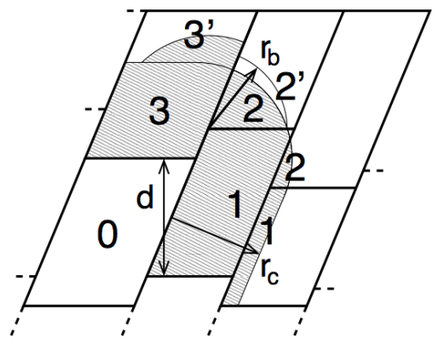
- **Large user base**:
  - 10k's academic & industry
  - deployed on most HPC resources

- **Open source**: LGPLv2
- **Open development:**
  - code review: `https://gerrit.gromacs.org`
  - bug-tracker: `https://redmine.gromacs.org`

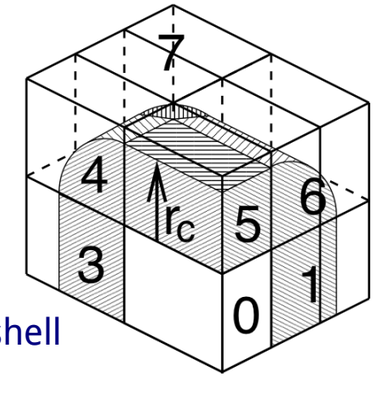




virtual interaction sites



Triclinic unit cell with
load balancing and
staggered cell boundaries

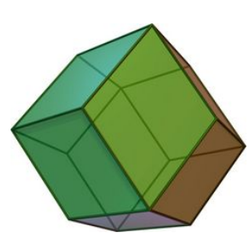


eighth shell
domain
decomposition

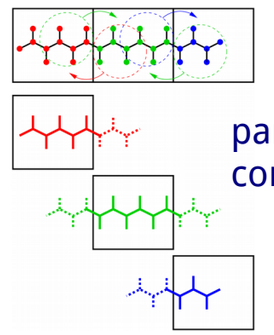
GROMACS

FAST. FLEXIBLE. FREE.

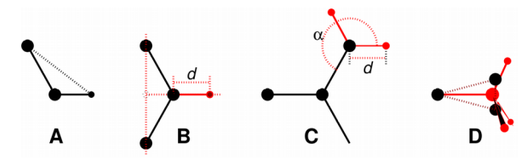
- **Code:**
  - C++14 (in transition from C99)
  - >1M lines of code (non-generated)
- **Parallelization:**
  - MPI: SPMD / MPMD (thread-MPI)
  - OpenMP
  - SIMD: 14 flavors
    - library abstraction above intrinsics
  - CUDA, OpenCL
- **Bottom-up performance tuning:**
  - absolute performance over “just scaling”
- **Portability in focus**
  - broad in-house CI testing, Linux distros
  - regular manual testing on all HPC arch



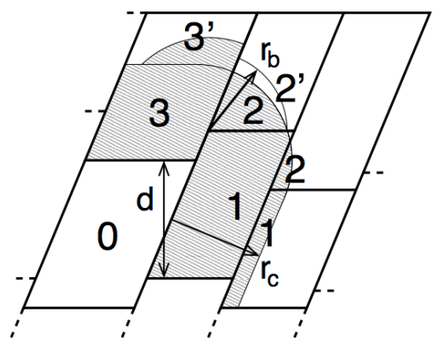
arbitrary units cells



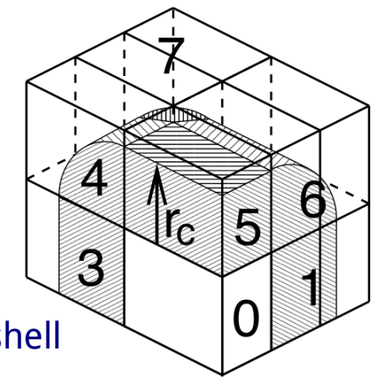
parallel constraints



virtual interaction sites

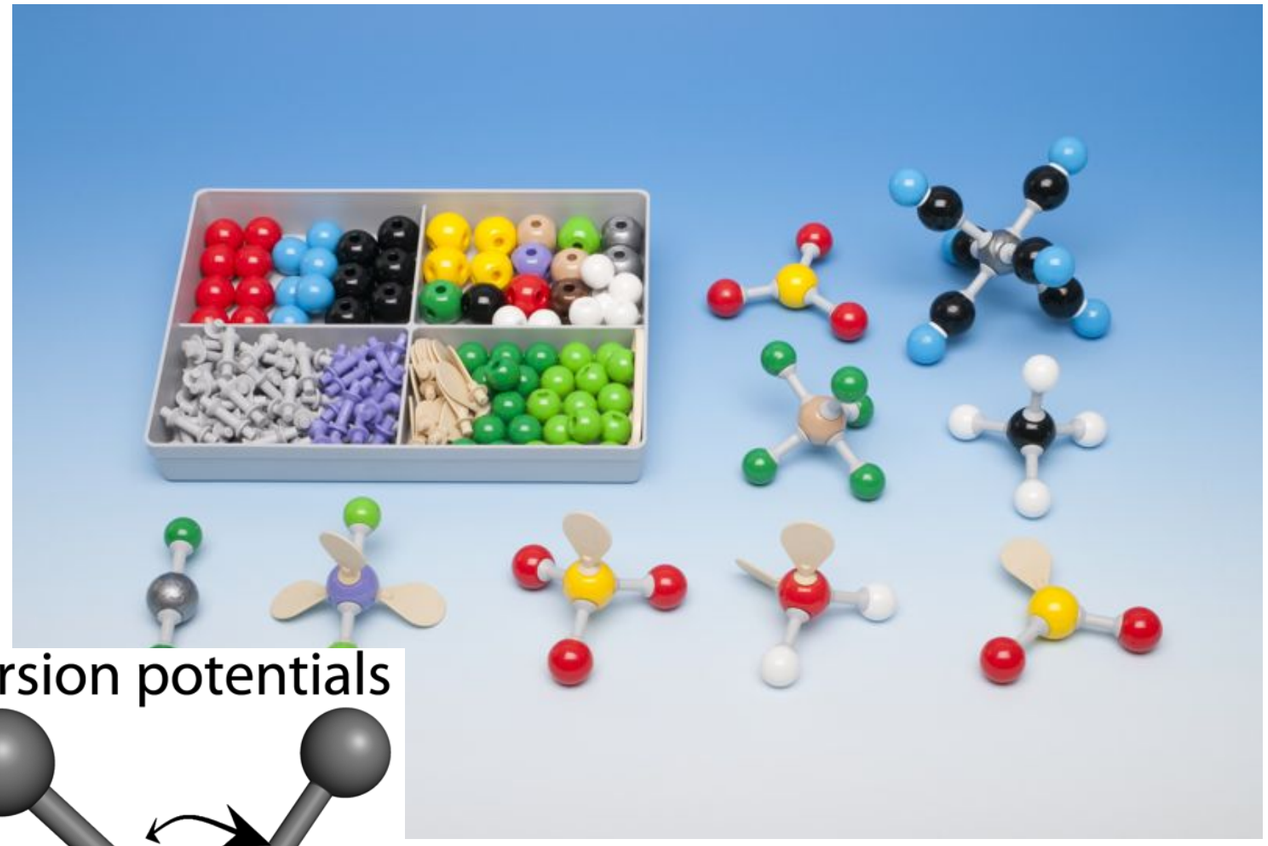


Triclinic unit cell with load balancing and staggered cell boundaries

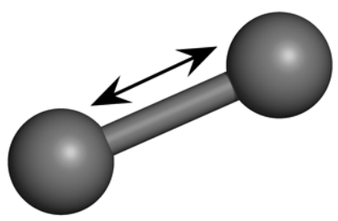


eighth shell domain decomposition

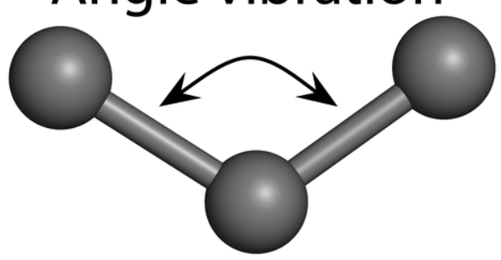
Molecular dynamics



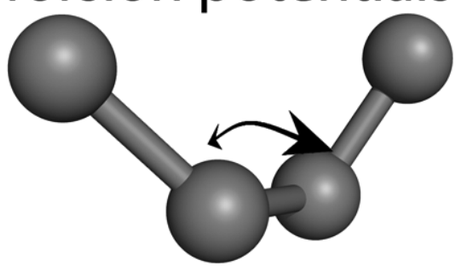
Bond vibration



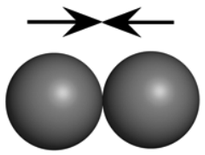
Angle vibration



Torsion potentials



van der Waals interactions



Electrostatics

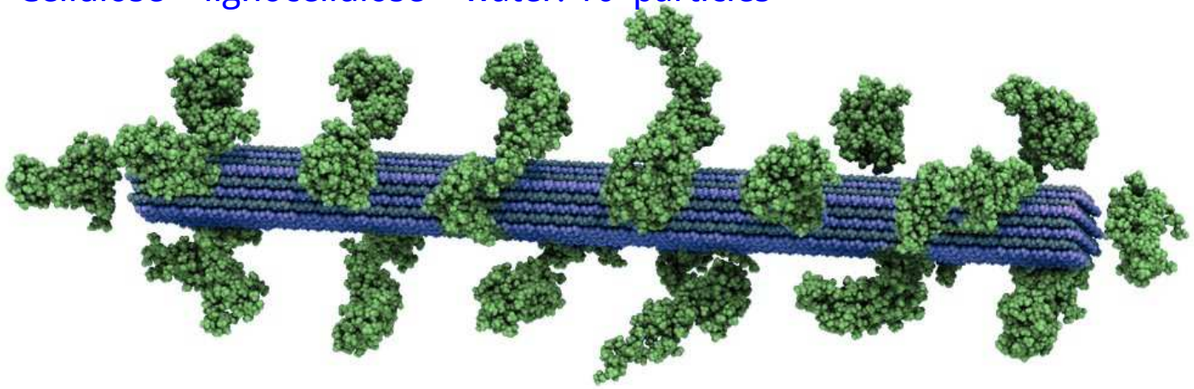
$$U(\mathbf{r}) = \sum_{bonds} U_{bond}(\mathbf{r}) + \sum_{angles} U_{angle}(\mathbf{r}) + \sum_{dihs} U_{dih}(\mathbf{r})$$ Bonded

$$+ \sum_{i} \sum_{j>i} \frac{q_i q_j}{4\pi\epsilon_0 r_{ij}} + \frac{A_{ij}}{r_{ij}^{12}} - \frac{B_{ij}}{r_{ij}^{6}}$$ Non-bonded

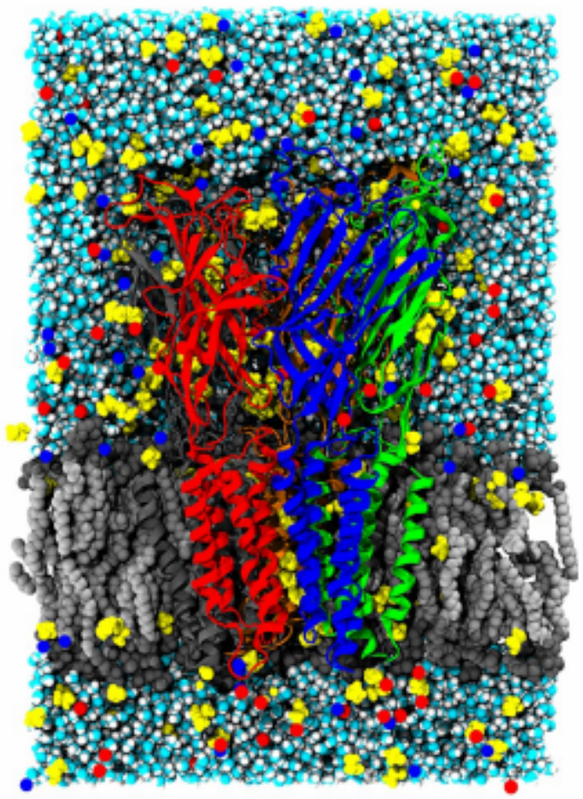
Over all atom-pairs!

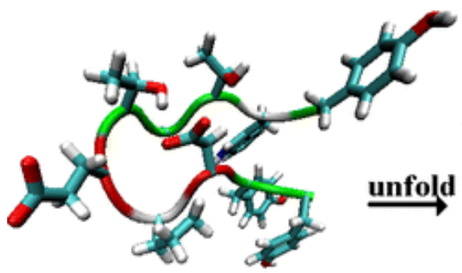
Molecular simulation: use-cases

Cellulose + lignocellulose + water: 10^7 particles

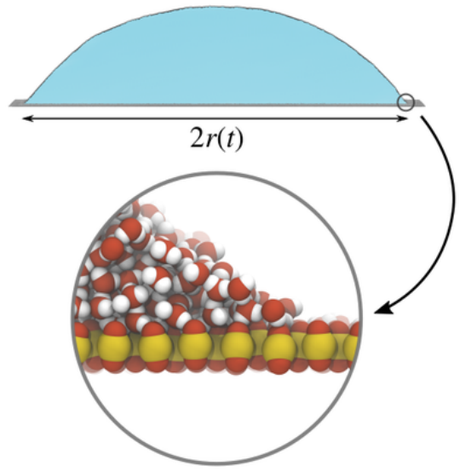


Membrane protein: 10^5 particles

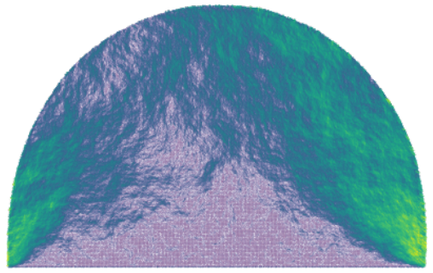


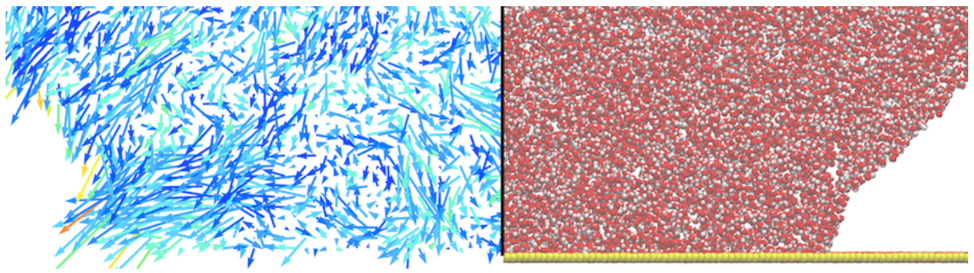


Chignolin folding: 10^4 particles



Contact line friction & wetting dynamics 10^7-10^9 particles





Biomolecular MD

time-scale challenge: femto- to millisecond

strong scaling

Materials MD:

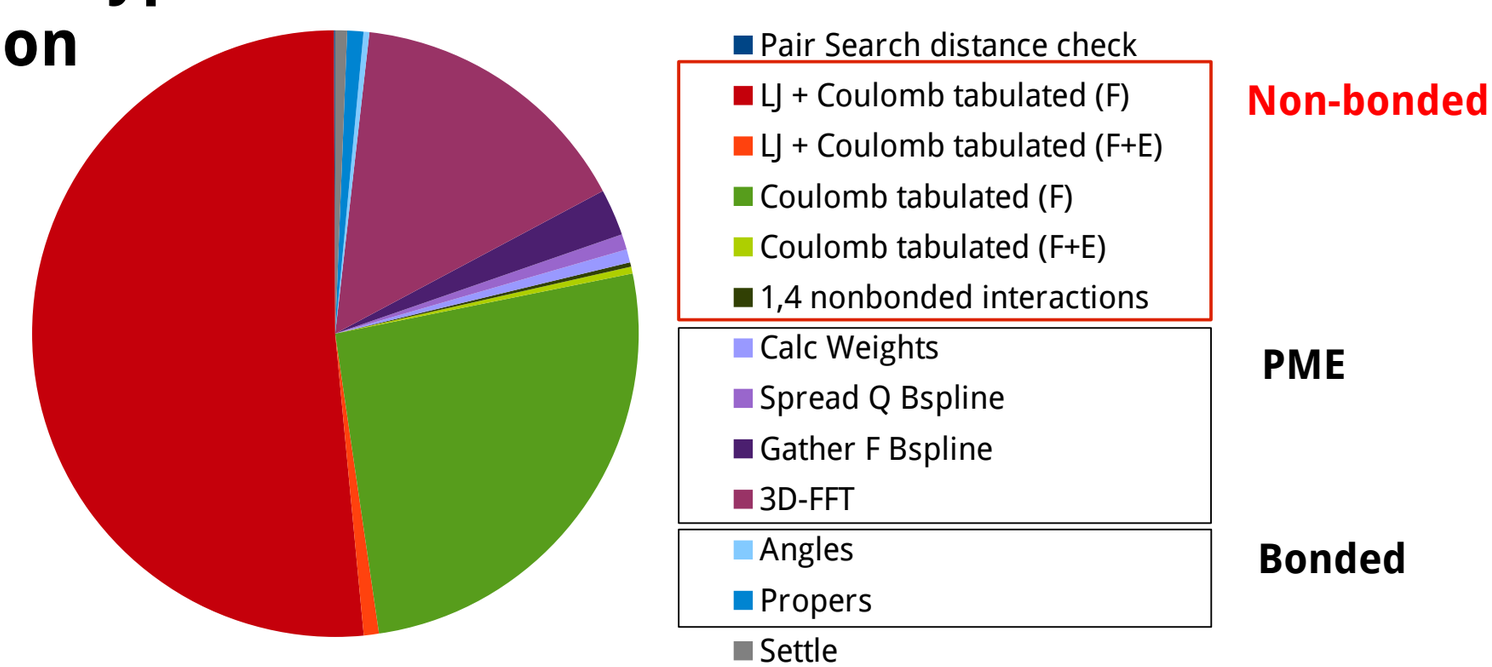
time- & length-scale challenge

strong / weak scaling

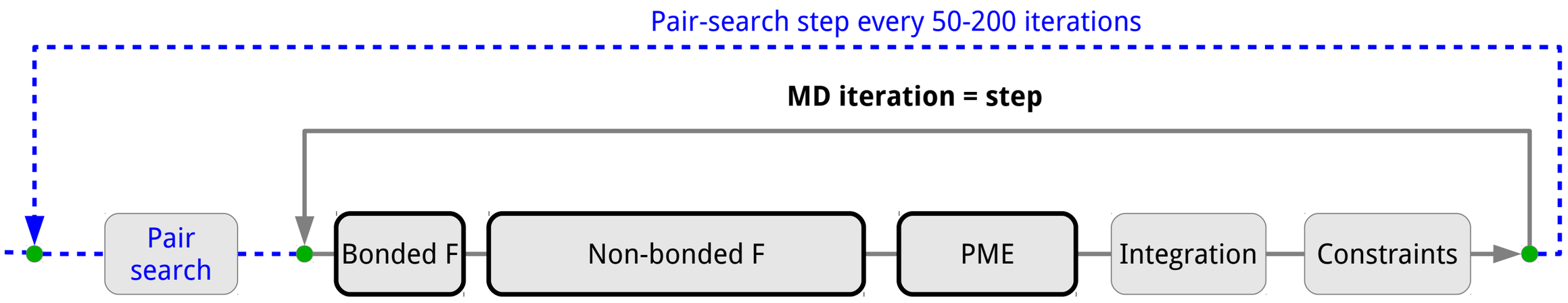
Computational challenge

- Why is MD compute-intensive?
  - Every MD time-step: 10^6 -10^8 Flops
  - Typical MD simulation: 10^6 -10^9 steps

FLOPs in a typical simulation



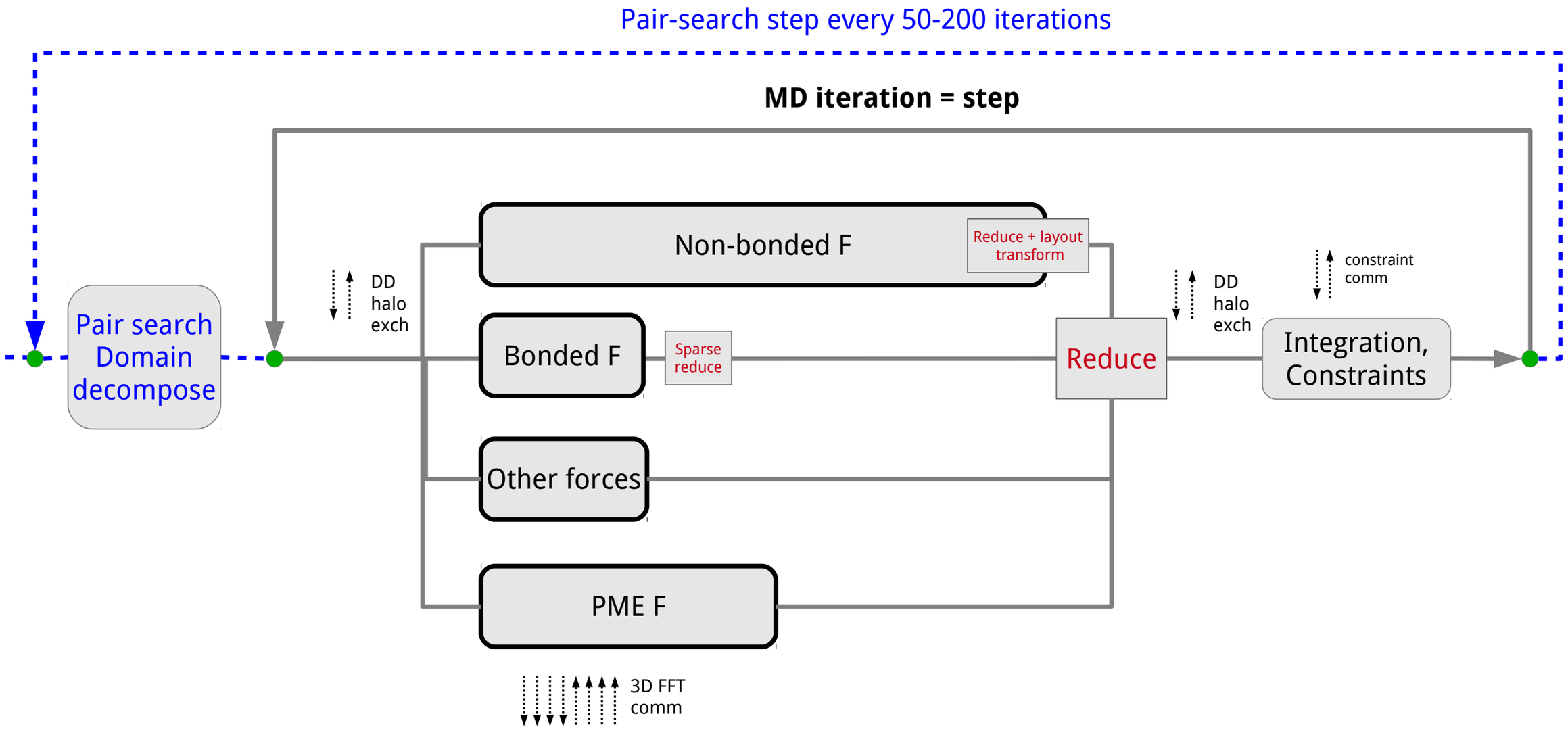
Molecular dynamics step



~ milliseconds or less

Goal: do it as fast as possible!

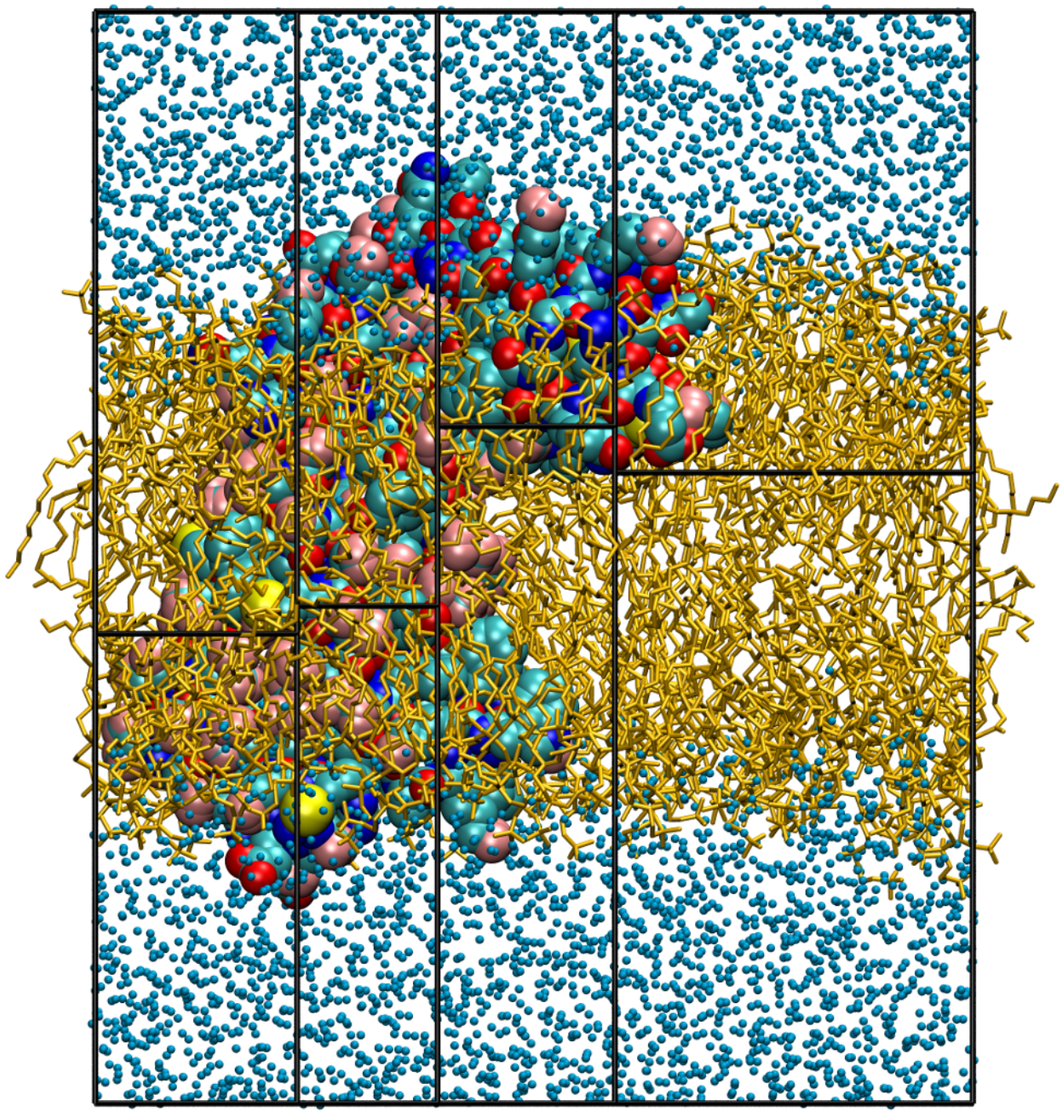
Molecular dynamics step

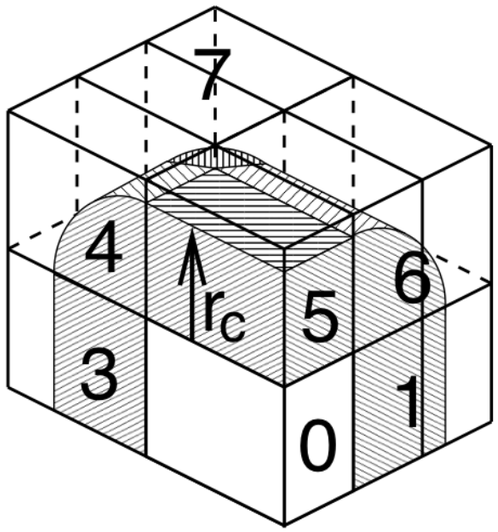


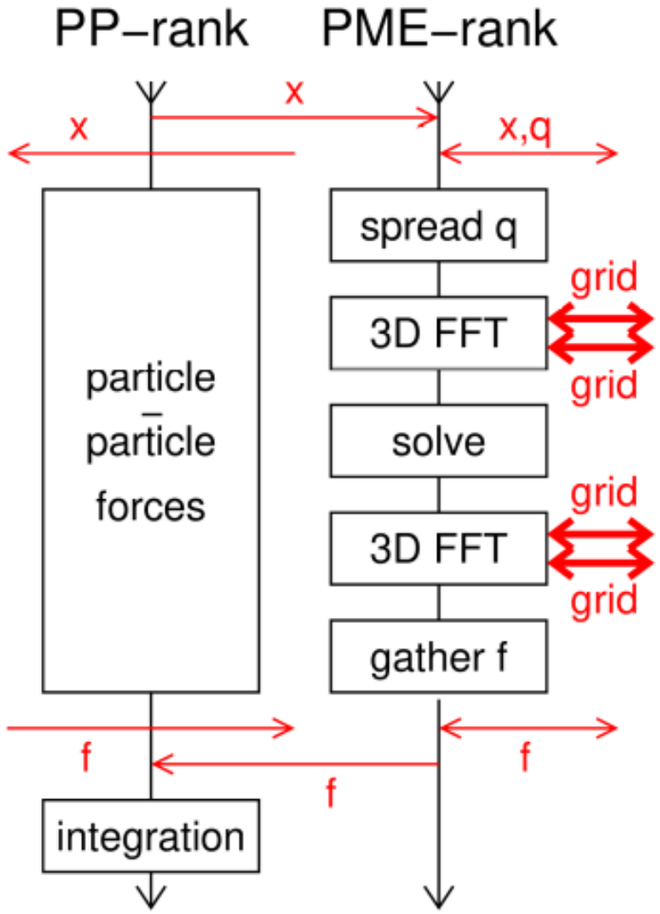
GROMACS parallelization

Parallelism exploited on multiple levels:

SIMD / threading / NUMA / async offload / MPI







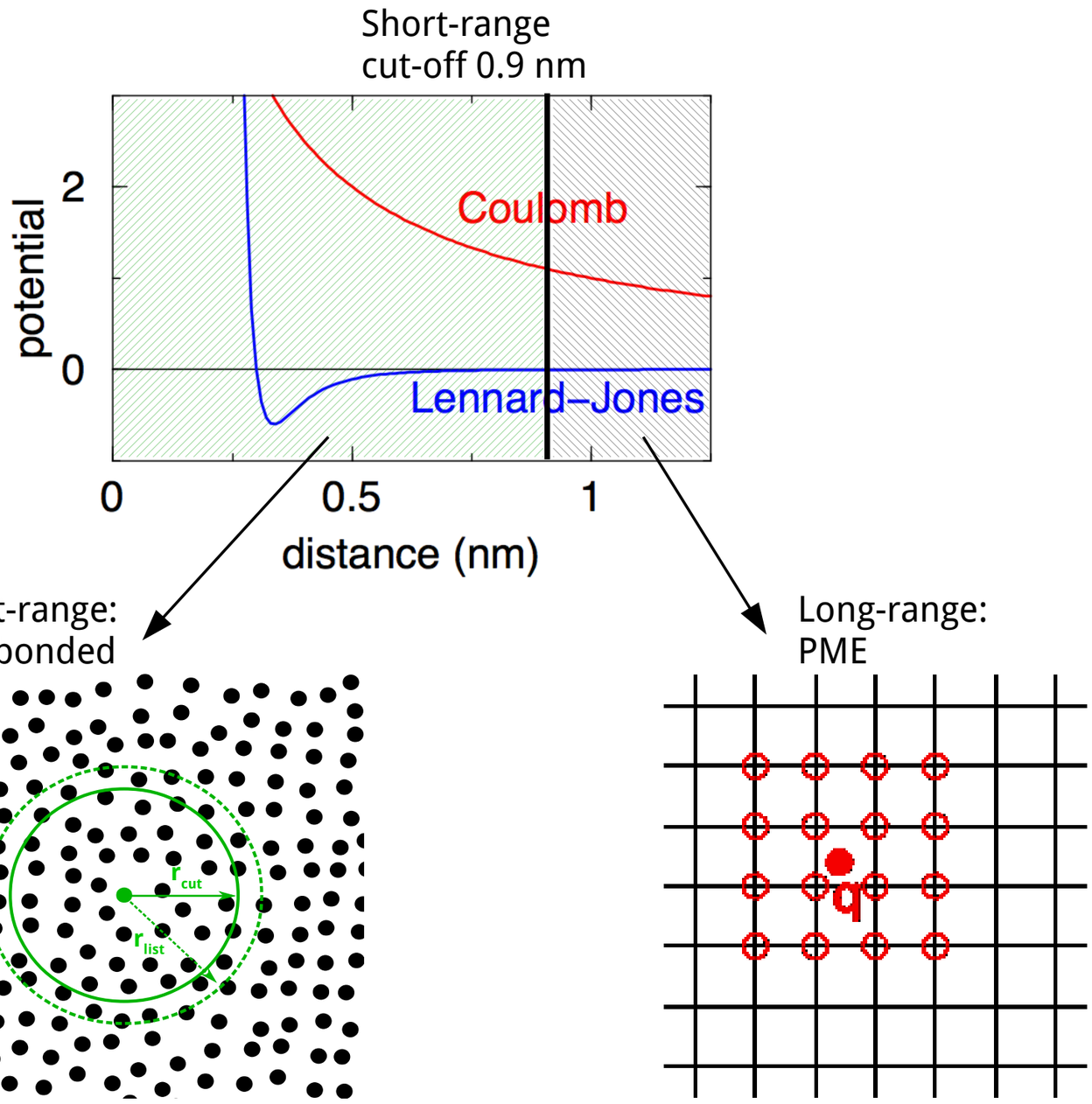
Non-bonded interactions

- Lennard-Jones:
  - decays fast, use cutoff
- Coulomb
  - decays slowly, need long range
- Short-range:

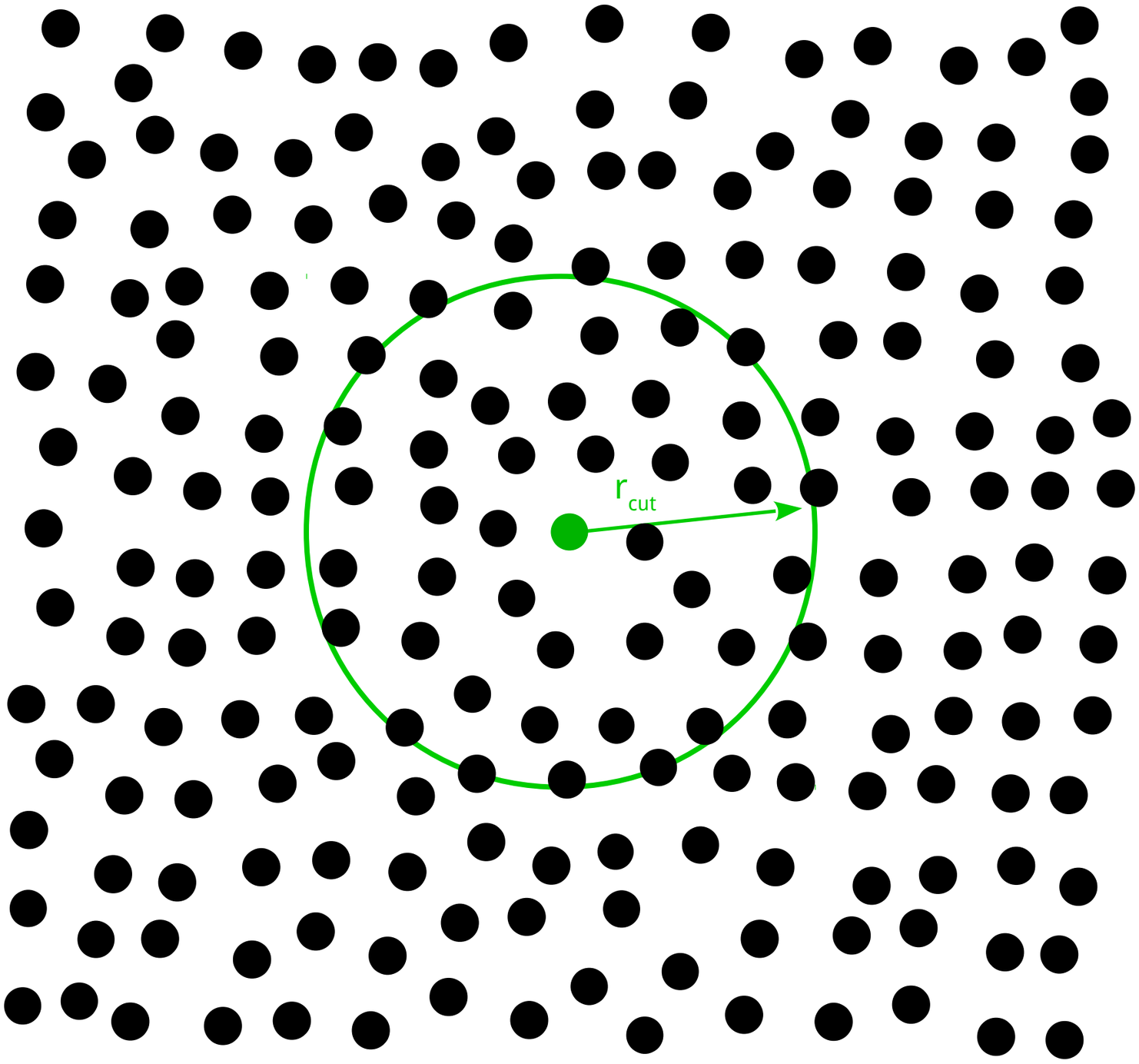
  calculate all intearctions

  within a spherical cutoff
- Long range:

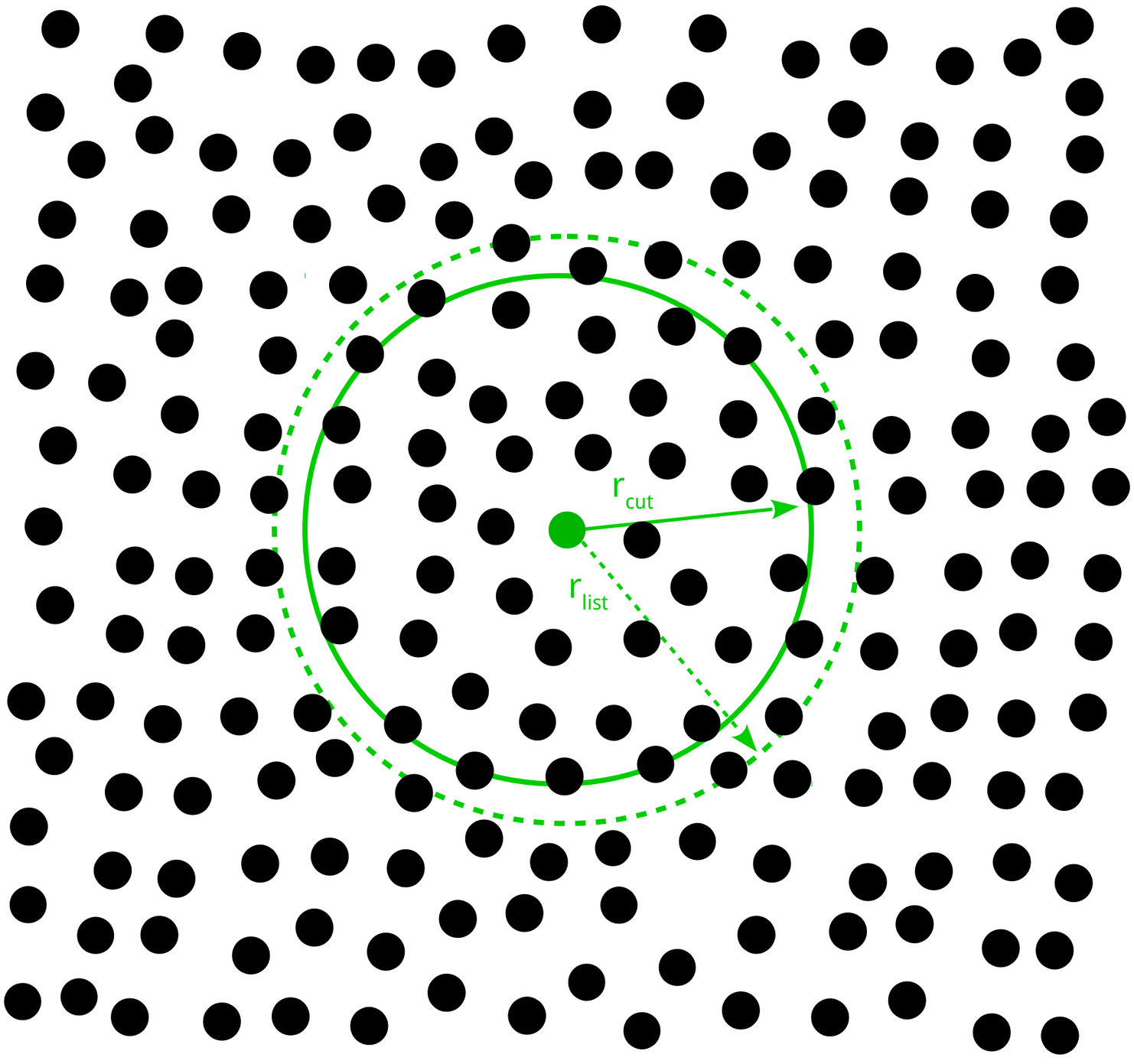
  PME grid-based solver



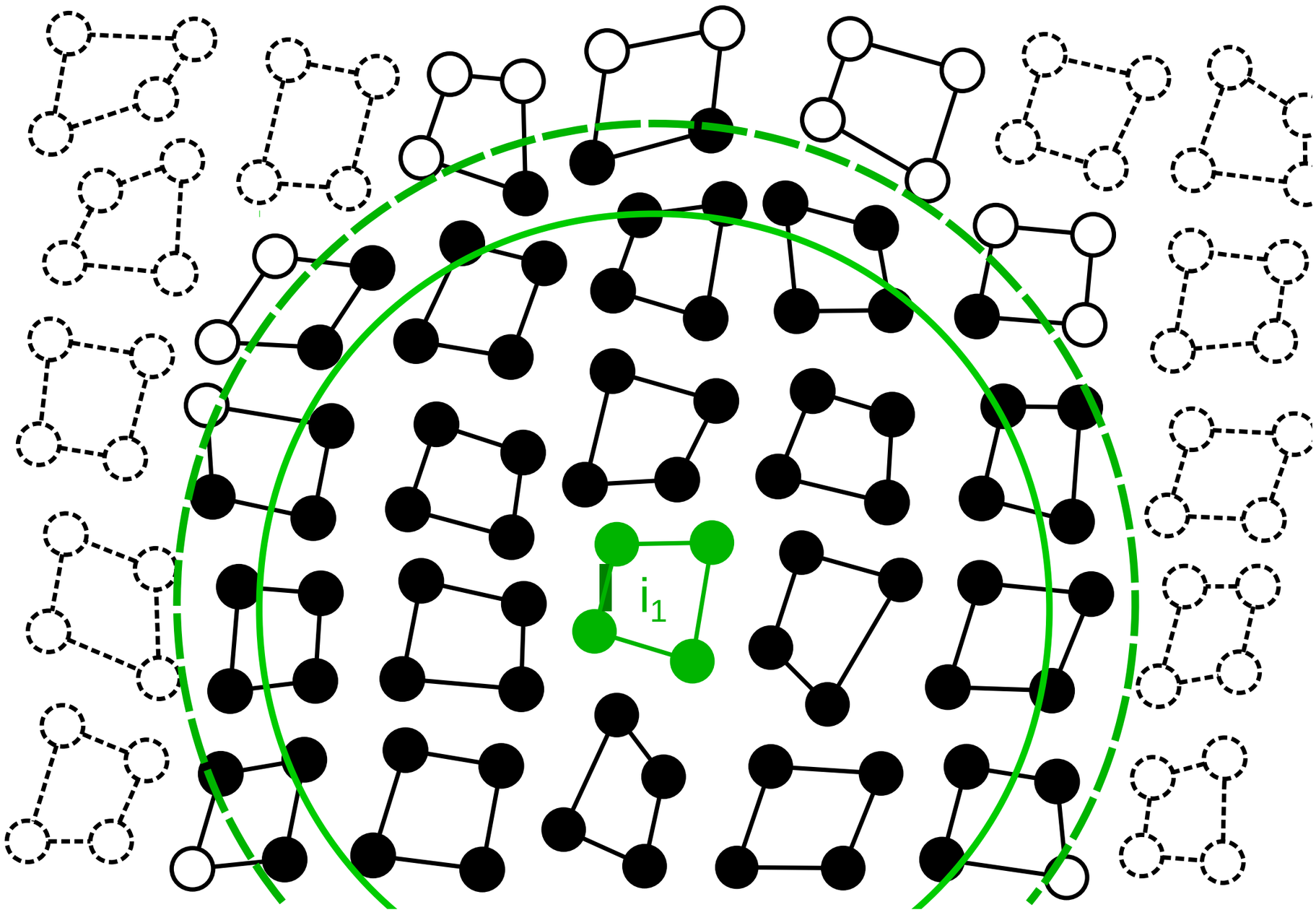
Pair interactions



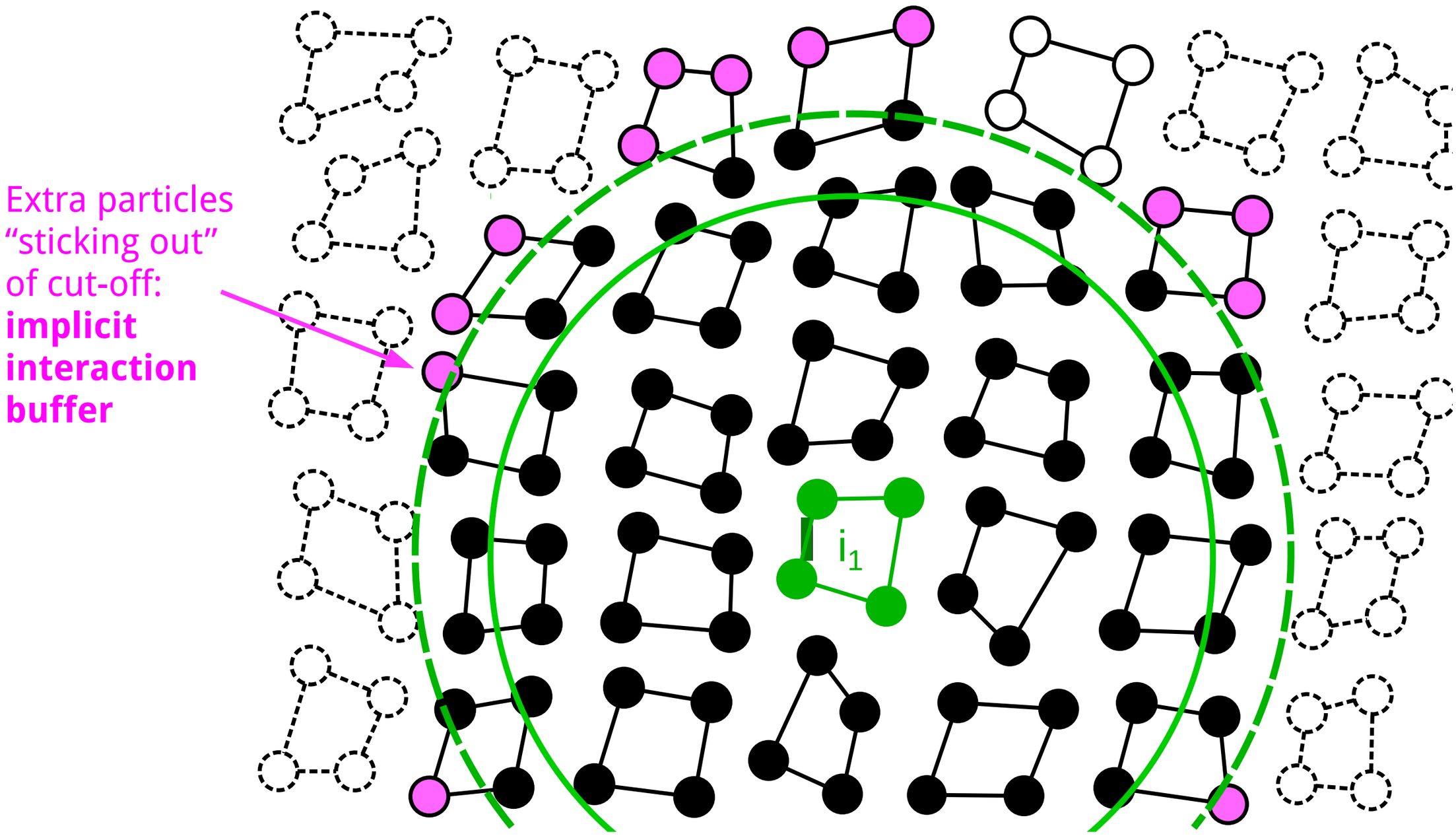
Pair interactions: Verlet algorithm



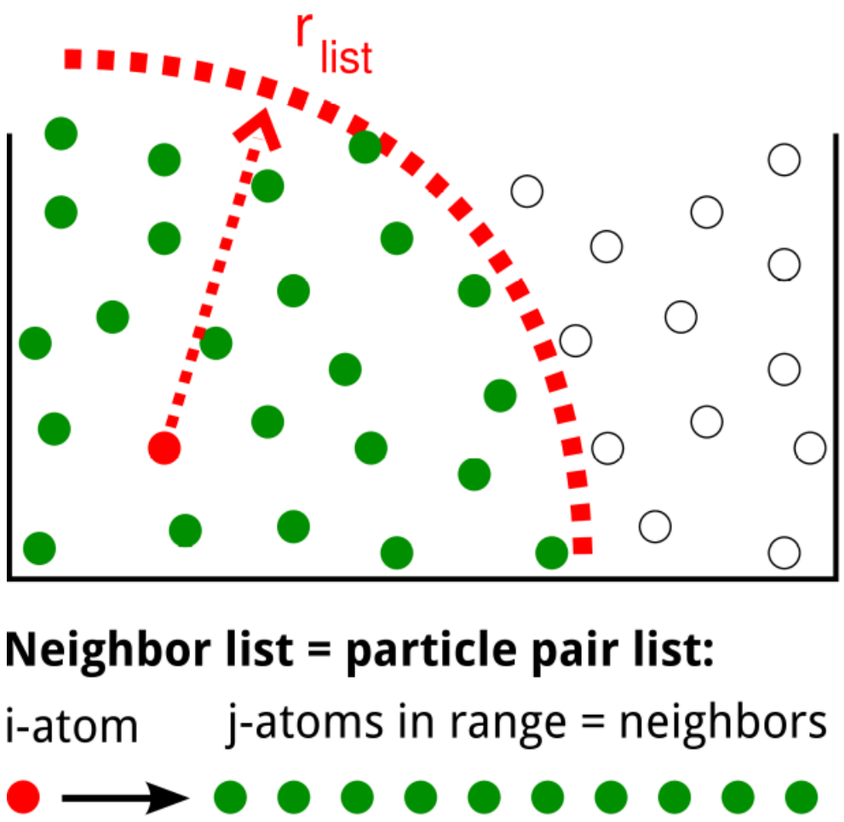
Cluster setup:
ragularize data!

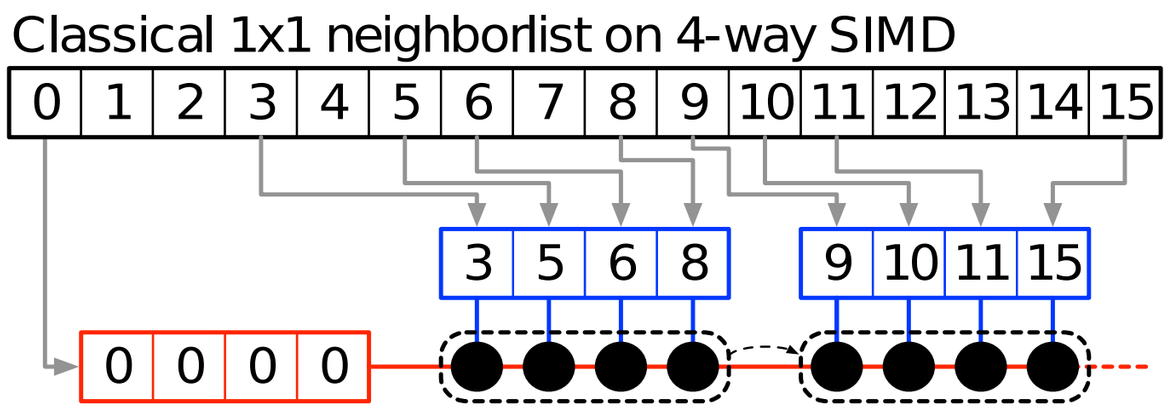


Cluster setup implicit buffer: useful overhead



Pair force calculation: SIMD-parallel traditional algorithm

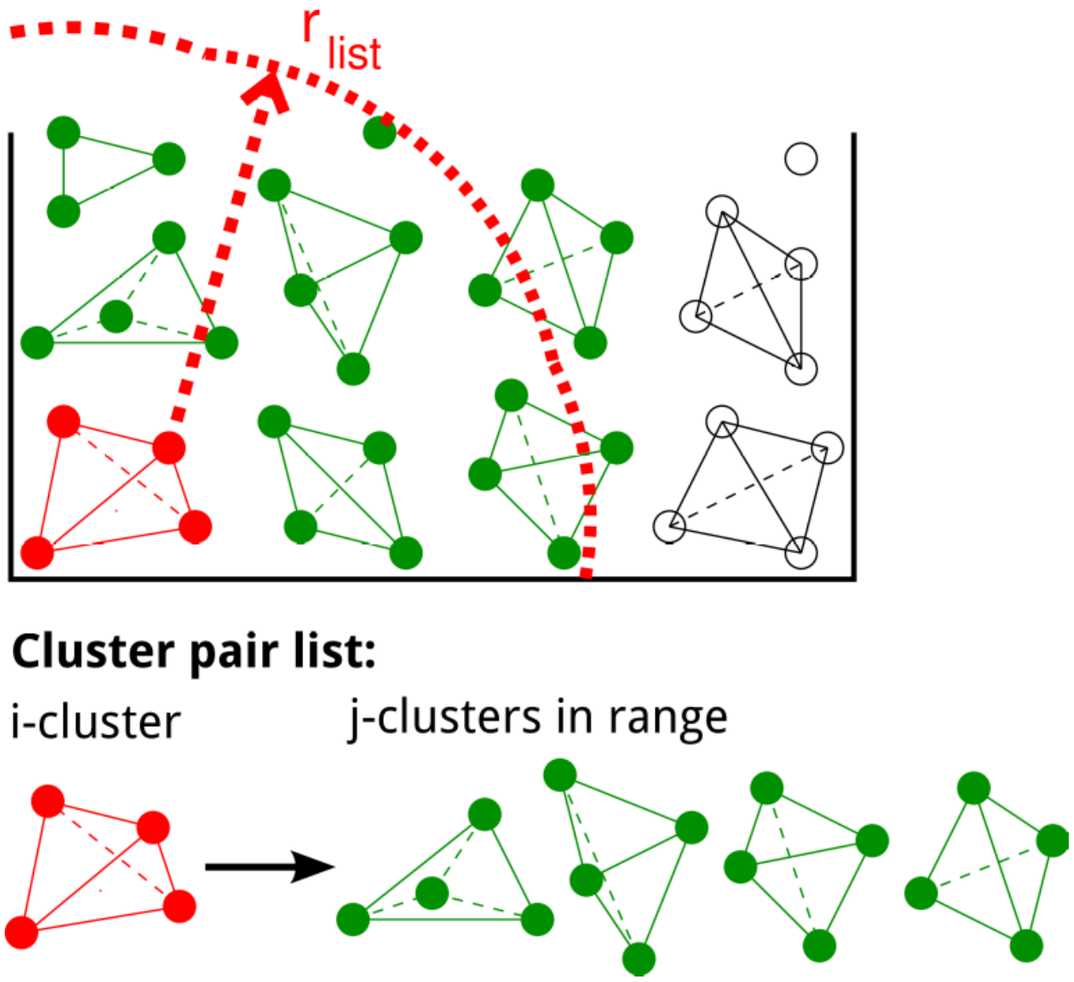


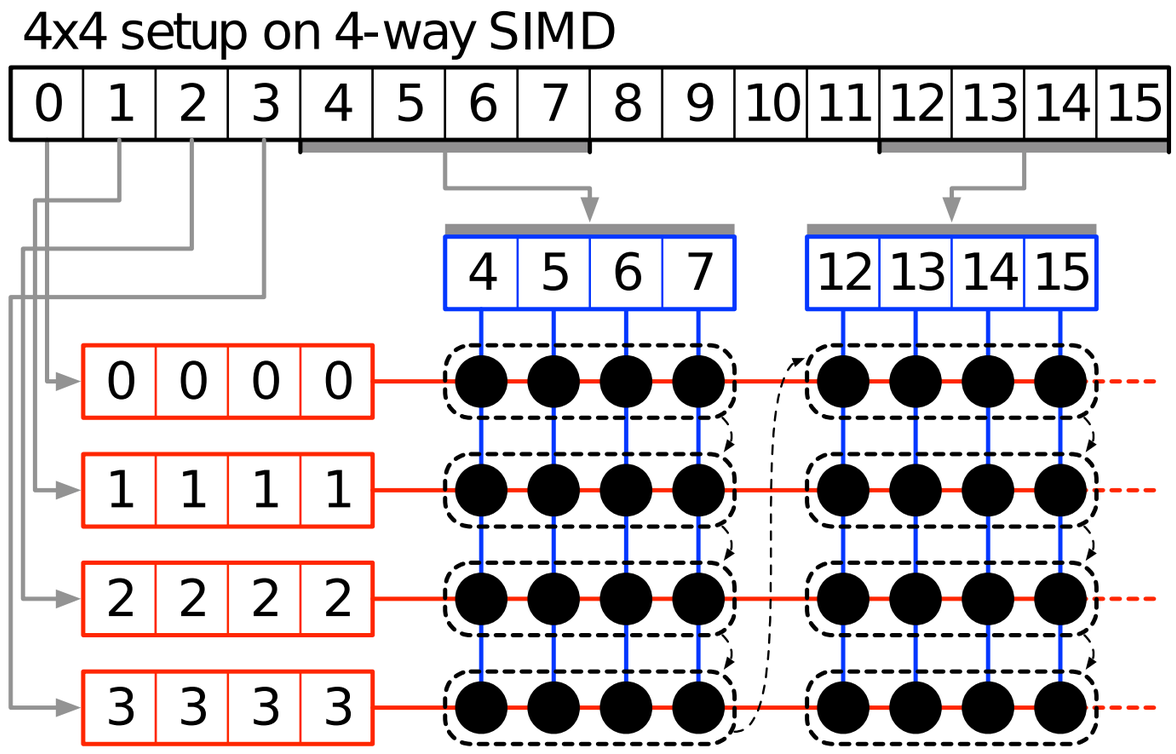


Traditional algorithm: cache pressure, ill data reuse, in-register shuffle bound

Bad for SIMD!

Pair force calculation: SIMD-parallel cluster algorithm



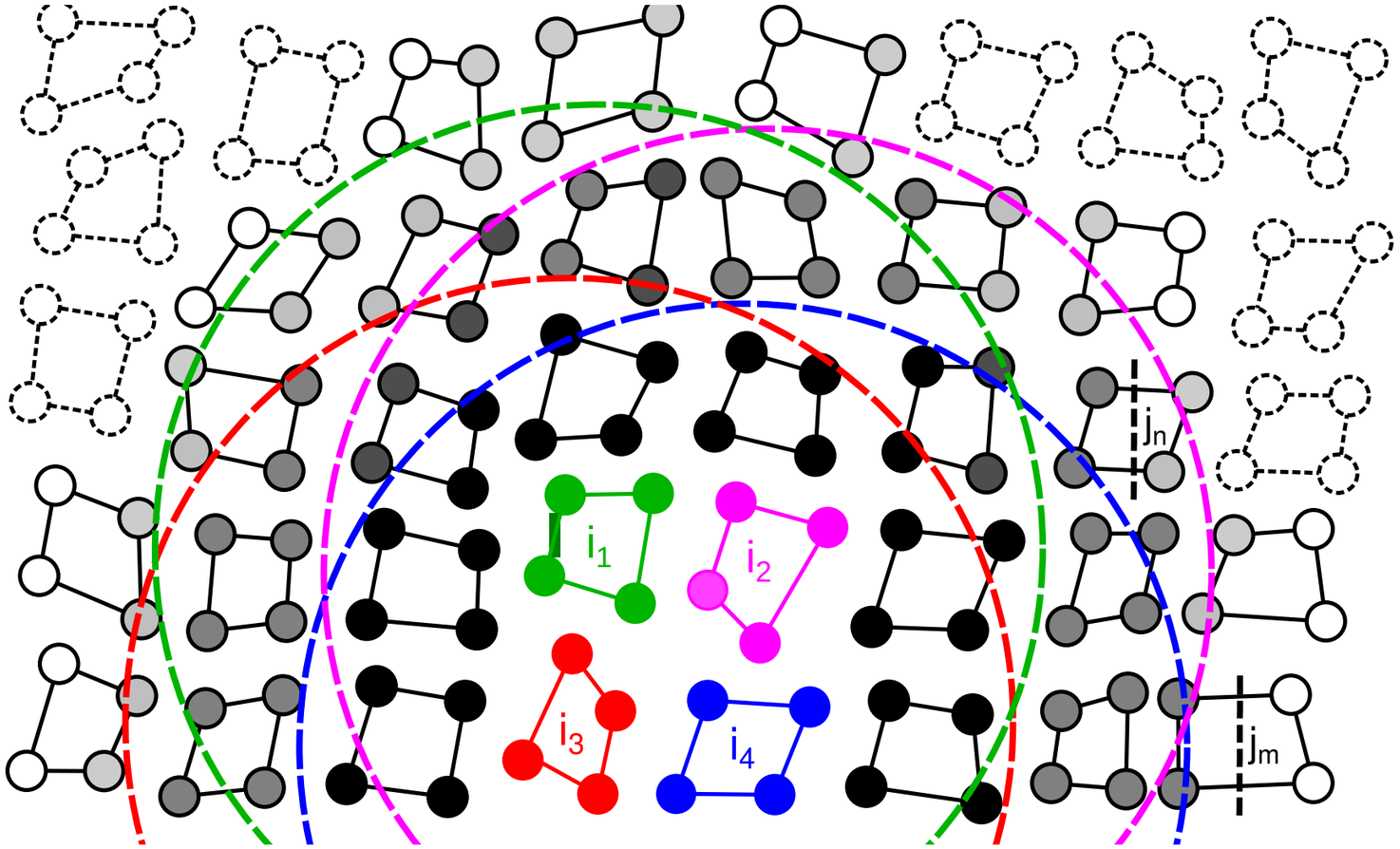


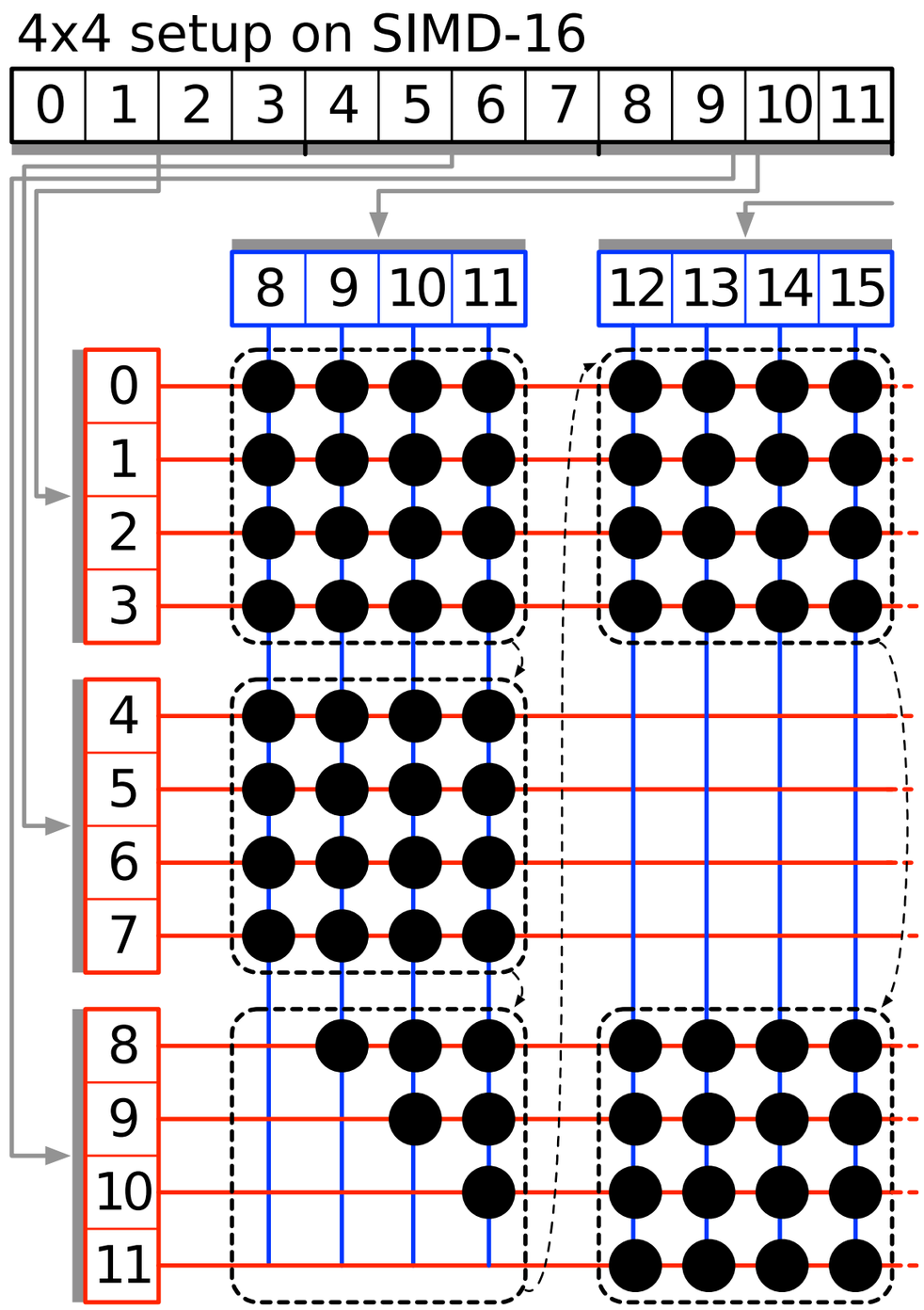
Cluster algorithm for SSE4, VMX, 128-bit AVX:
cache friendly, 4-way j-reuse

Cluster sizes are the “knobs” to adjust for a specific arch:

- SIMD width
- cache-efficiency, data reuse
- arithmetic intensity

Super-cluster setup for GPUs



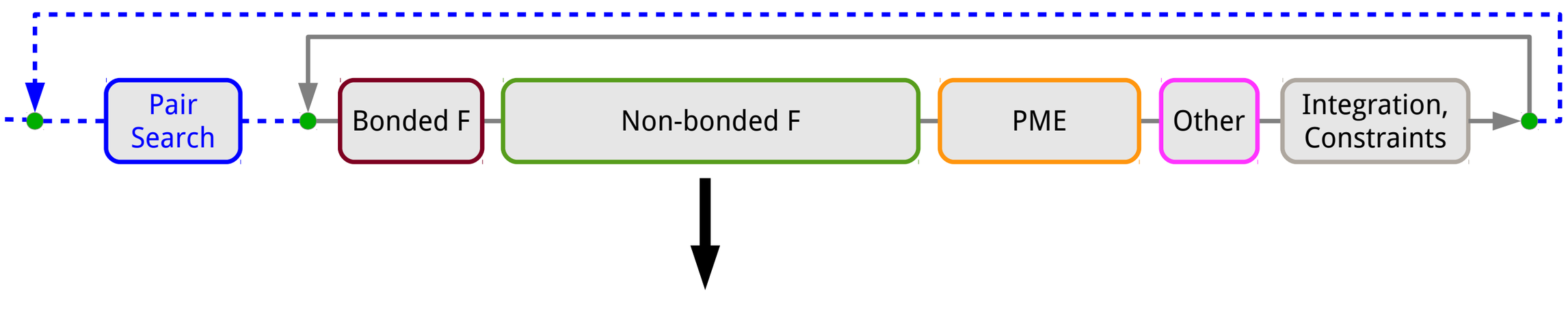


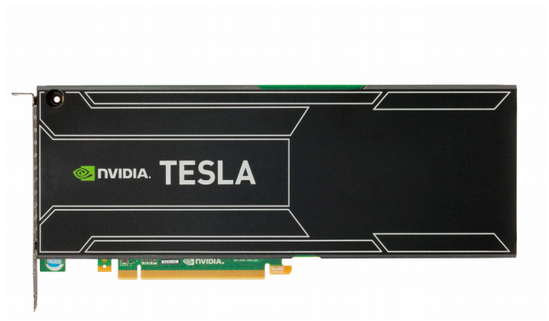
Wider execution: use **hierarchical pair list to avoid work-efficiency cliff**

- joint list for multiple i-clusters
- swap i-j loop order

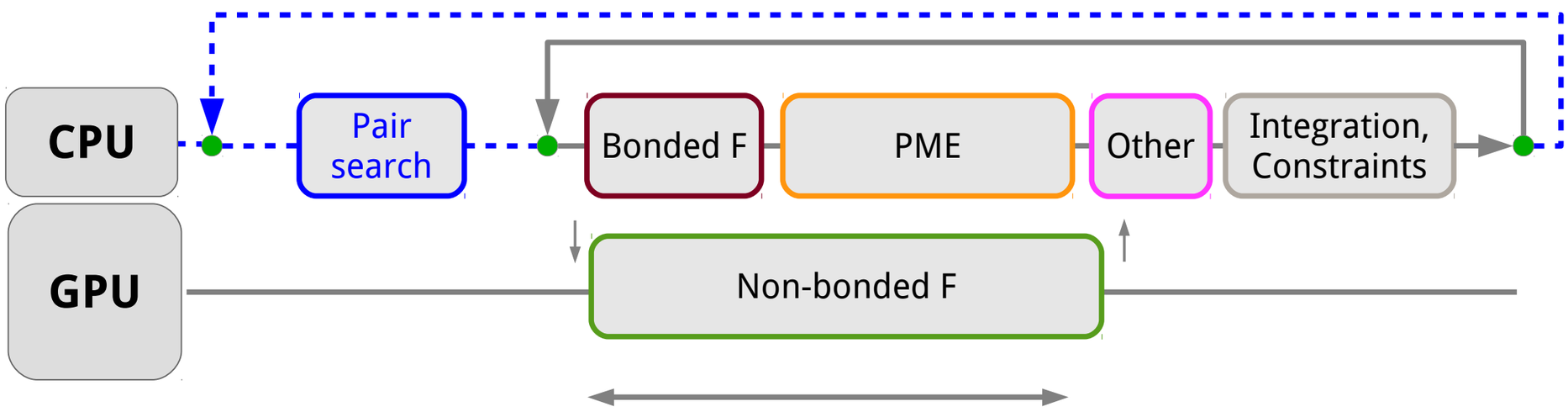
Flavors: NVIDIA 8x4, AMD 8x8, Intel?

GPU offload: **nonbonded pair interactions**
GROMACS v4.6 – v2016



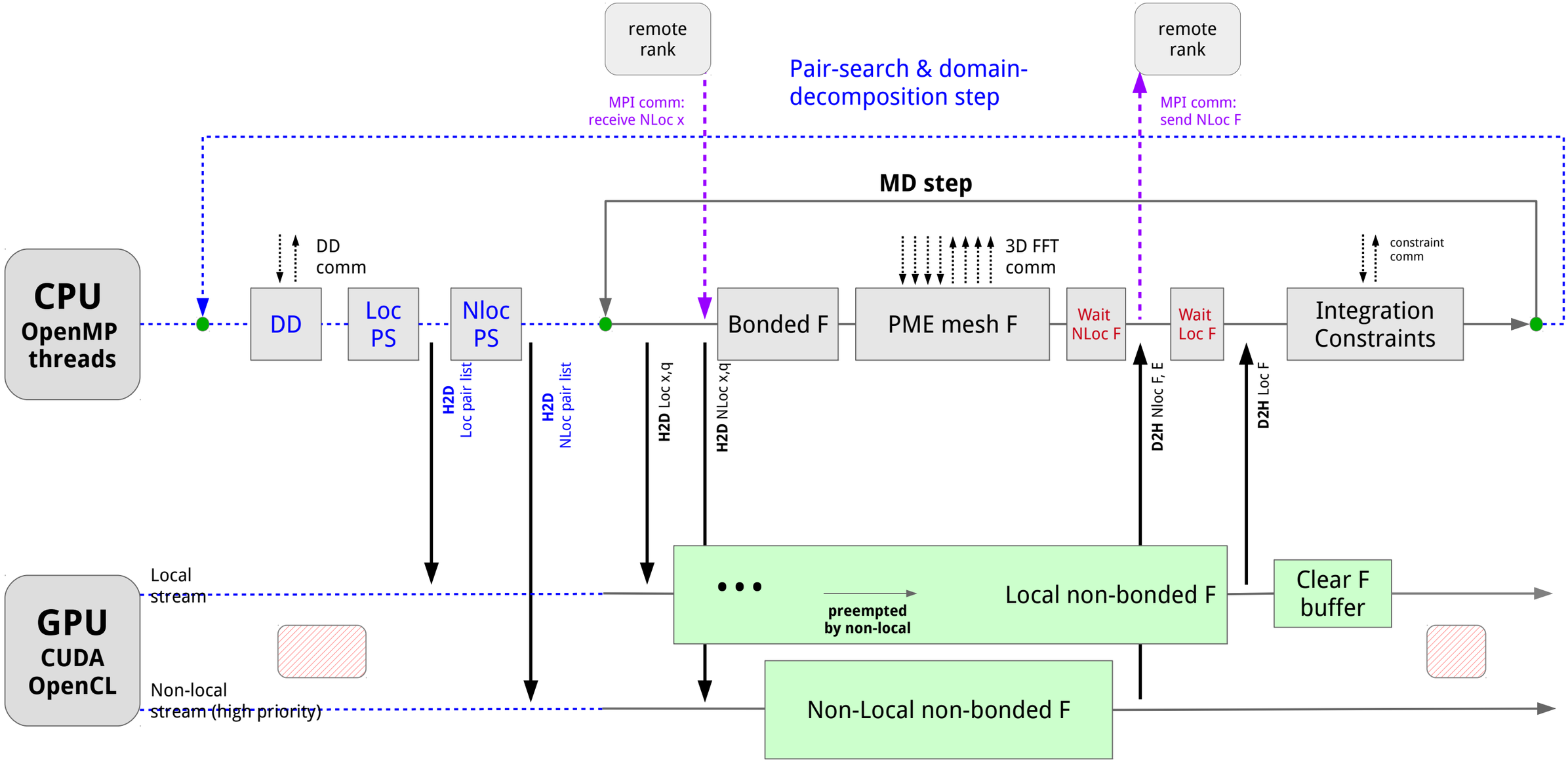


GPU offload: **nonbonded pair interactions**
GROMACS v4.6 – v2016

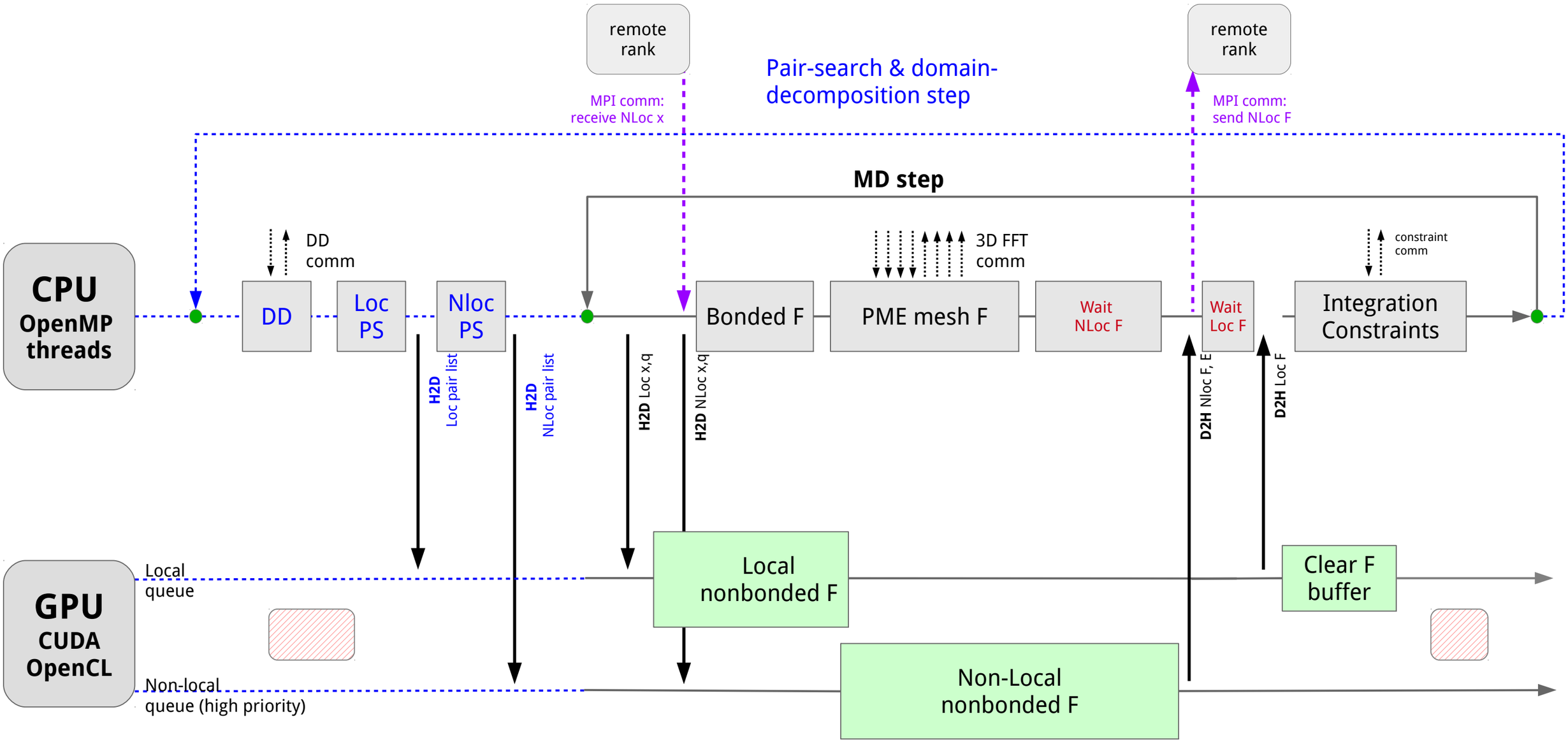


- Initial GPU offload support: CUDA-only (2013)
- OpenCL from v5.1 (2015) – contribution by StreamHPC
  - AMD and NVIDIA GPU support
- v2019: Intel iGPU support (Intel contribution)

Heterogeneous parallel MD: CUDA



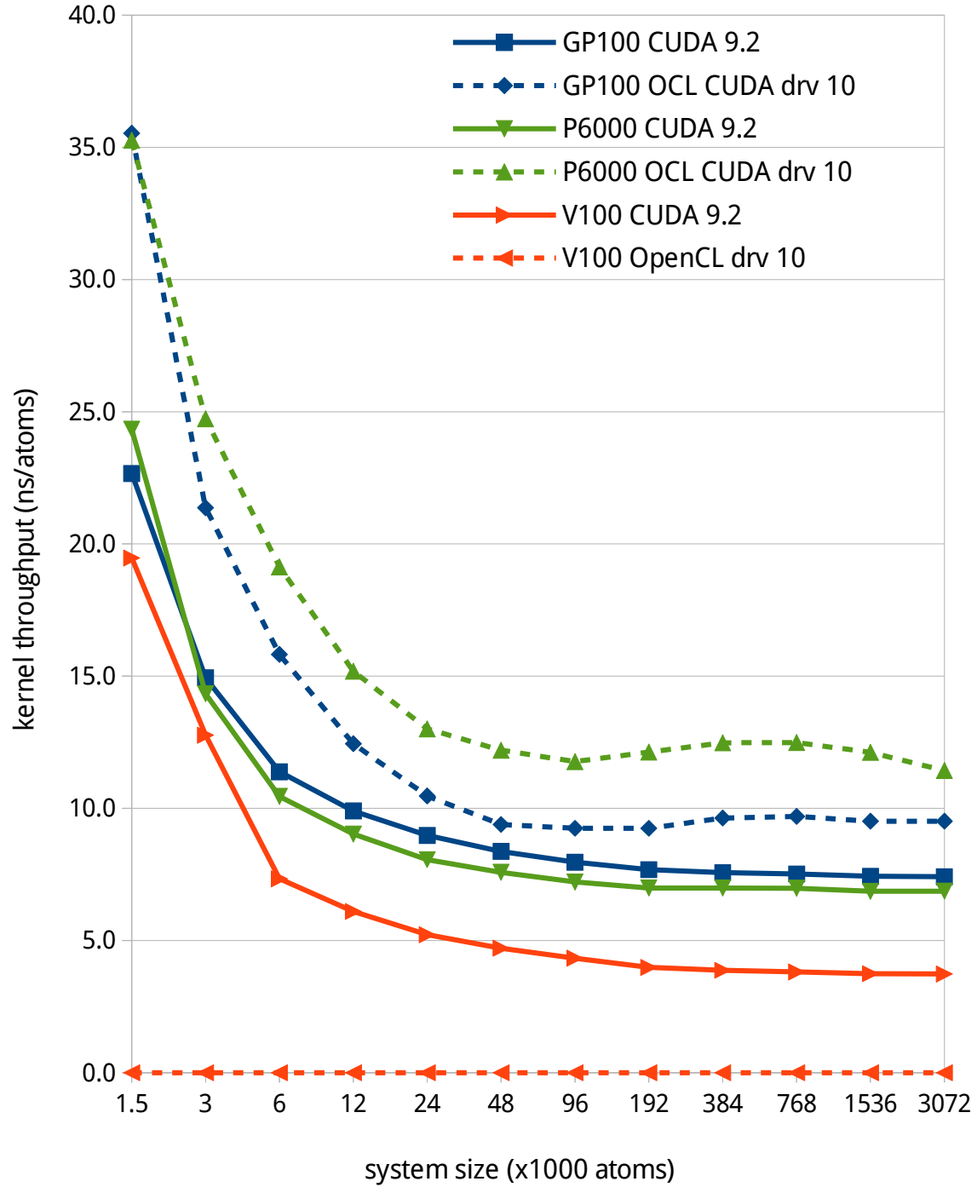
Heterogeneous parallel MD: OpenCL <=2.0

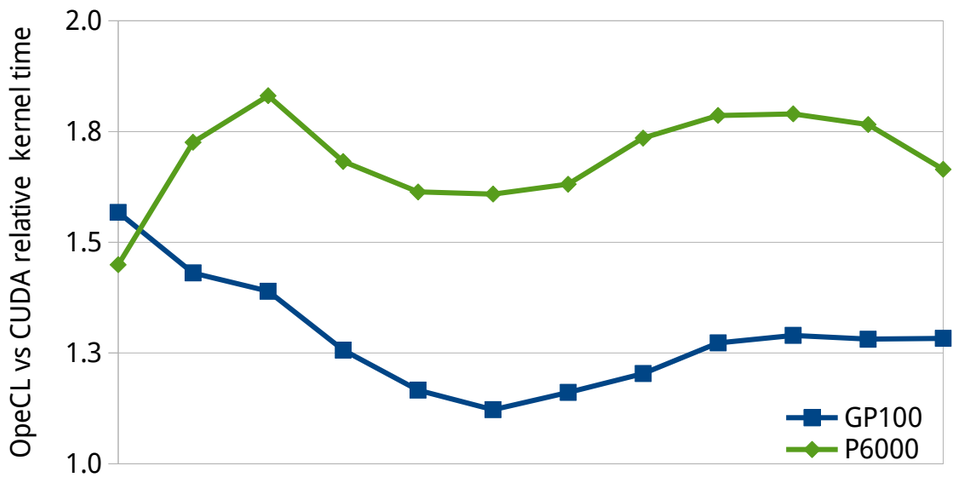


Without **queue priorities** optimizing for halo exchange critical path is challenging!

Nonbonded kernels on “big” GPUs

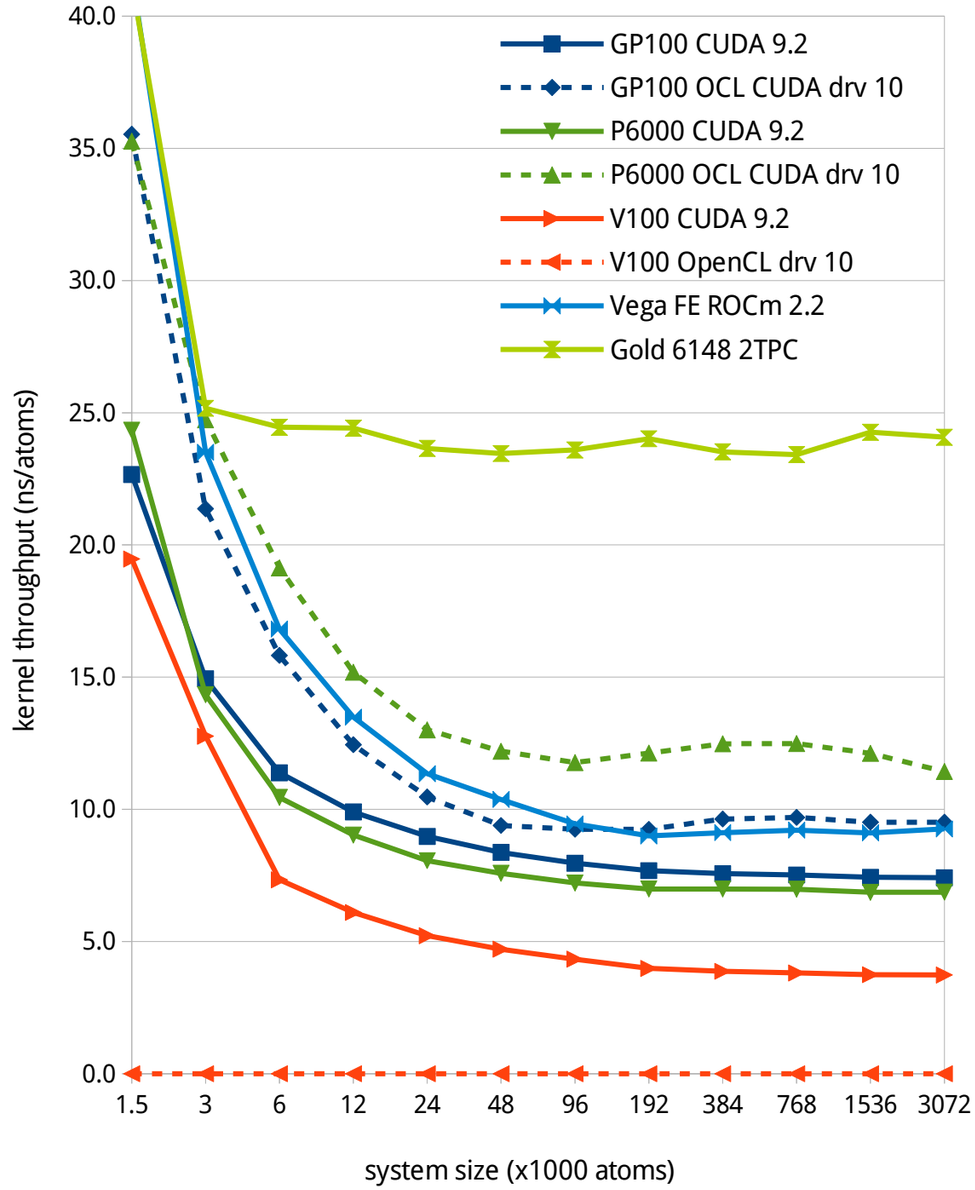
- NVIDIA: state of OpenCL vs CUDA
  - GP100: 1.2-1.5x **slower**
  - GP102: 1.5-1.9x **slower**
  - V100: **does not work**
  - partly due to shuffle/wg reductions





Nonbonded kernels on “big” GPUs

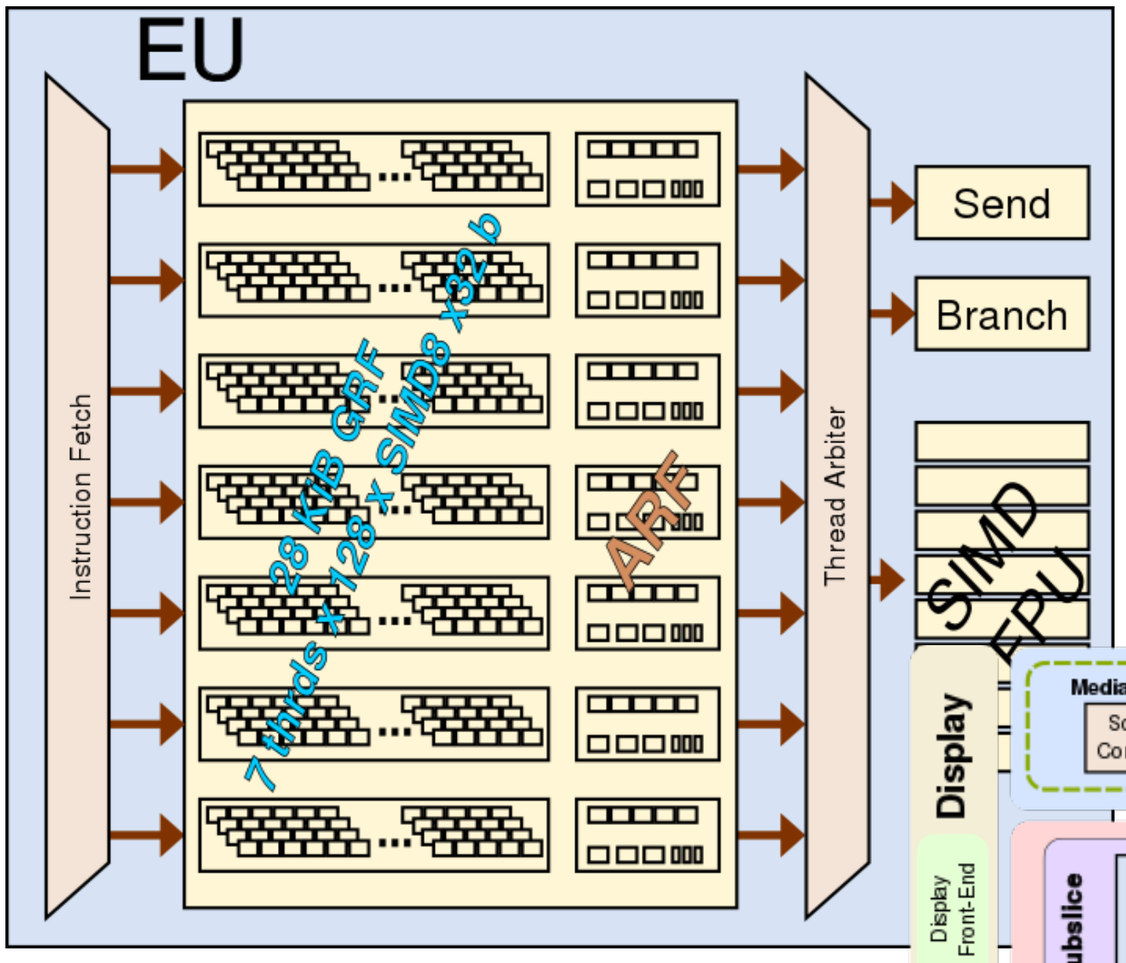
- NVIDIA: state of OpenCL vs CUDA
  - GP100: 1.2-1.5x **slower**
  - GP102: 1.5-1.9x **slower**
  - V100: **does not work**
  - partly due to shuffle/wg reductions



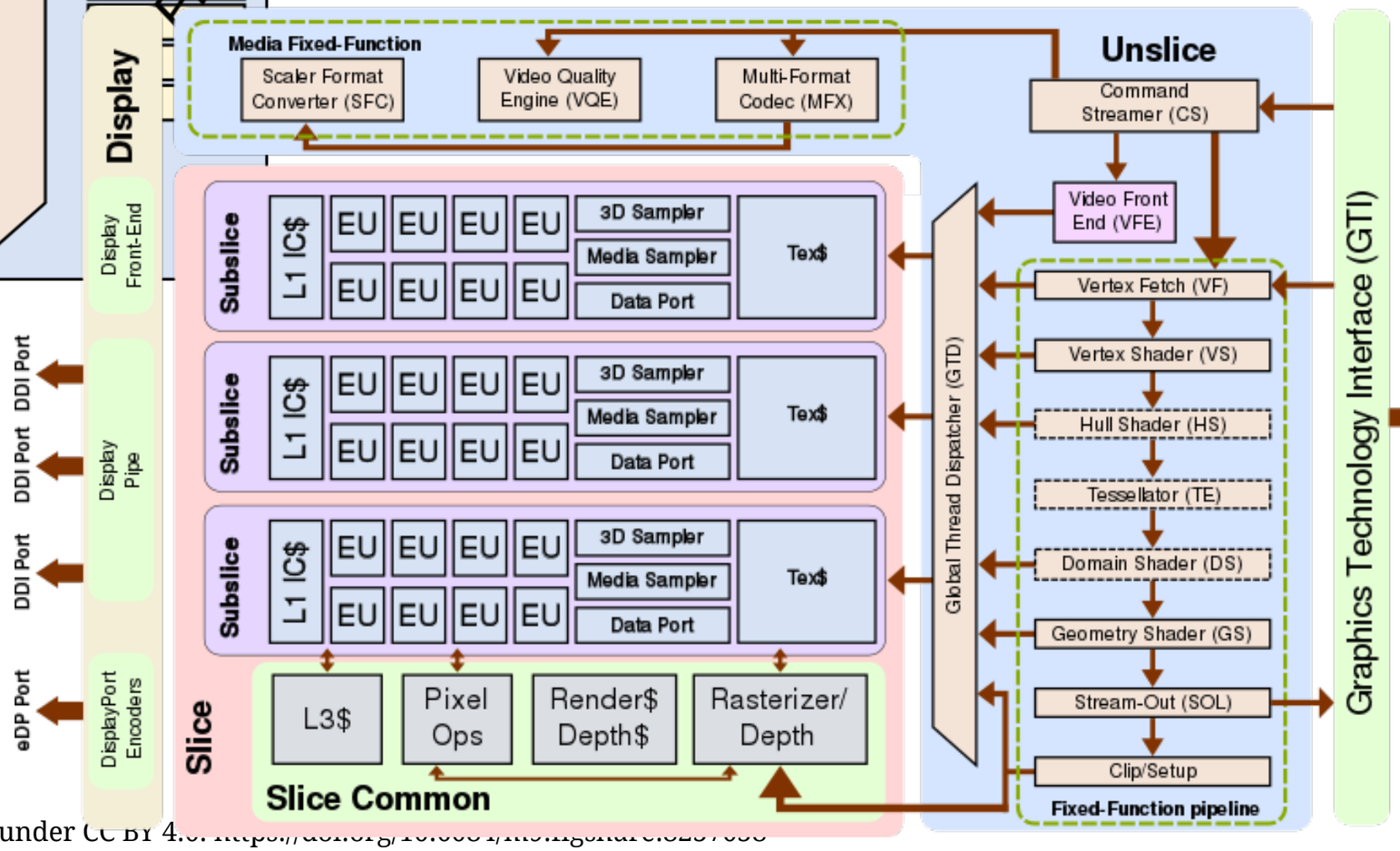
- AMD challenges:
  - ROCm holds great promise
  - compiler regressions are unfortunately still a regular occurrence
  - shuffle-reduction needed

- Scaling to small sizes a challenge especially for strong scaling

Intel Gen9/9.5 arch



- 128 SIMD-8 registers
- 4 issue/clock (2 FPU)



Intel iGPU cluster pair kernel

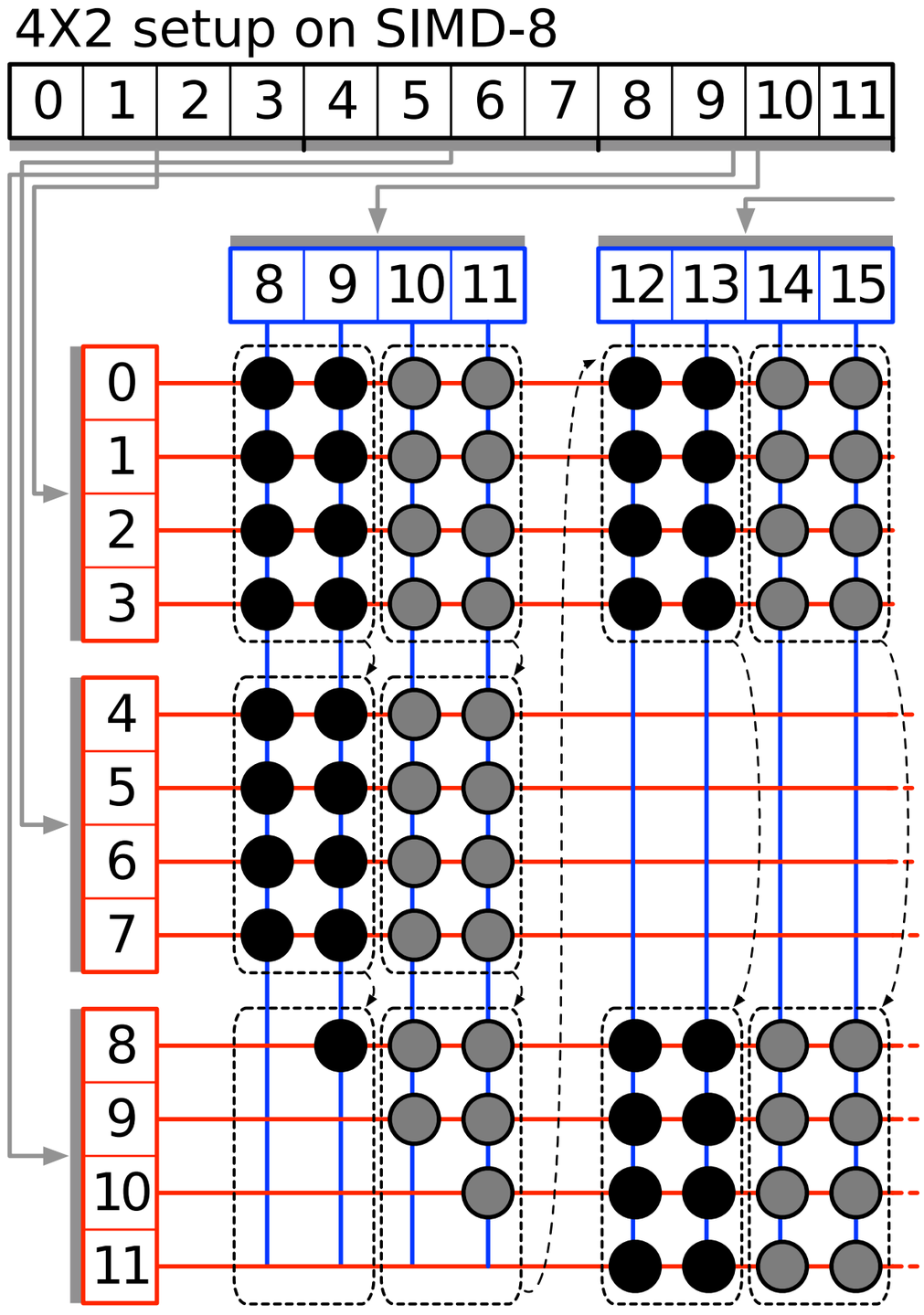
- Cluster-setup:

  search 4x4, prune to 4x2 for force

  (parallel work efficiency advantage)

- Kernels:
  - SIMD8 4x2 (register limited)
  - 8-way super-clustering
  - subgroup shuffle reductions

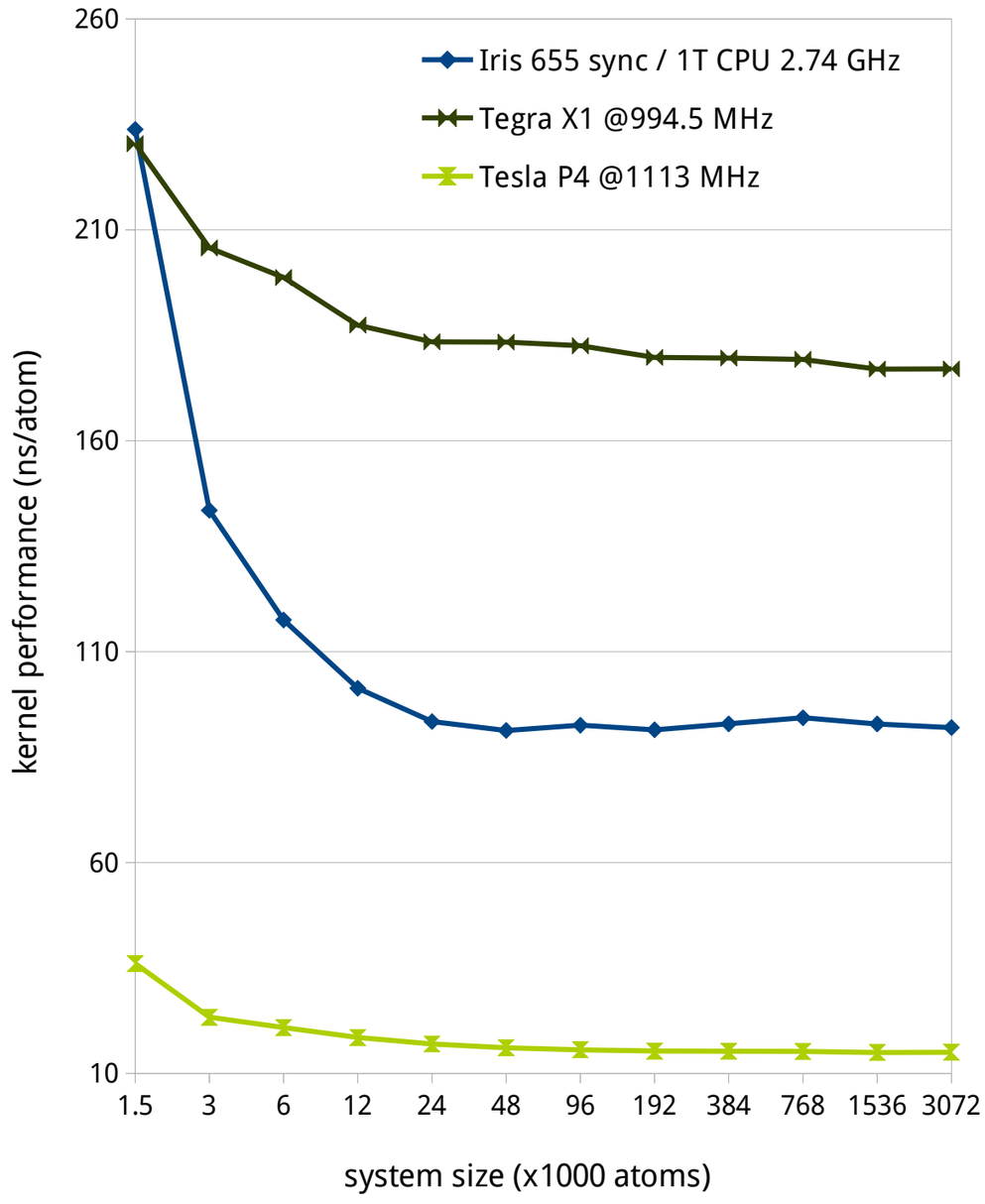
- Further optimization planned later (after we know whether Xe is similar)



Gen9.5 nonbonded kernel performance

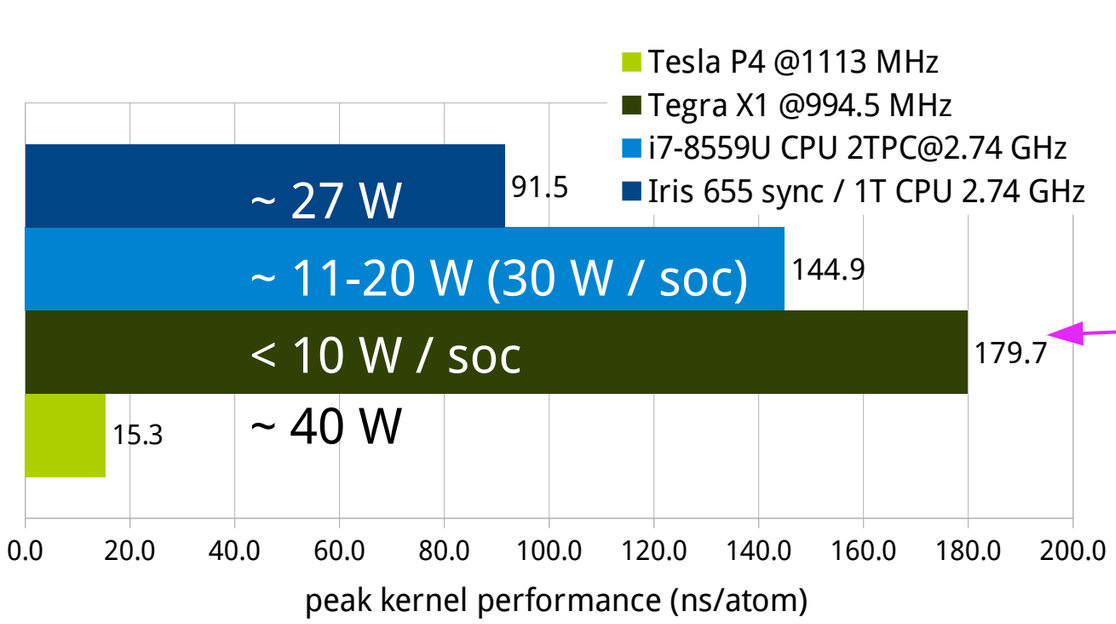
- Core i7-8559U
  - CPU
    - 4C @ 2.7 Ghz base, 4.5 Ghz max turbo
  - GPU: Iris Plus 655 300-1200 MHz
    - GT3e, 24 EU
    - up to 883 GFLOPs SP
  - 28 W TDP / SoC

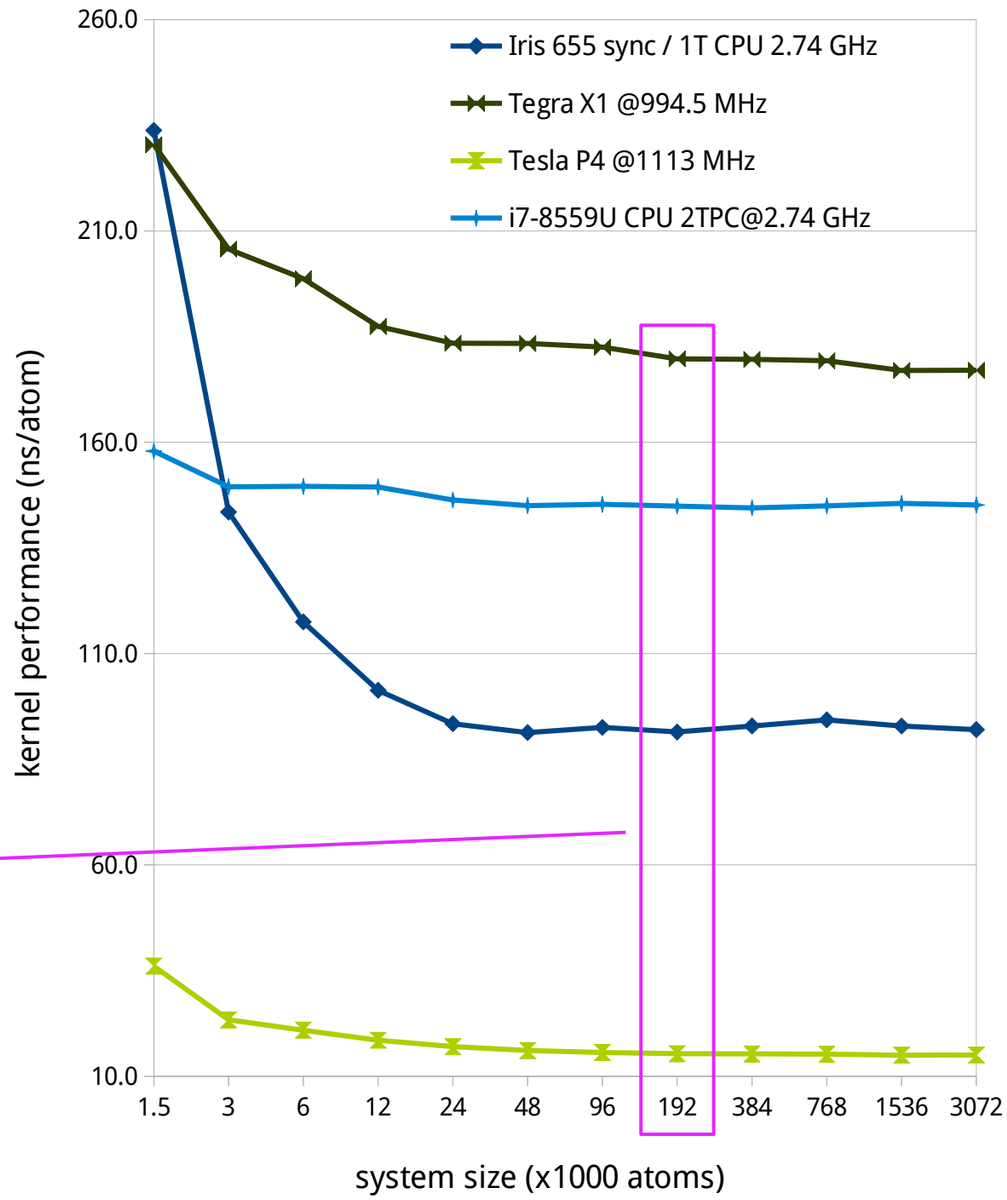
- Performance evaluation setup
  - blocking kernel
  - fixed CPU clock (at base)



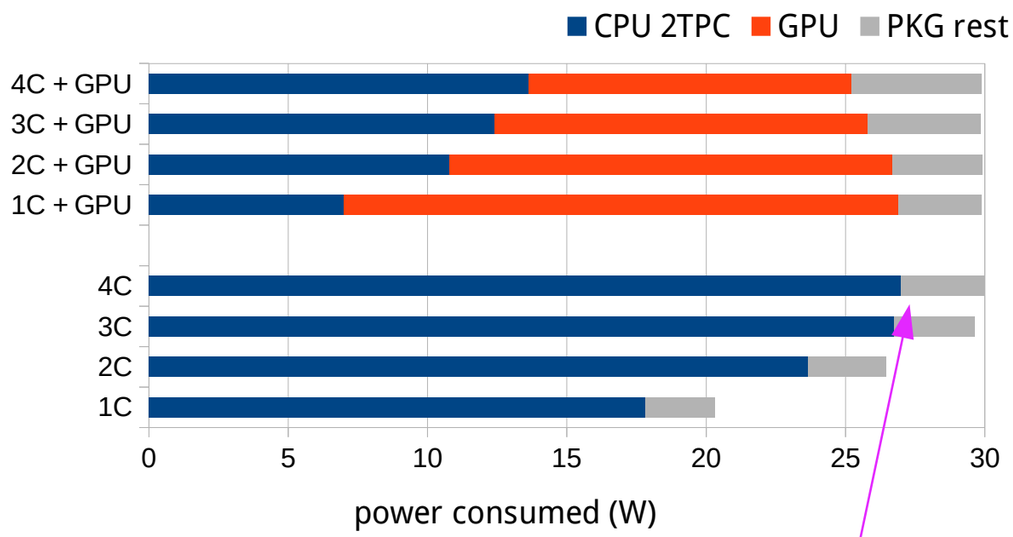
Gen9.5 nonbonded kernel performance

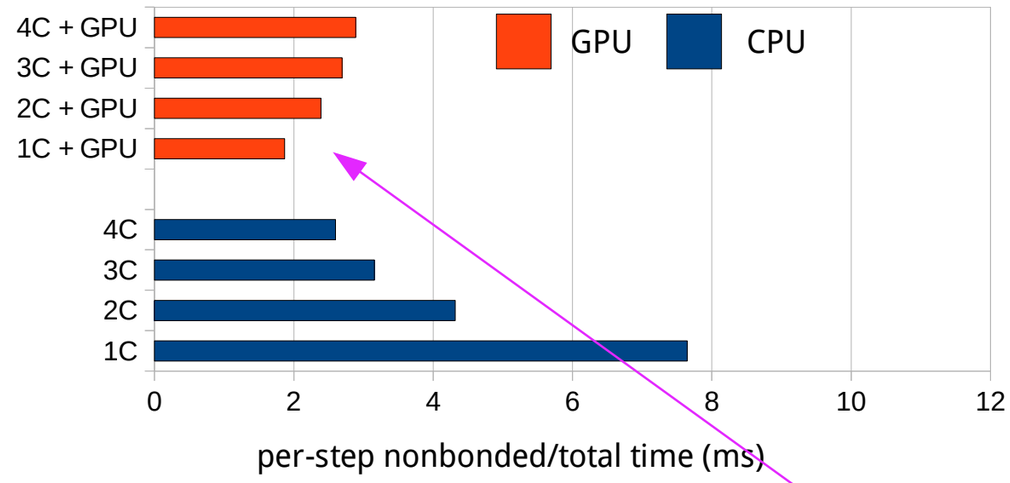
- Perf peaks around 20k atoms/GP
  - similar to CPU
  - efficiency drops like on other GPUs
-





SoC: CPU + GPU share TDP





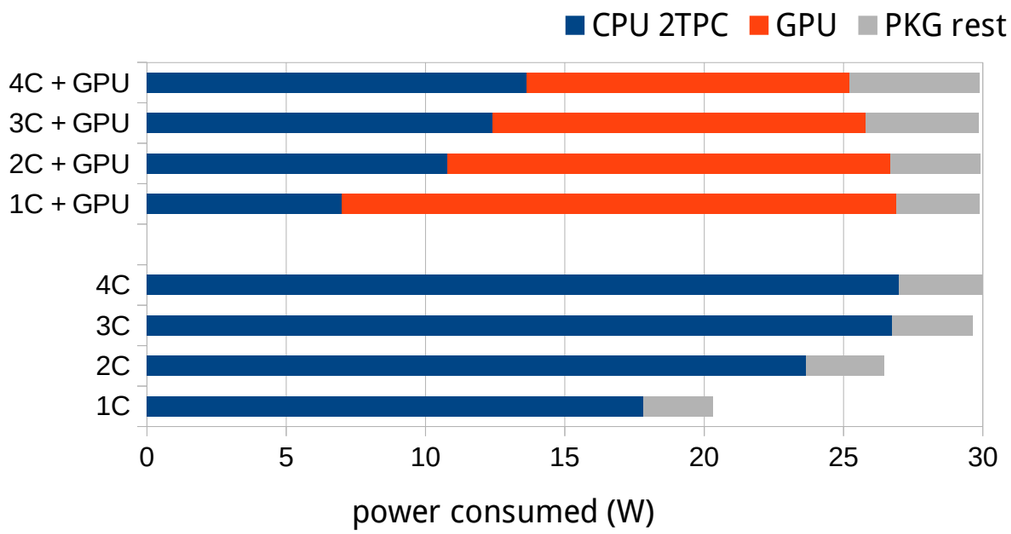
Turbo boost

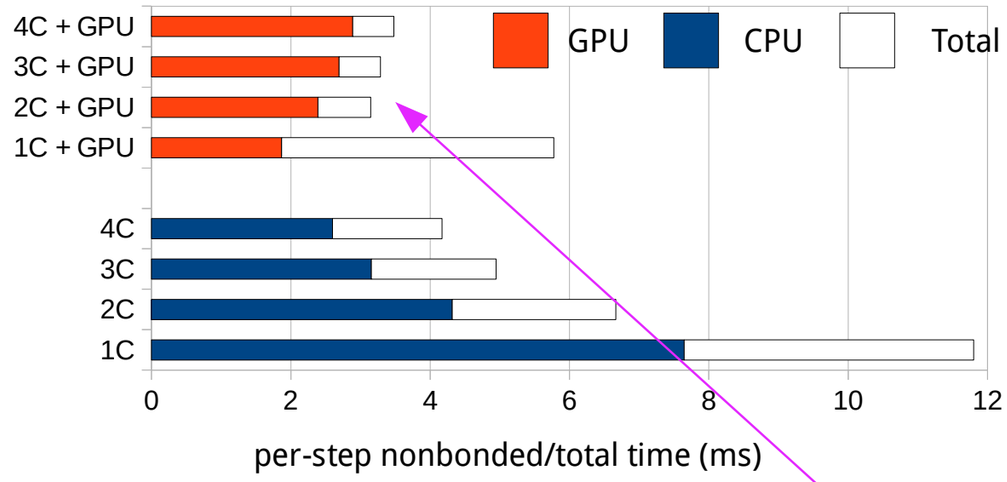
the fewer CPU cores
the more of the TDP GPU gets

Power consumption measured using likwid.

Default CPU clocks, turbo boost on, performance governor.

SoC: CPU + GPU share TDP





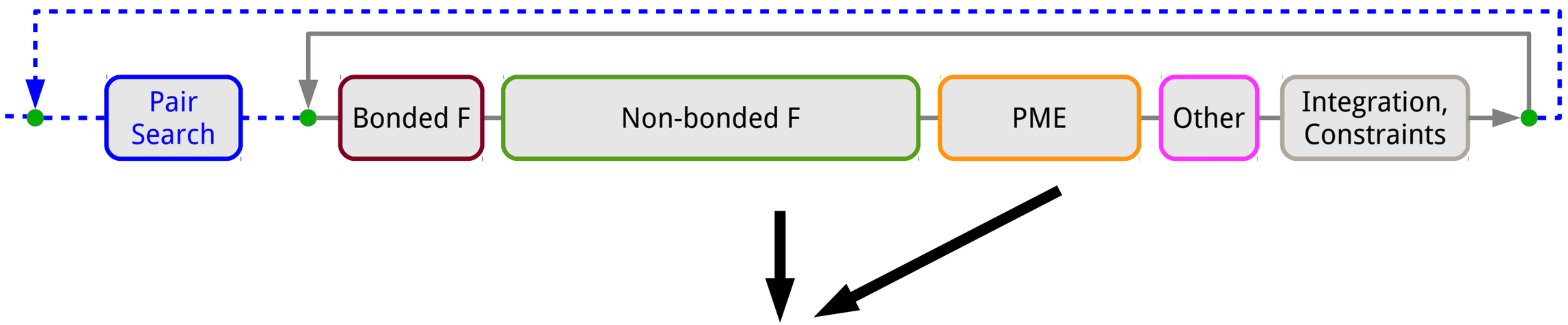
peak performance with
2 cores + GPU

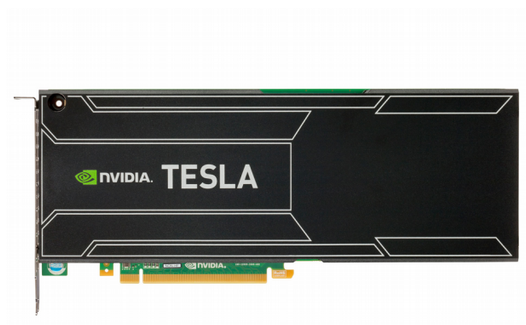
- HPC heterogeneus SoC?
  - could use better control of where to shift energy
  - kernel annotation?

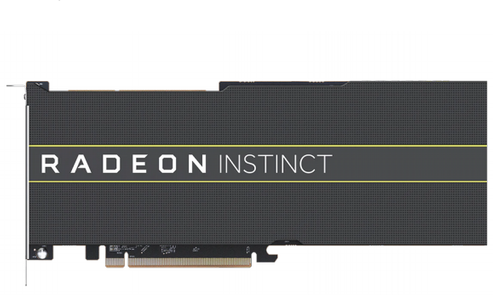
Power consumption measured using likwid.

Default CPU clocks, turbo boost on, performance governor.

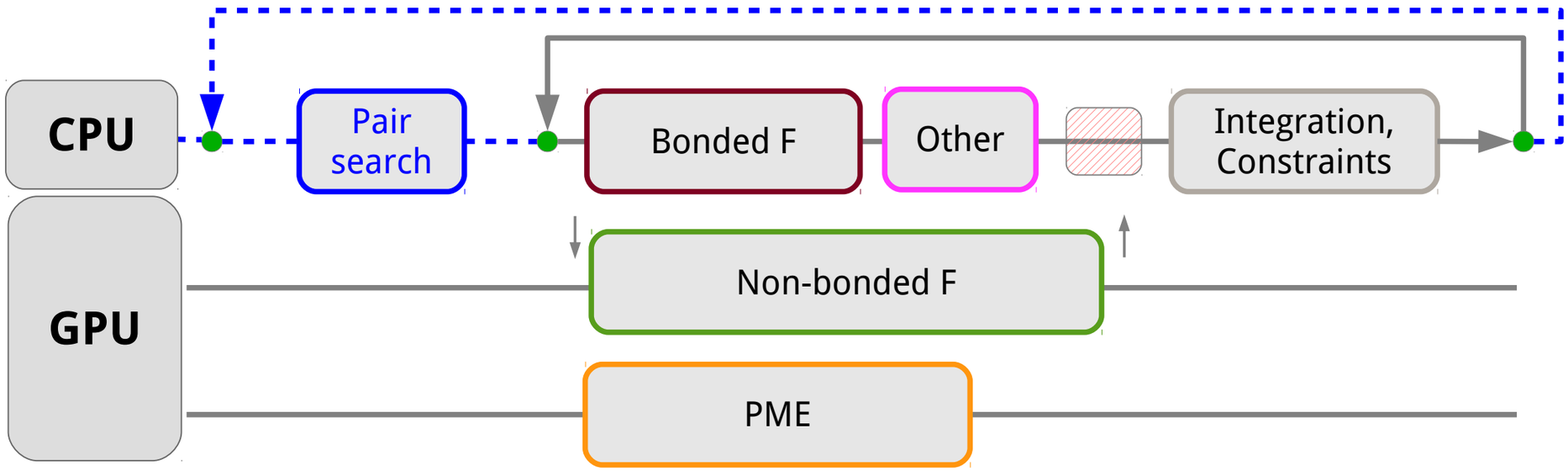
GPU offload: nonbonded & **PME**
GROMACS 2018 / 2019





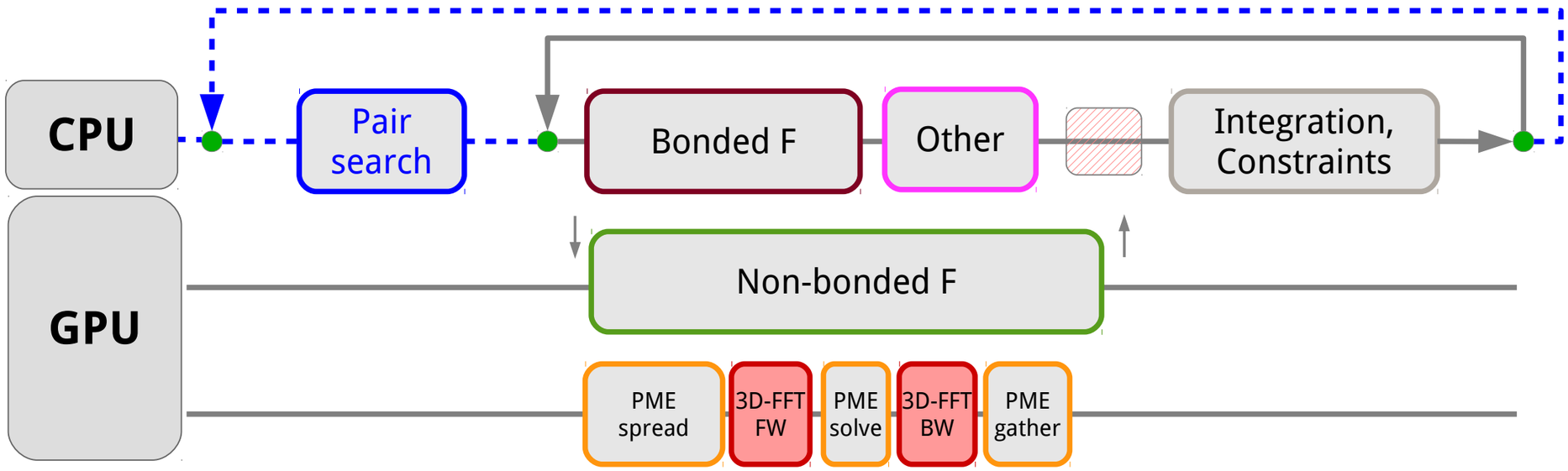


GPU offload: nonbonded & **PME**
GROMACS 2018 / 2019



- PME offload: initially CUDA-only in 2018
- OpenCL support in v2019
  - perf tuning needed

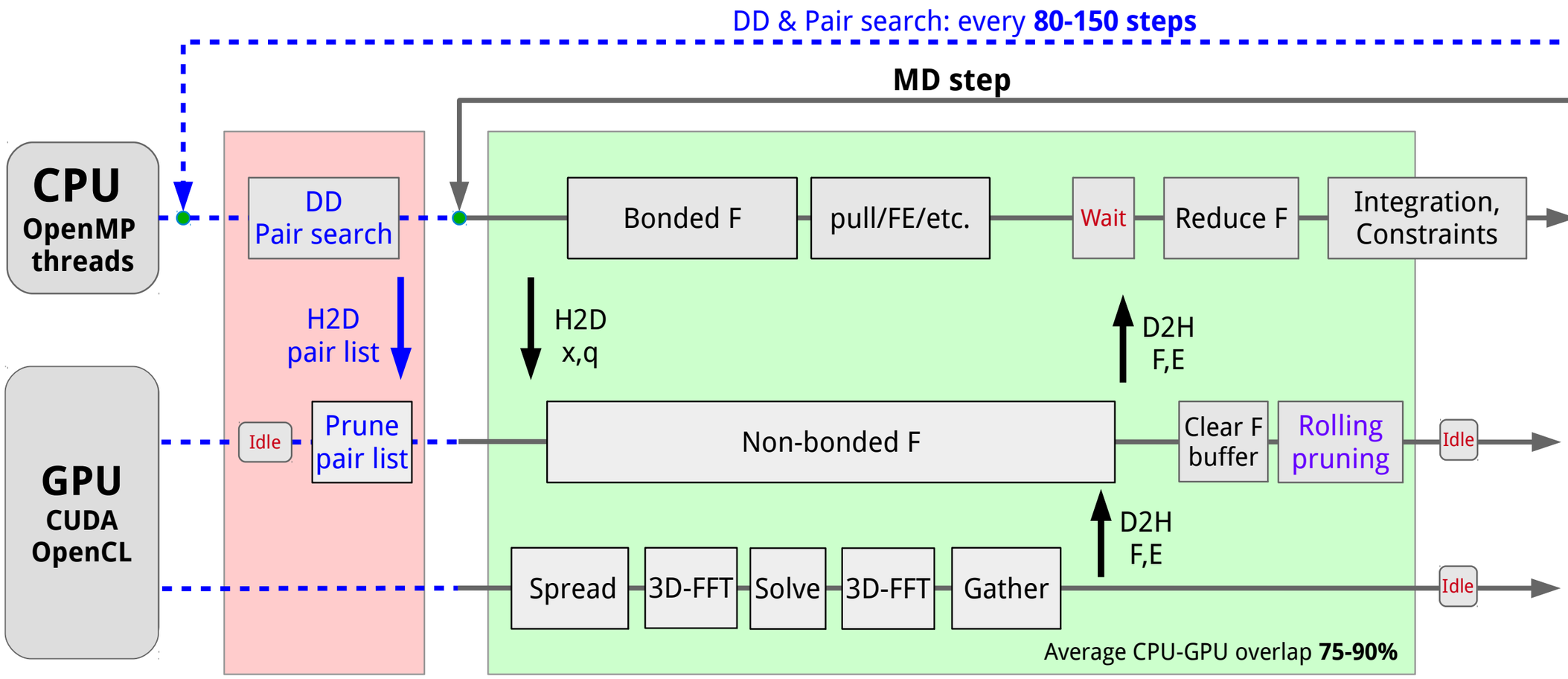
GPU offload: nonbonded & **PME** GROMACS 2018 / 2019



- PME offload: initially CUDA-only in 2018
- OpenCL support in v2019: AMD
  - NVIDIA and Intel disabled in the release
  - perf tuning needed

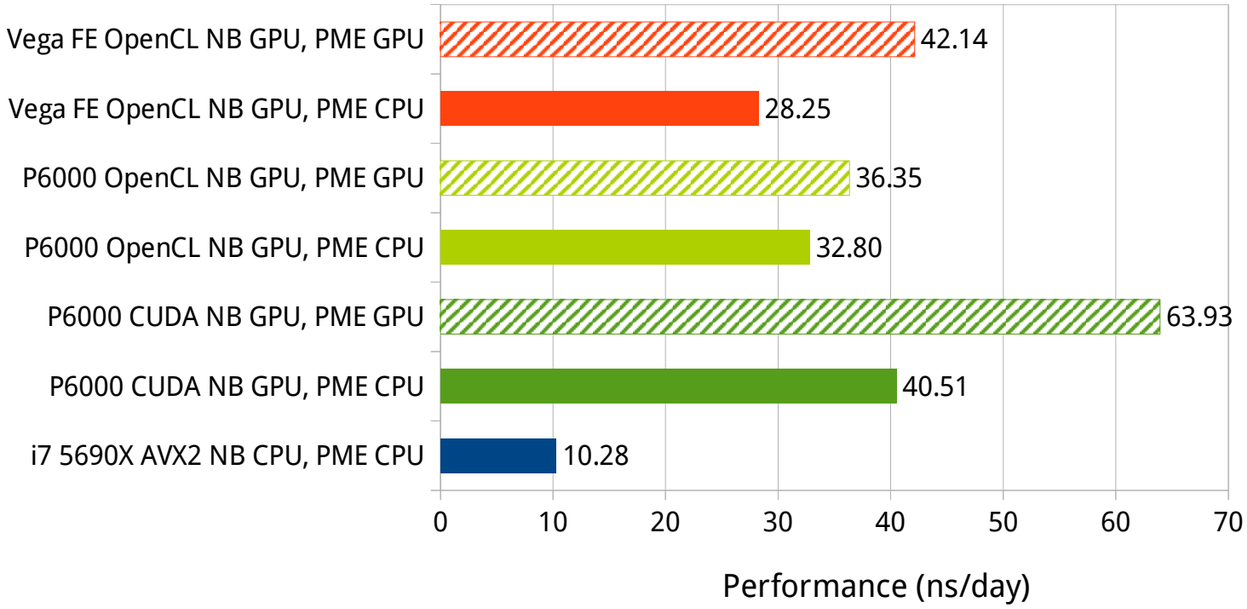
Need better OpenCL FFT library!
- vendors (rocFFT? OneAPI / MKL?)
- wish: a 3rd party portable “clFFTW”

Heterogeneous parallel scheme



PME offload performance

Input case: “ion_channel” 144k atoms



Hardware:

CPU:

i7-5960X (8C@ 3.5GHz)

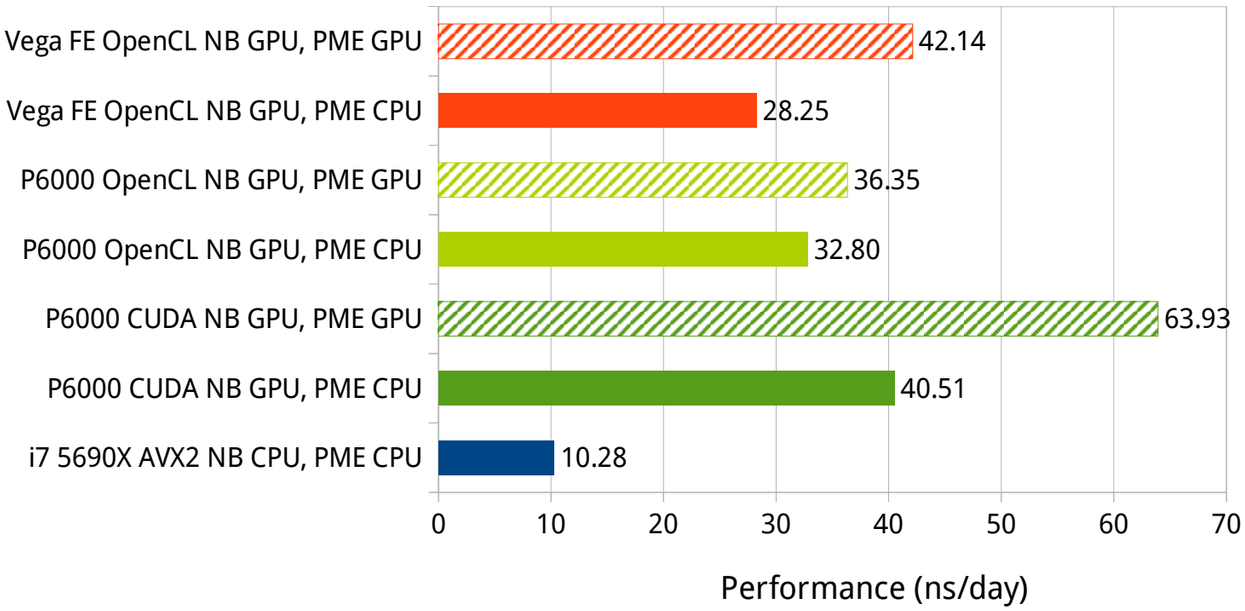
GPU:

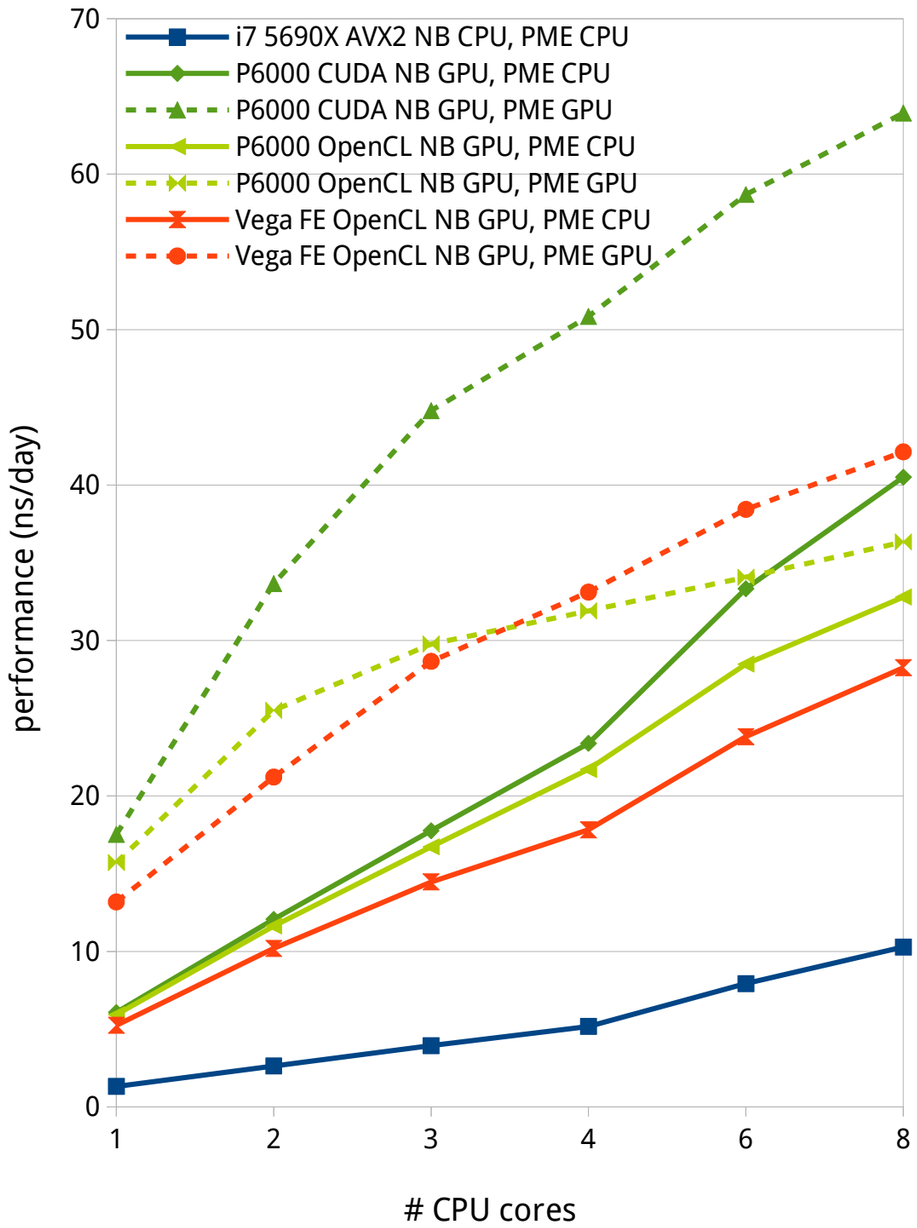
NVIDIA Quadro P6000

AMD Vega Frontier

PME offload performance

Input case: “ion_channel” 144k atoms





Hardware:

CPU:

i7-5960X (8C@ 3.5GHz)

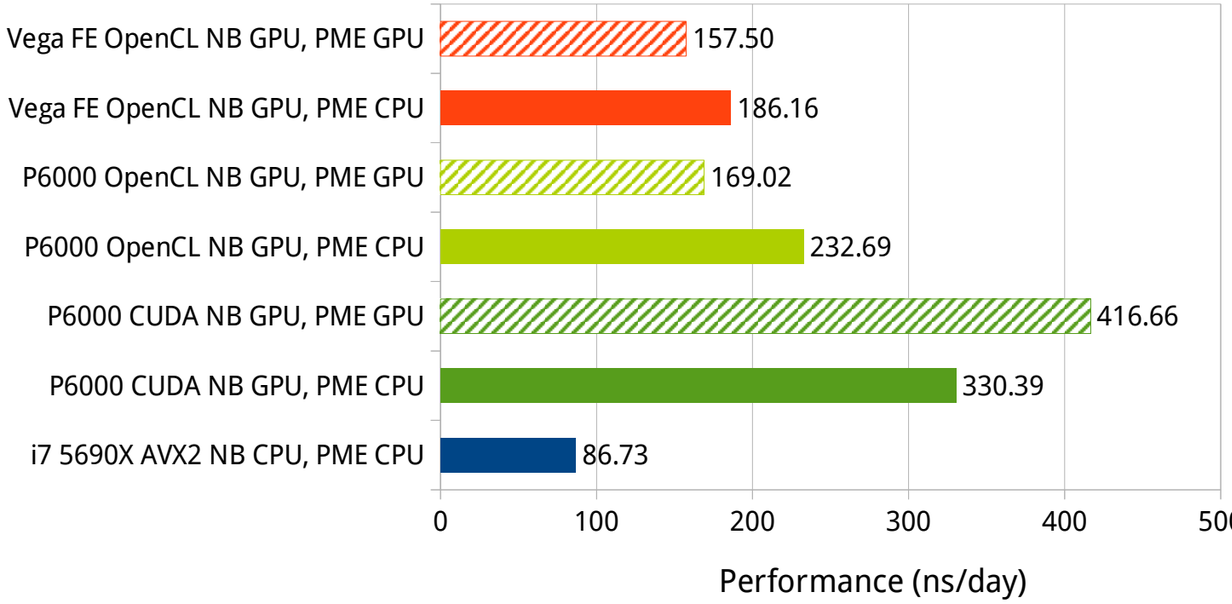
GPU:

NVIDIA Quadro P6000

AMD Vega Frontier

PME offload performance: the ugly

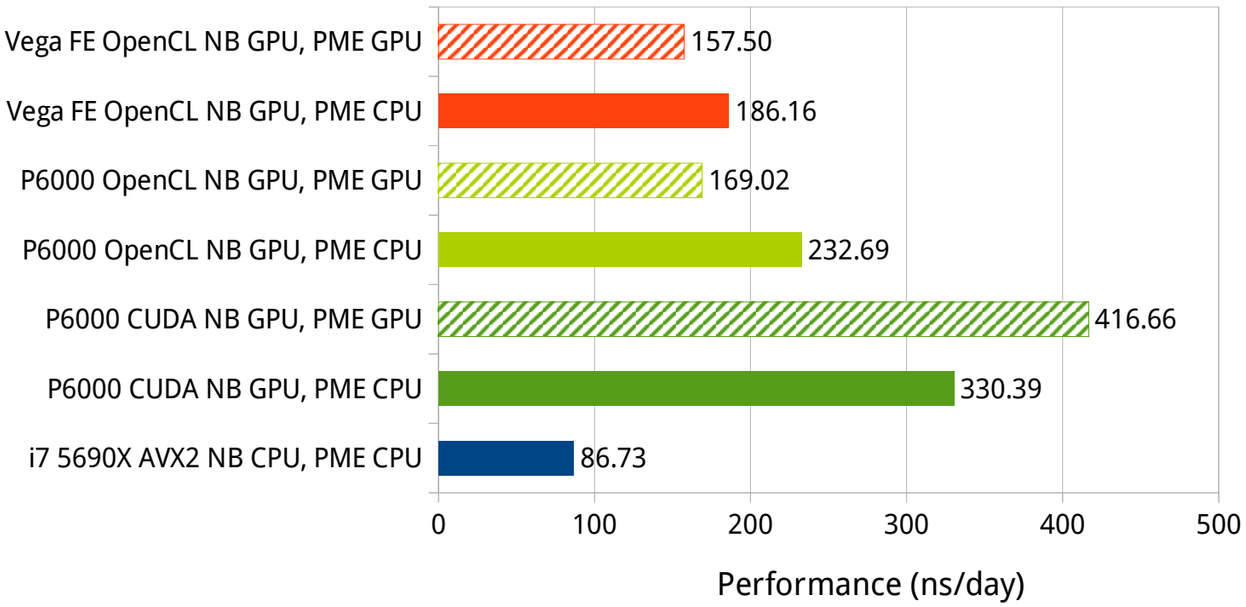
Input case: “rnase_cubic” 24k atoms

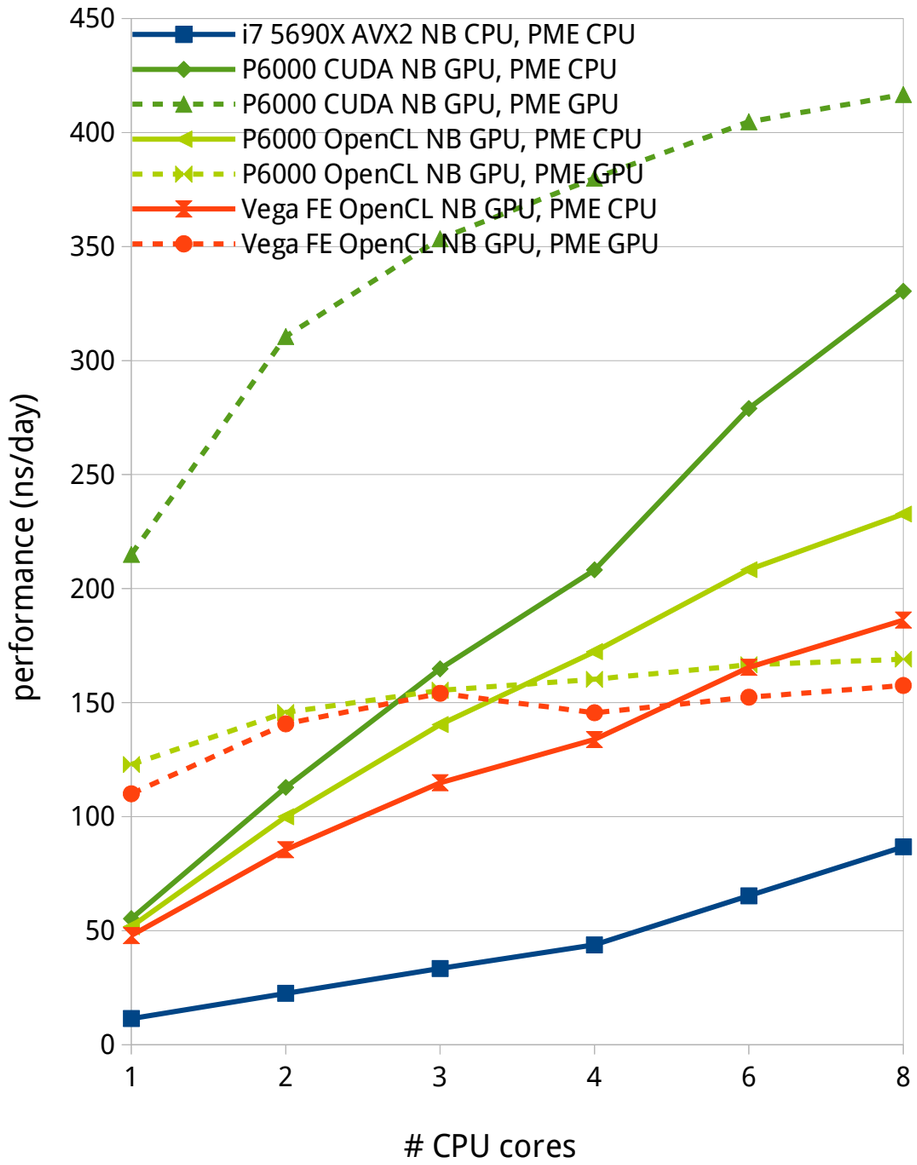


- Smaller test case highlights runtime overheads
  - API launch overheads
  - async calls blocking
  - CPU noise

PME offload performance: the ugly

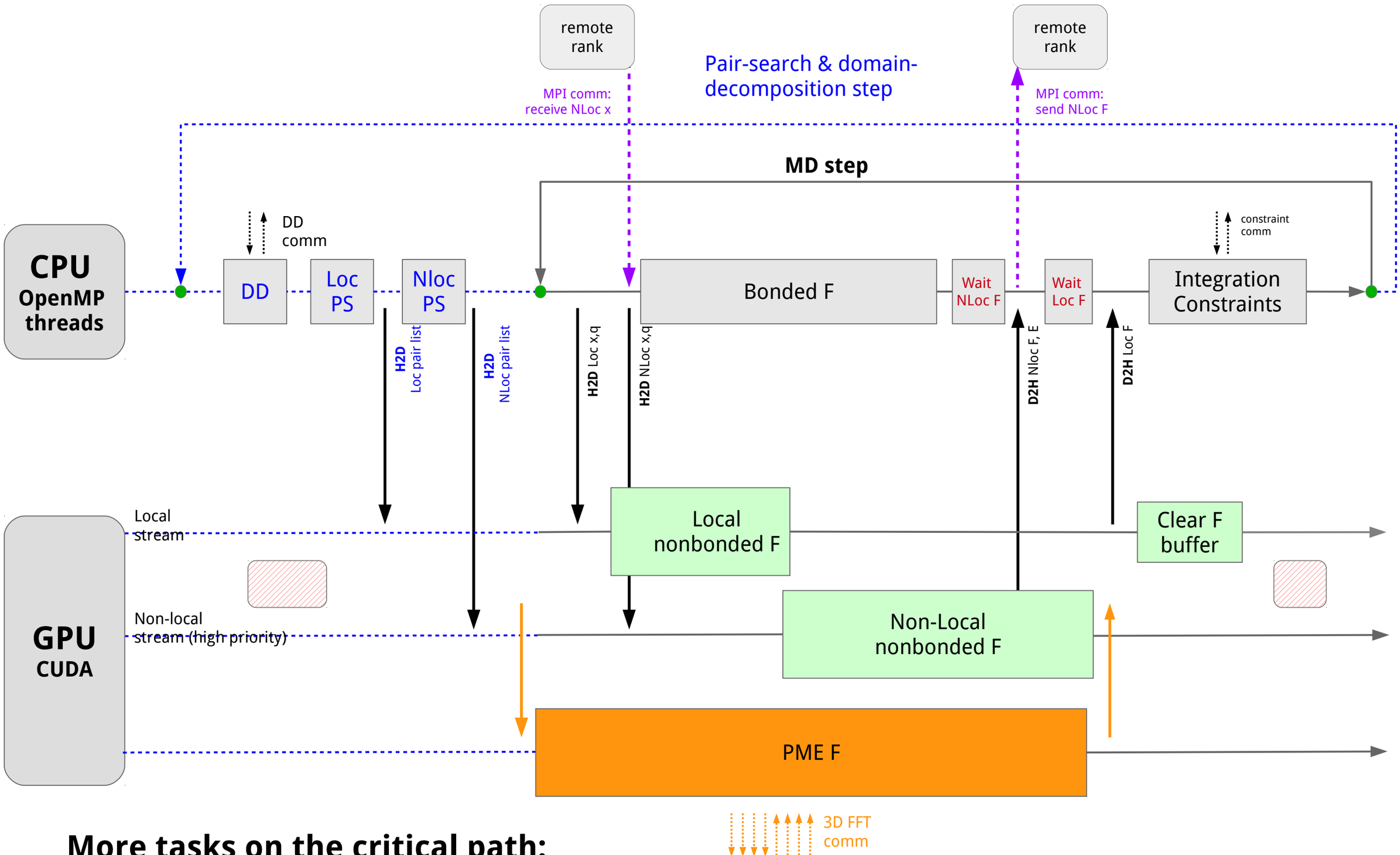
Input case: “rnase_cubic” 24k atoms





- Smaller test case highlights runtime overheads
  - API launch overheads
  - async calls blocking
  - CPU noise

PME offload with DD: more trouble on the critical path



More tasks on the critical path:
forward progress on the critical path more challenging

Critical path optimization challenges

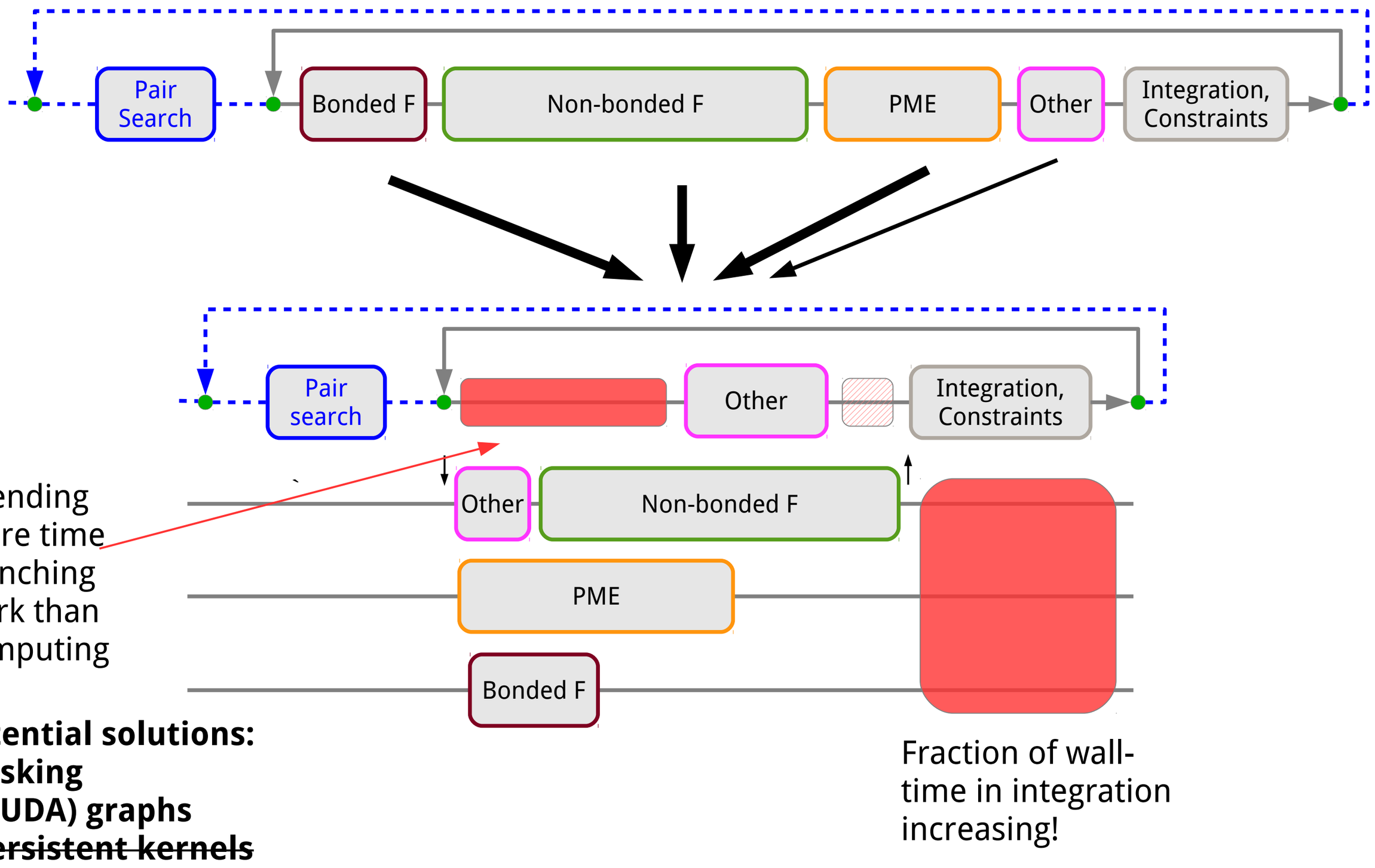
- Forward progress is not ensured by priority
  - eager execution fills the GPU
  - low-prio kernel(s) compete with high low-prio kernels
    - offloading small tasks for critical path / locality not performance may not be worth it?

Competing high prio work and “backfill” low prio kernel

Maxwell

Kepler

GPU offload: future



Potential solutions:
- tasking
- (CUDA) graphs
- ~~persistent kernels~~

Summary

- OpenCL in GROMACS:
  - we made a long-term investment into portability
  - hoping to see the returns!

- OpenCL software stack:
  - can we have 2.x? subgroup primitives without assembly
  - compilers, runtime: Intel leading, AMD needs to mature, NVIDIA ?
  - libraries: FFT lacking (need small 3D-FFTs, SP R2C/C2R)
  - perf tools

- Programmability / performance concerns
  - task launch overheads (batched / “graph” launch?)
  - ensuring async progress: CPU-GPU and multipler kernels on GPU
  - GPU partitioning?
  - runtime noise

Acknowledgments

GROMACS

Berk Hess

Roland Schulz

Aleksei Yupinov

Erik Lindahl

Mark Abraham

Partners

Funding

Q&A