Tackling Performance Bottlenecks in the Diversifying CUDA HPC Ecosystem: a Molecular Dynamics Perspective

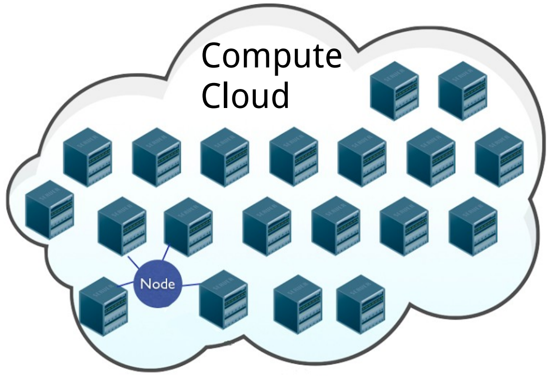
Szilárd Páll

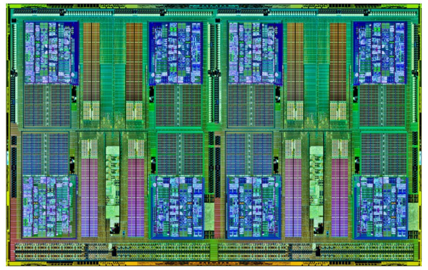
KTH Royal Institute of Technology

GTC 2015

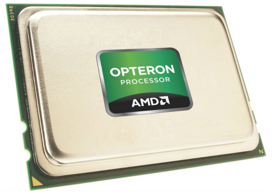
Diversifying hardware & complex parallelism

- Increasingly:
  - parallel on multiple levels
  - heterogenous
  - power constrained

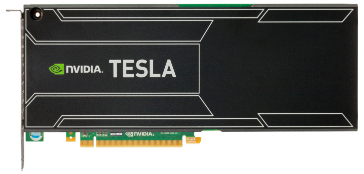




Widening SIMD/SIMT units

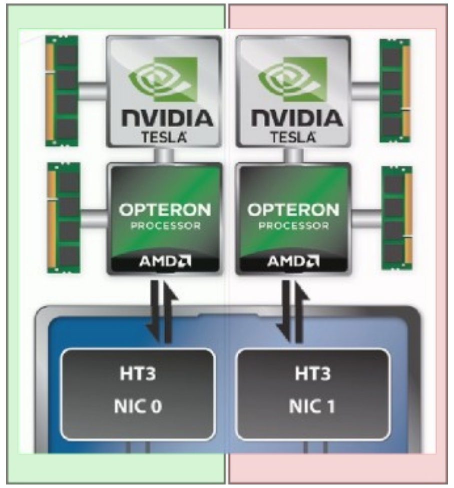


+



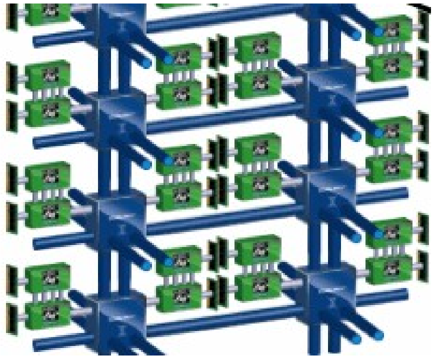
Increasing CPU/GPU core count
NUMA on die

x2-8



“Skinny” workstations to fat compute nodes
NUMA, NUAA, ...

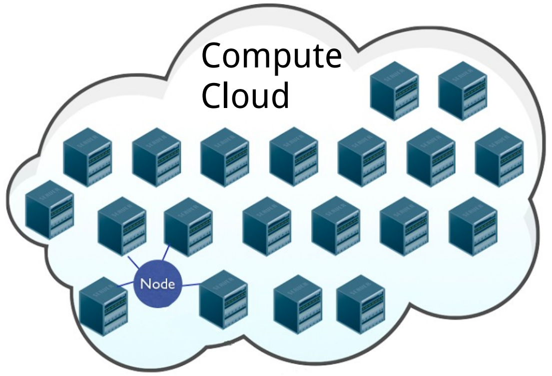
$x10^{2-5}$

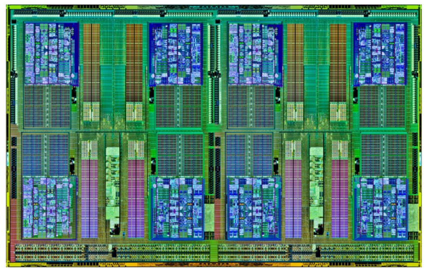


Mini-clusters, petaflop machines, & On-demand computing

Diversifying hardware & complex parallelism

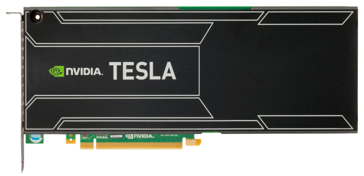
- Need to address each level
- Choice of parallelization important
- How much of the burden is placed on the user?





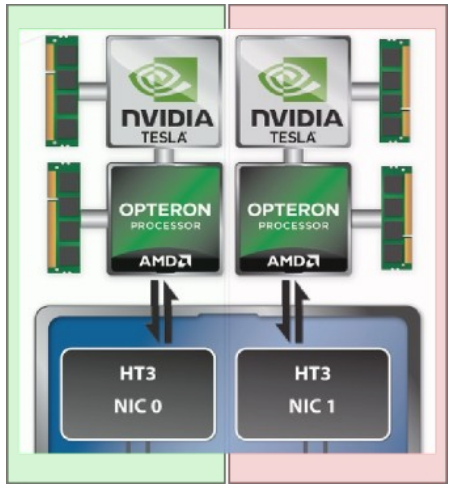
Widening SIMD/SIMT units

+



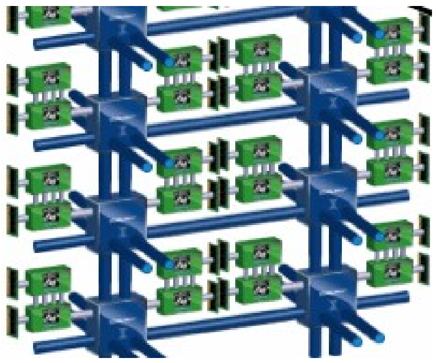
Increasing CPU/GPU
core count
NUMA on die

x2-8



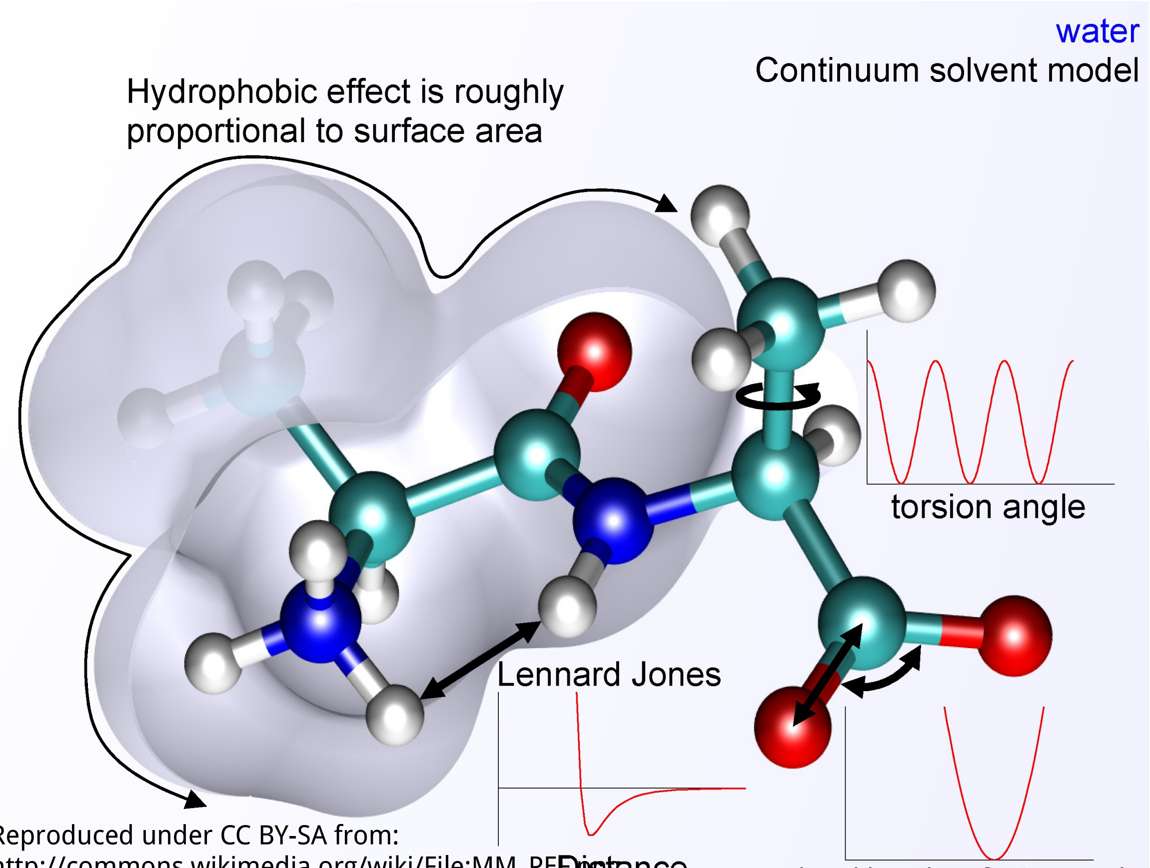
“Skinny” workstations
to fat compute nodes
NUMA, NUAA, ...

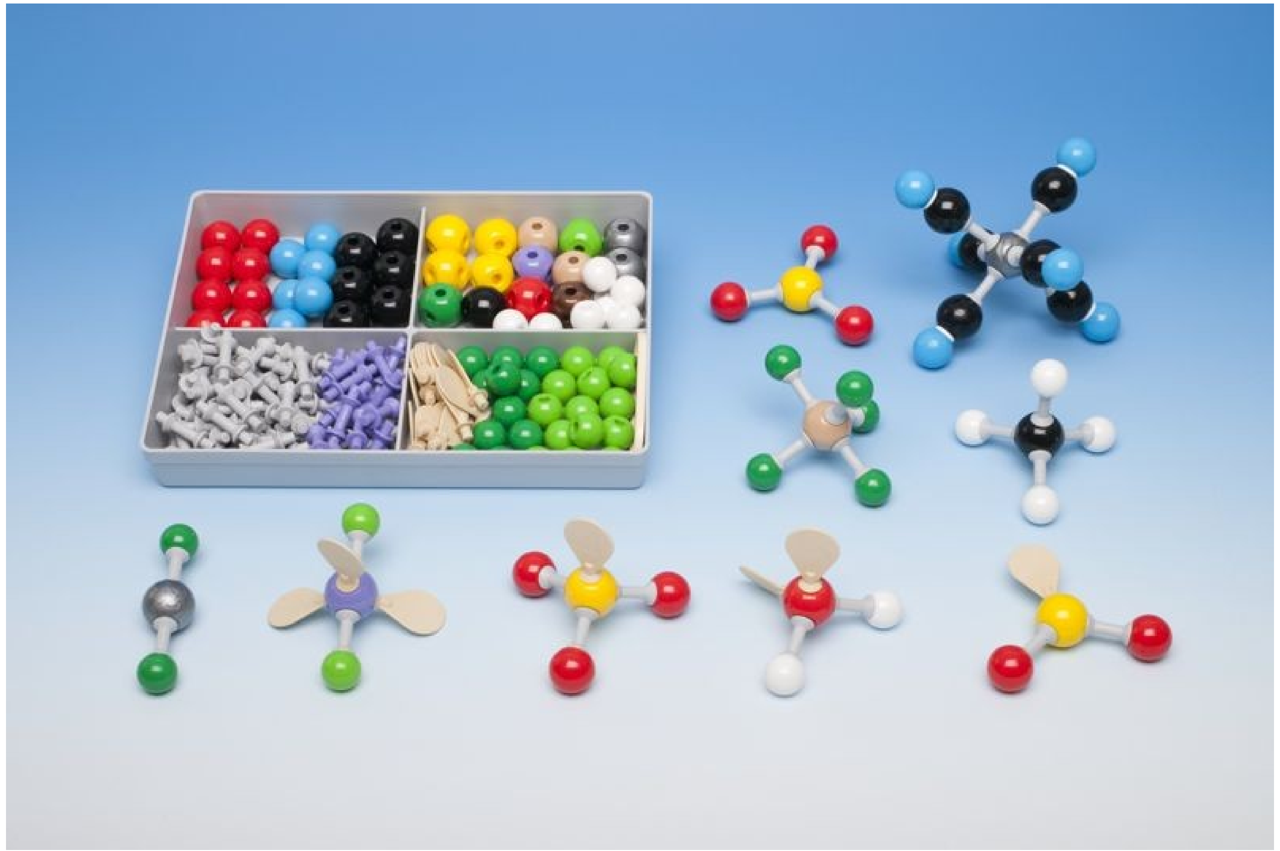
x10^{2-5}



Mini-clusters,
petaflop machines, &
On-demand computing

Molecular dynamics: modelling physics





Molecular dynamics: basics

- Given:
  - N particles
  - masses
  - potential V

Newton's equations

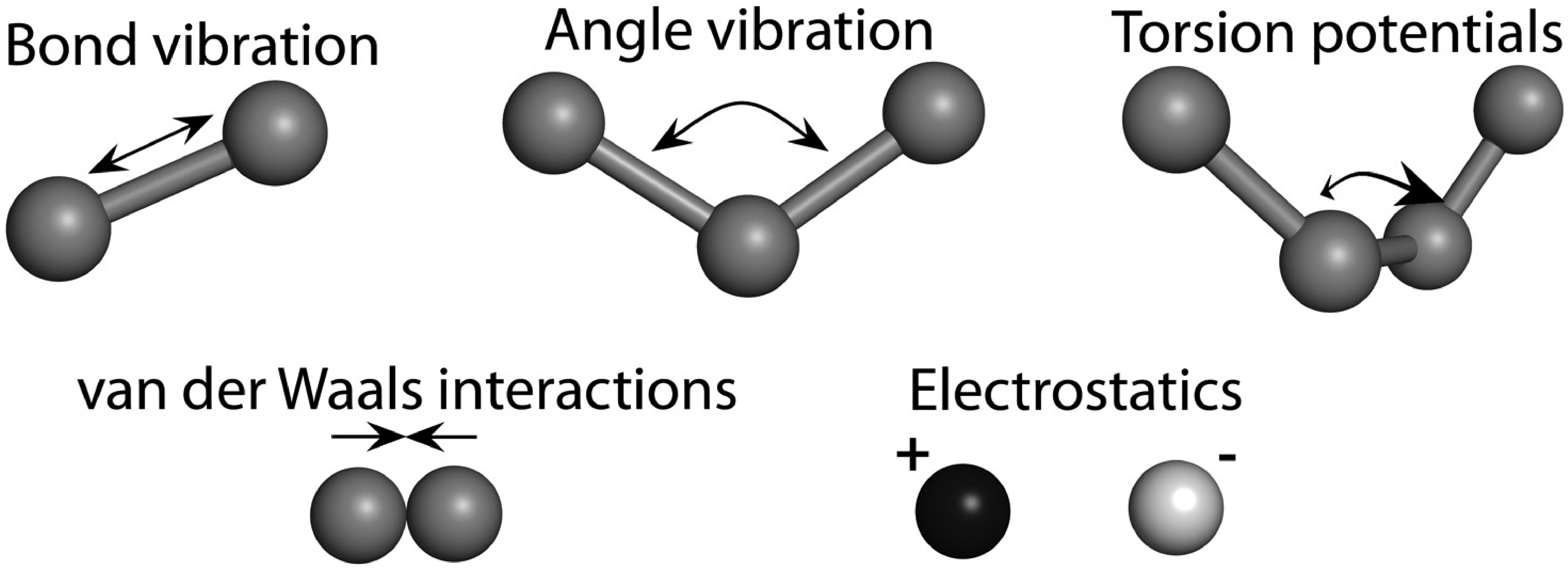
$$m_i \frac{d^2 \mathbf{x}_i}{d t^2} = -\nabla_i V(\mathbf{x}) \quad , \qquad i = 1, \ldots, N$$

- Integrate (leap frog)
  - acceleration → velocities
  - velocities → coordinates

$$v_i(t + \frac{\Delta t}{2}) = v_i(t - \frac{\Delta t}{2}) + a_i(t)\Delta t$$

$$x_i(t + \Delta t) = x(t) + v_i(t + \frac{\Delta t}{2})$$

Molecular dynamics: interactions



Molecular dynamics: forces

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

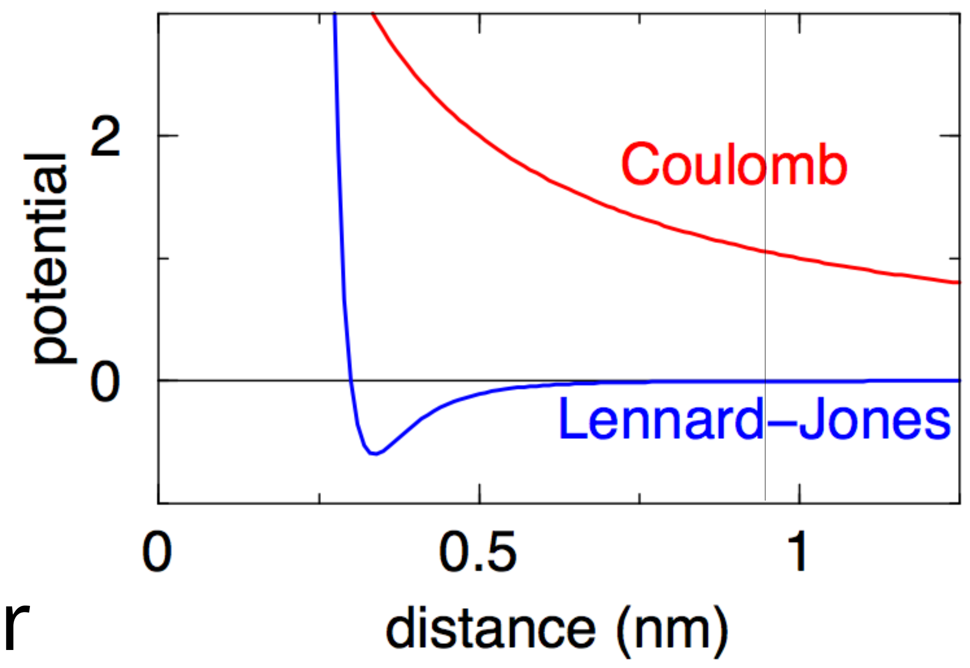
Over all atom-pairs!

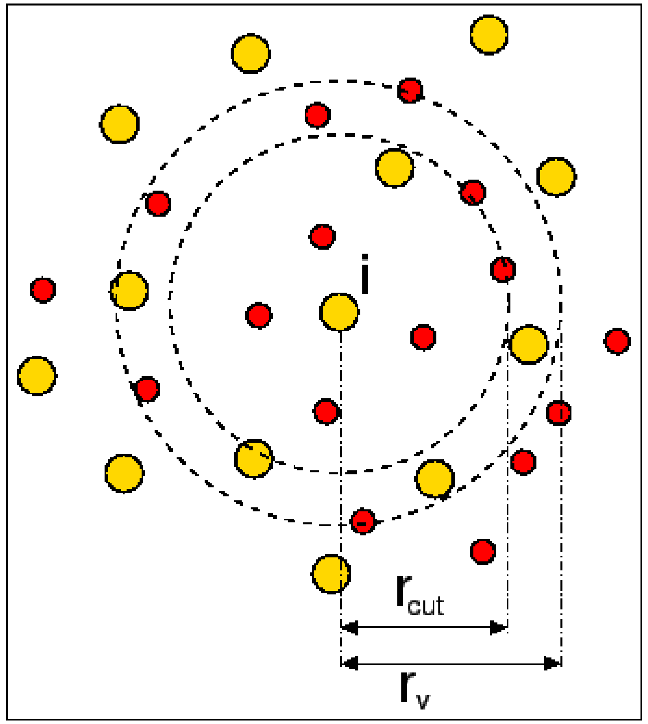
- Bonded forces: loop over all interactions
  - few but localized → imbalance challenge (threading & domain decomposition)
- Non-bonded: a double loop?

  → too expensive: limit the interaction range (cut-off)

Pair interactions: cut-off

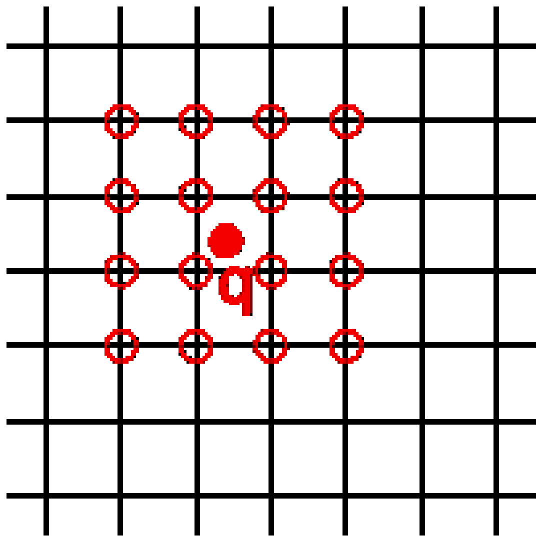
- LJ decays fast: 1/r^6
  - can use a cut-off





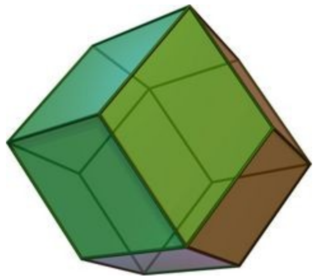
- Coulomb decays slowly: 1/r
  - cut-off is not good enough

  → treat long-range part separately: Particle Mesh Ewald

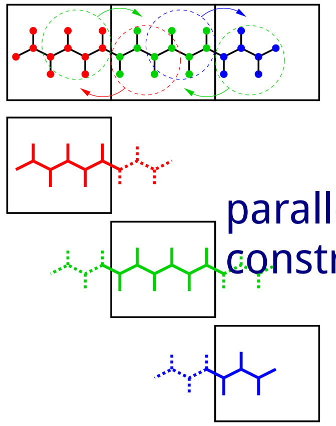


GROMACS: fast, flexible, free

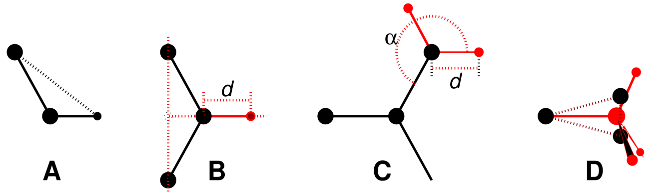
- **Developers**: Stockholm & Uppsala, SE and many more worldwide
- **Open source**: LGPLv2
- **Open development:** `https://gerrit.gromacs.org`
- **Large user base**:
  - 10k's academic & industry
  - 100k's through F@H
- **Supports all major force-fields**



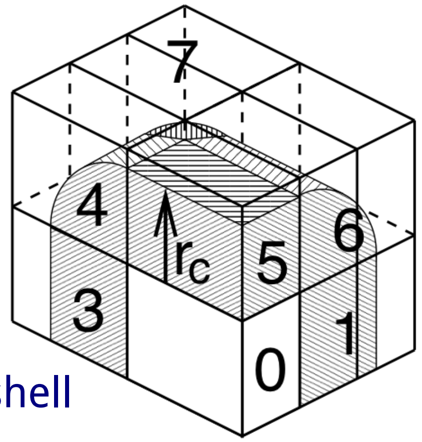
arbitrary units cells



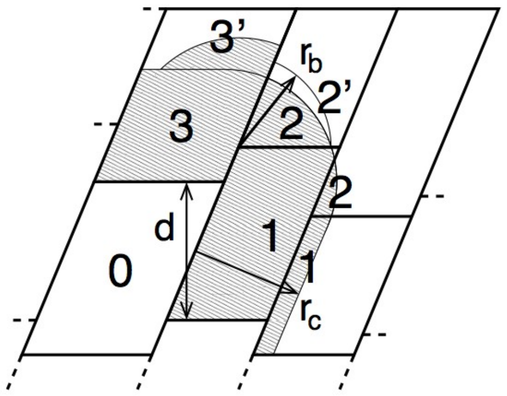
parallel constraints



virtual interaction sites



eighth shell domain decomposition



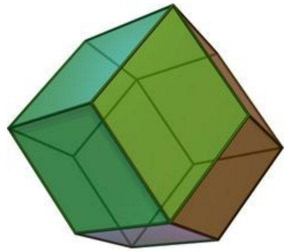
Triclinic unit cell with load balancing and staggered cell boundaries

GROMACS: fast, flexible, free

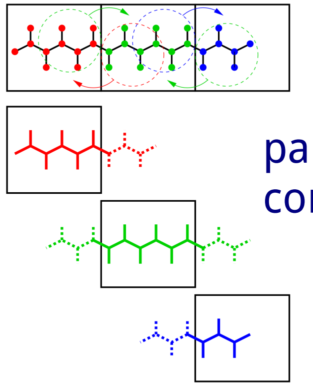
- **Code: portability of great importance**
  - C++98 (subset)
  - CMake
- **Pretty large:**
  - LOC: ~2 mil. ½ of which is SIMD!

- Bottom-up performance tuning

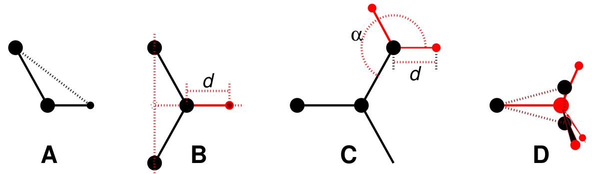
  → **absolute performance is what matters (to users)**



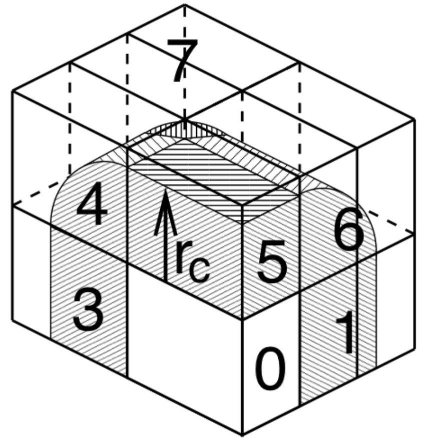
arbitrary units cells



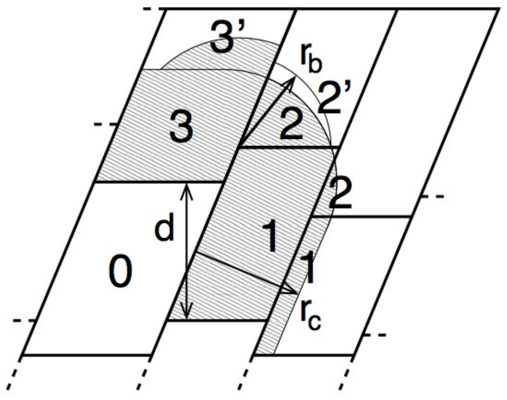
parallel constraints



virtual interaction sites



eighth domain decomposition

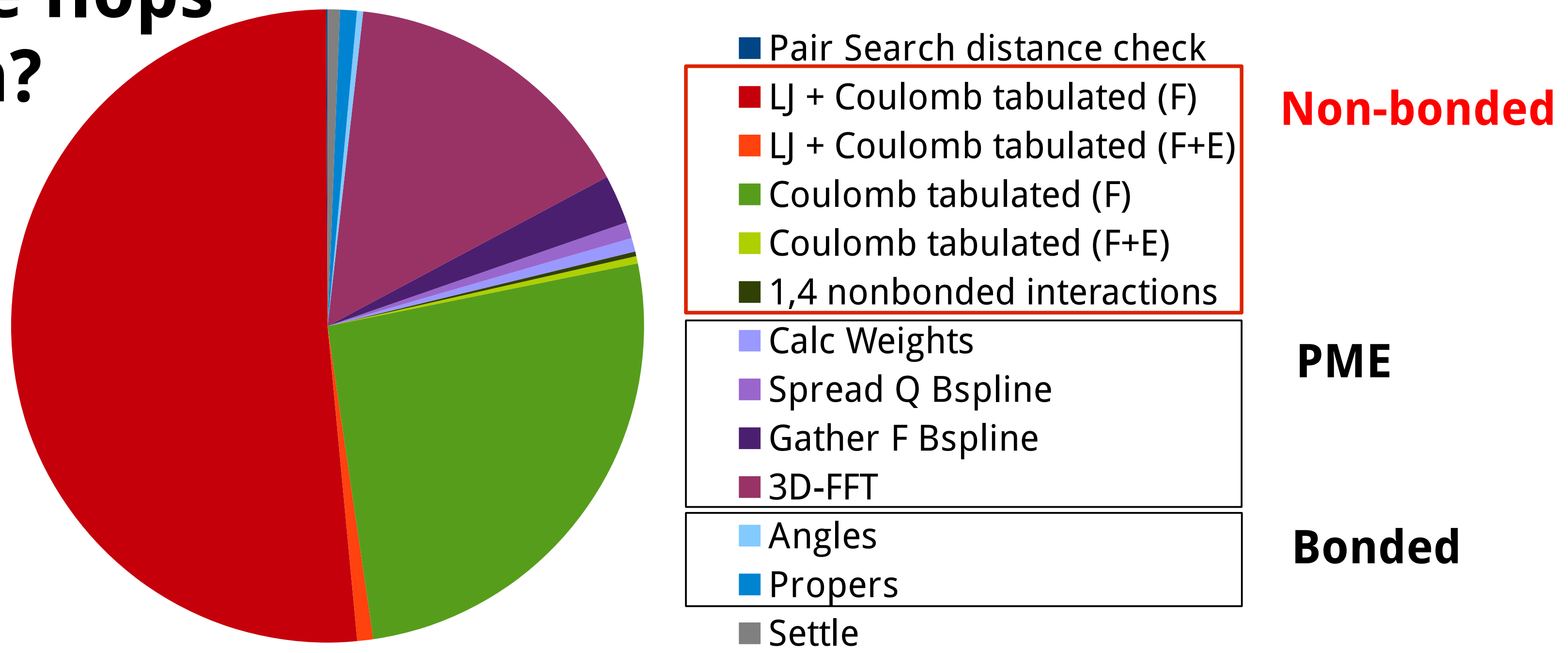


Triclinic unit cell with load balancing and staggered cell boundaries

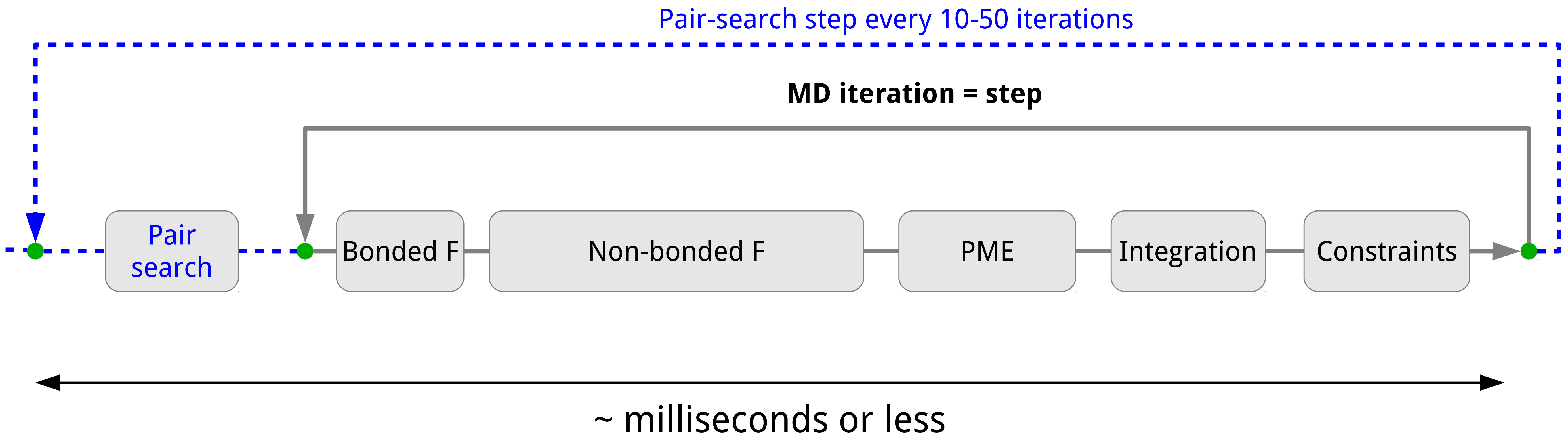
Costs in MD

- Every step: 10^6 -10^8 Flops
- Every simulation: 10^6 -10^8 steps

What are flops spent on?



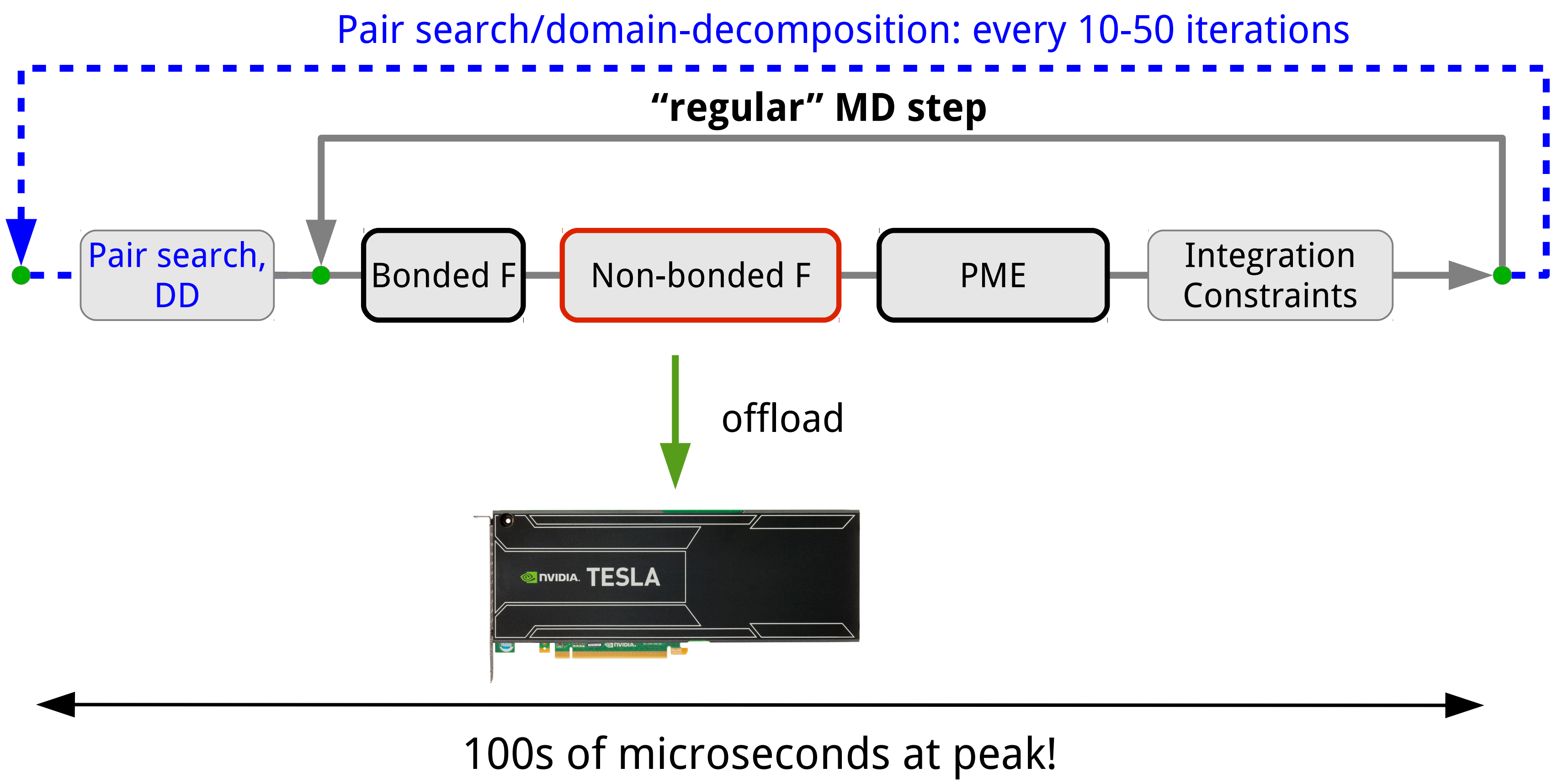
Molecular dynamics step



Goal: do it as fast as possible!

Herogenous accelerated GROMACS

- 2nd gen GPU acceleration:
  since GROMACS v4.6
- **Advantages:**
  - 2-4x speedup
  - offload → multi-GPU “for free”
  - wide feature support
- **Challenges:**
  - added latencies, overheads
  - load balancing

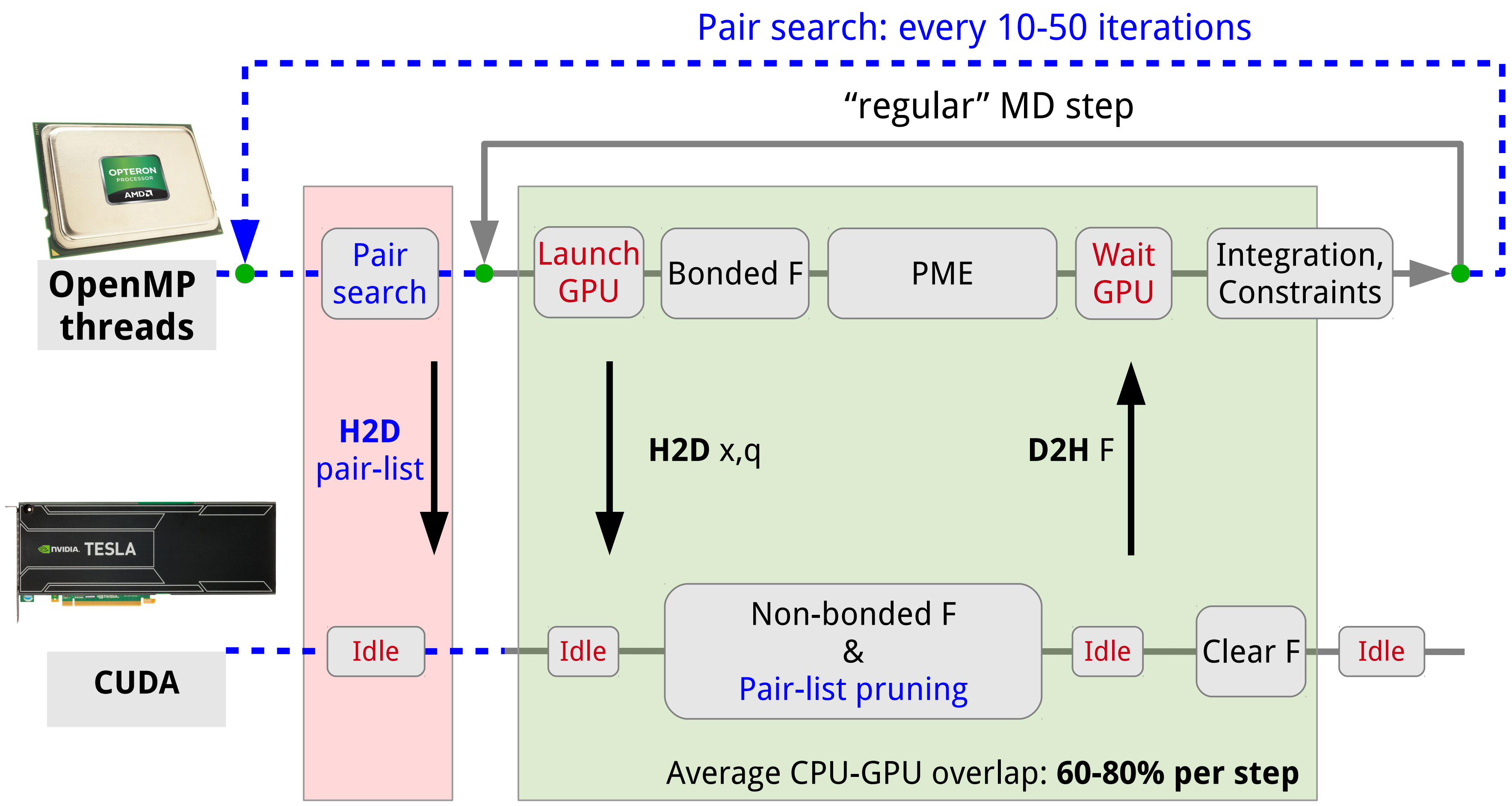


Herogenous accelerated MD

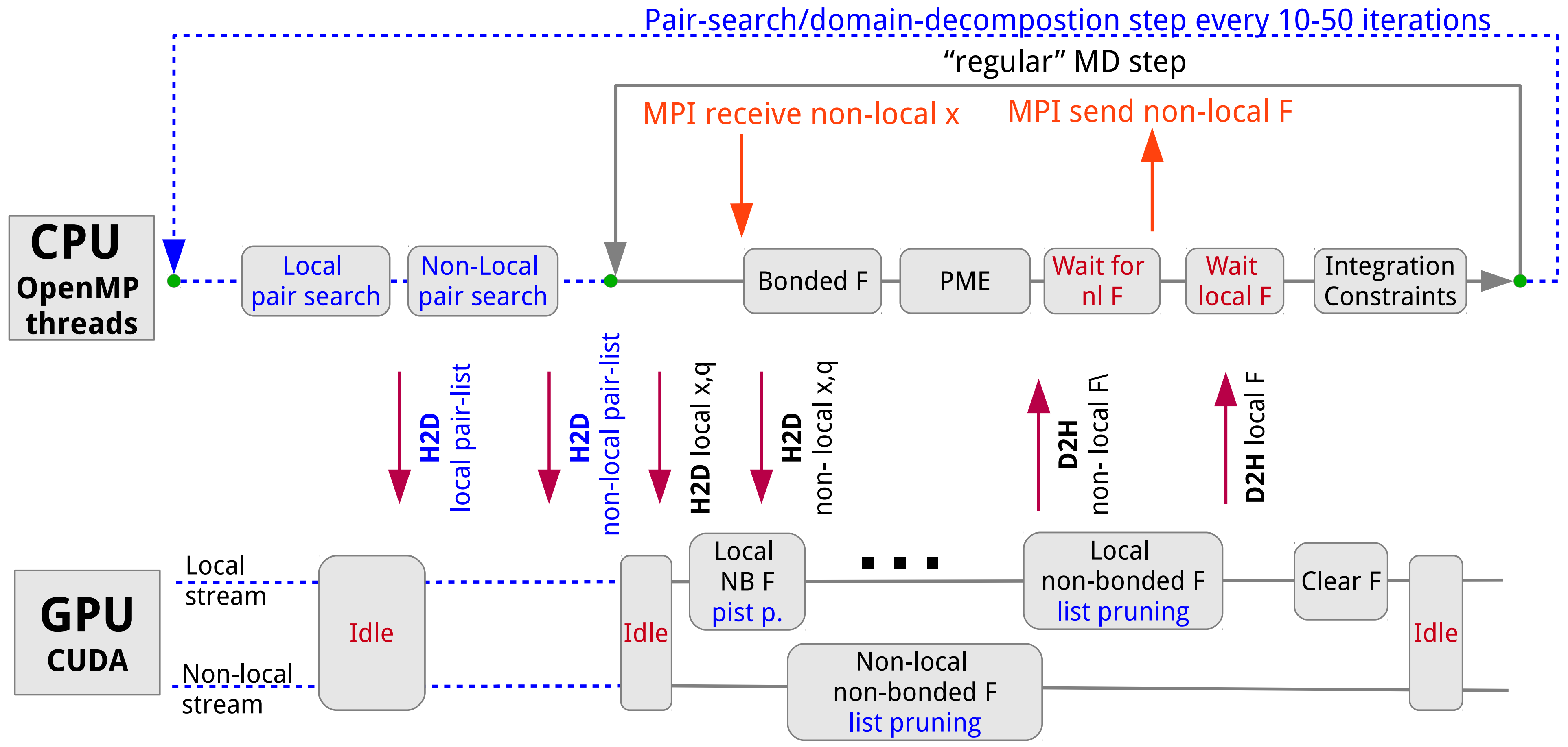
- 2nd gen GPU acceleration:

  since GROMACS v4.6

- **Advatages:**
  - 3-4x speedup
  - offload → multi-GPU “for free”
  - wide feature support
- **Challenges:**
  - added latency, re-casting work for GPUs
  - load balancing: intra-GPU, intra-node,...



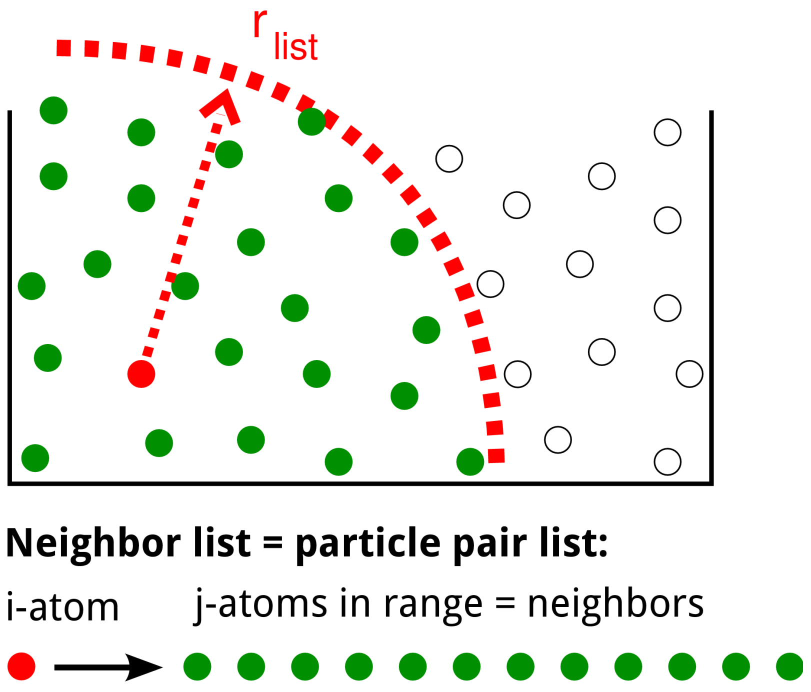
Parallel heterogeneous MD

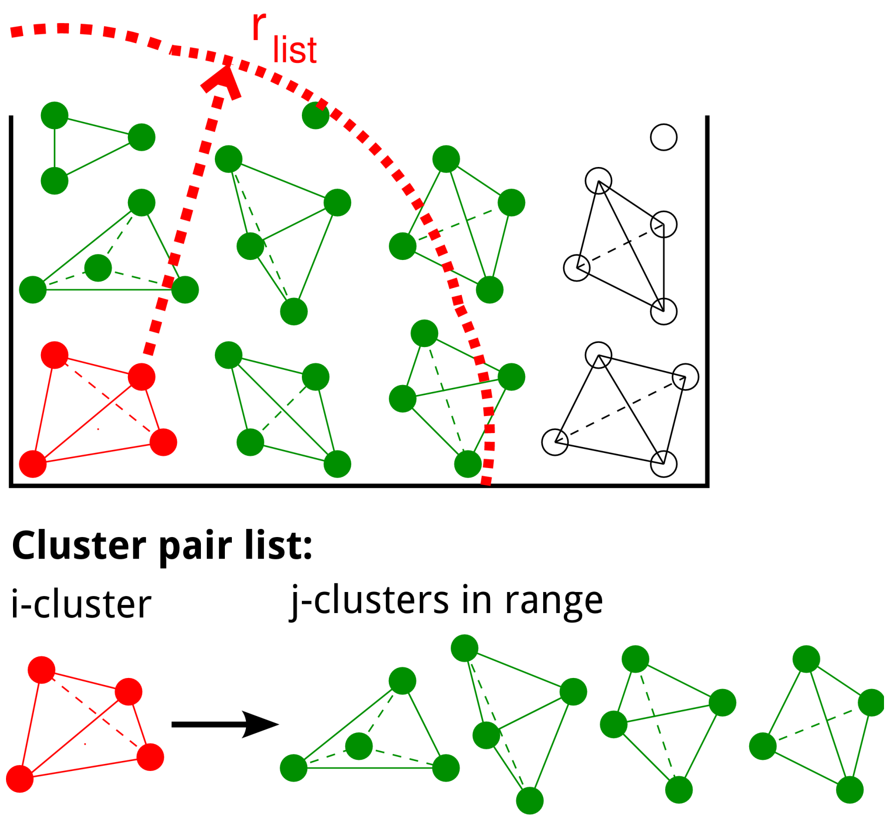


Intra-node:
The accelerator

SIMD/SIMT targeted algorithms

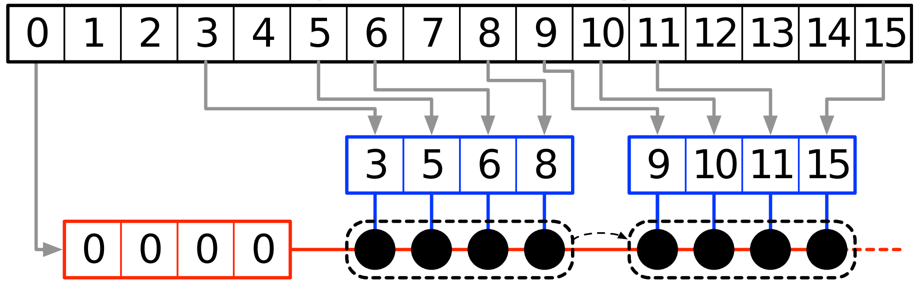
- Cluster pair interaction algorithm
  - lends itself well to efficient **fine-grained parallelization**
  - **adaptable** to the characteristics of the architecture





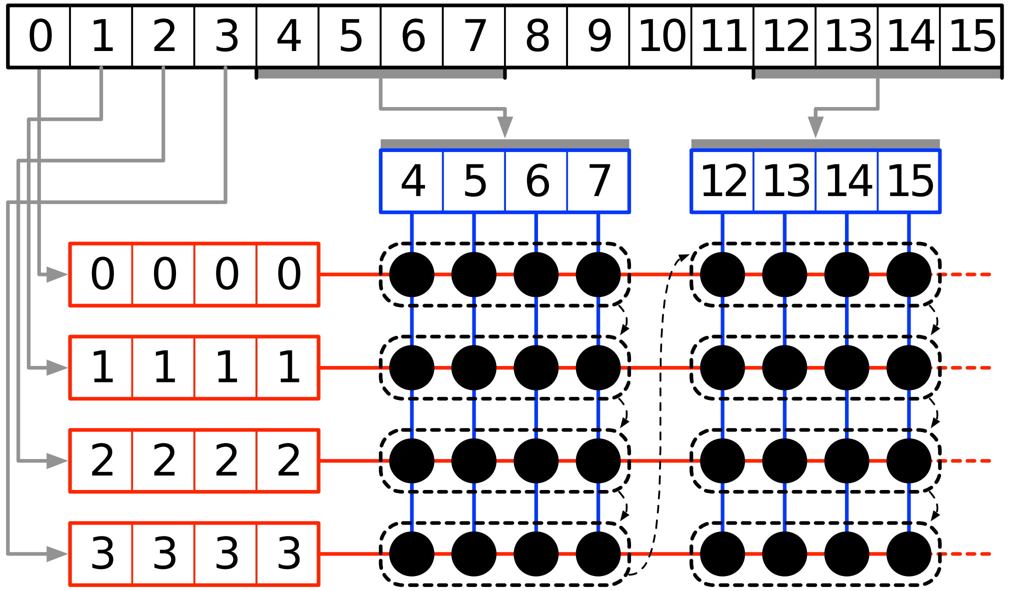
Particle cluster algorithm: SIMD implementation

Classical 1x1 neighborlist on 4-way SIMD



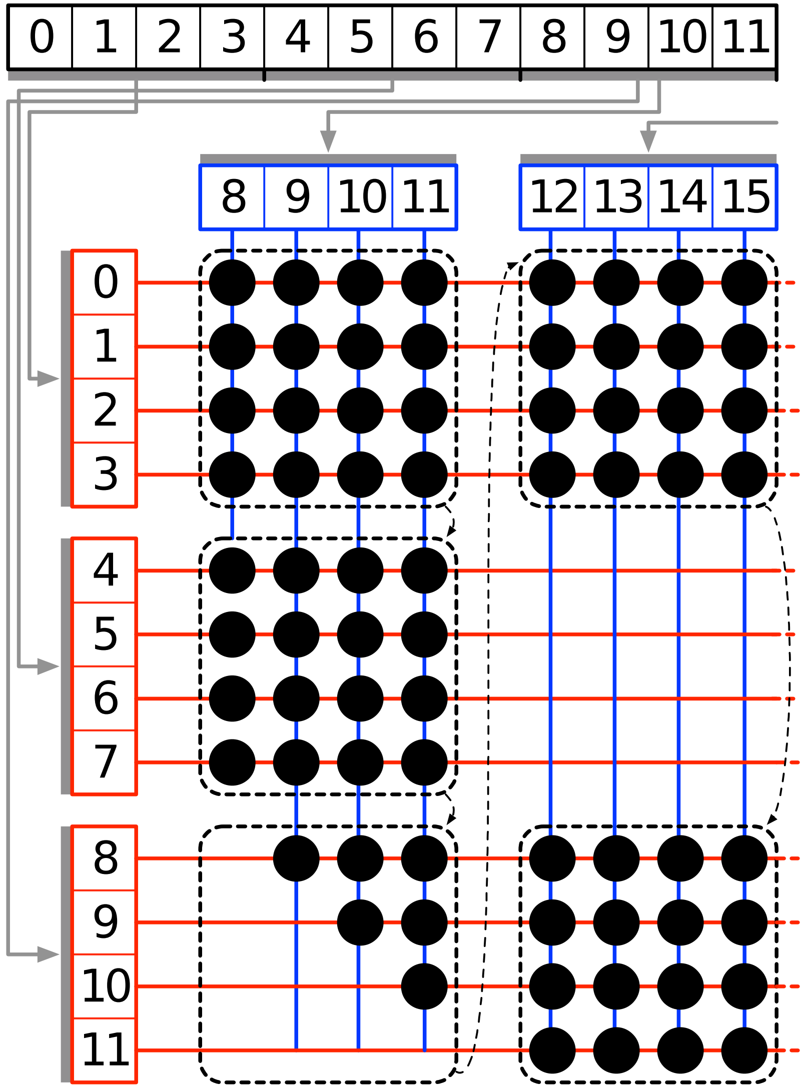
Traditional algorithm: cache pressure, ill data reuse, in-register shuffle bound

4x4 setup on 4-way SIMD



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

4x4 setup on SIMT-

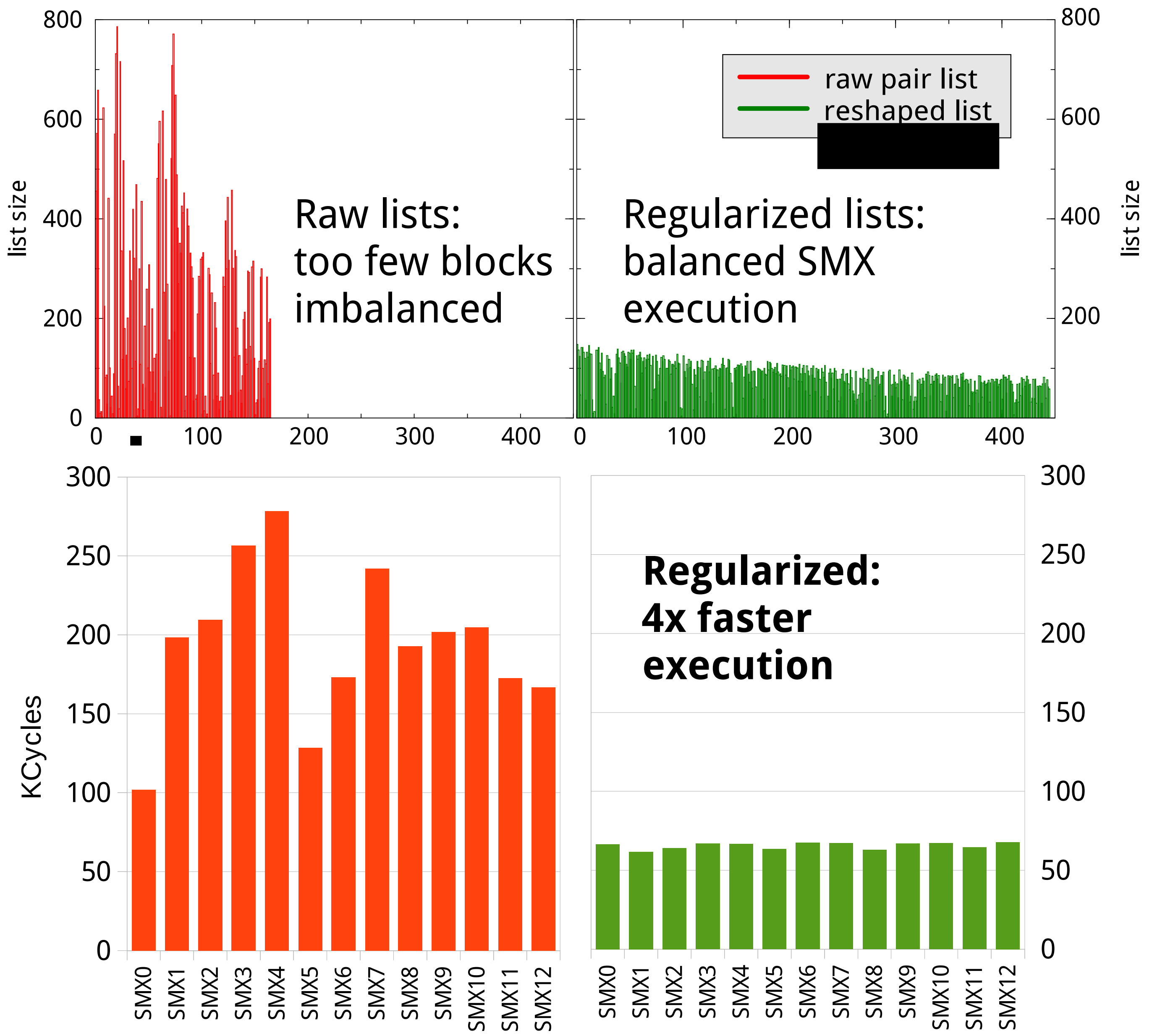


Cluster algorithm for fine-grained hardware threading: SIMT, etc.

Cluster size and grouping are the "knobs" to adjust for a specific arch:

- data reuse
- arithmetic intensity
- cache-efficiency

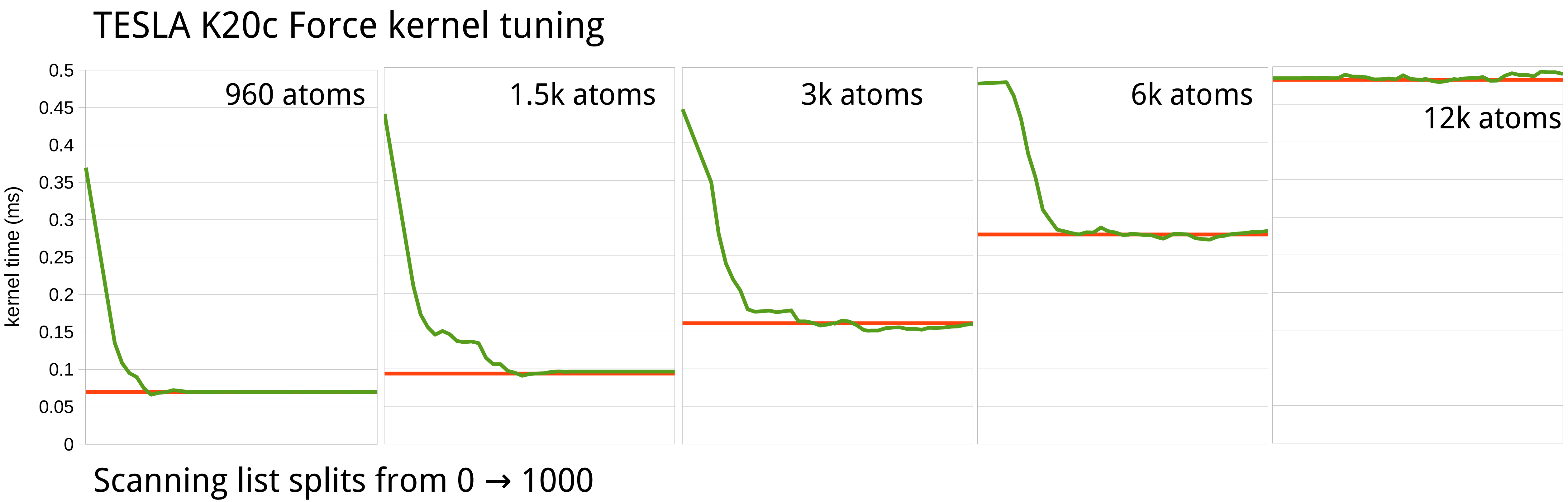
Tuning kernels: feeding the GPU



Workload per SMX: Tesla K20c, 1500 atoms

- **Avoid load imbalance:**
  - create enough idependent work units = blocks per SM[X|M]
  - sort them
- **Workload regularization** improves by up to:
  - 2-3x on “narrower”
  - 3-5x on “wider” GPUs
- (Re-)tuning is needed for new architectures
- Tradeoffs:
  - lowers j-particle data reuse
  - atomic clashes

Tuning kernels: ready for automation

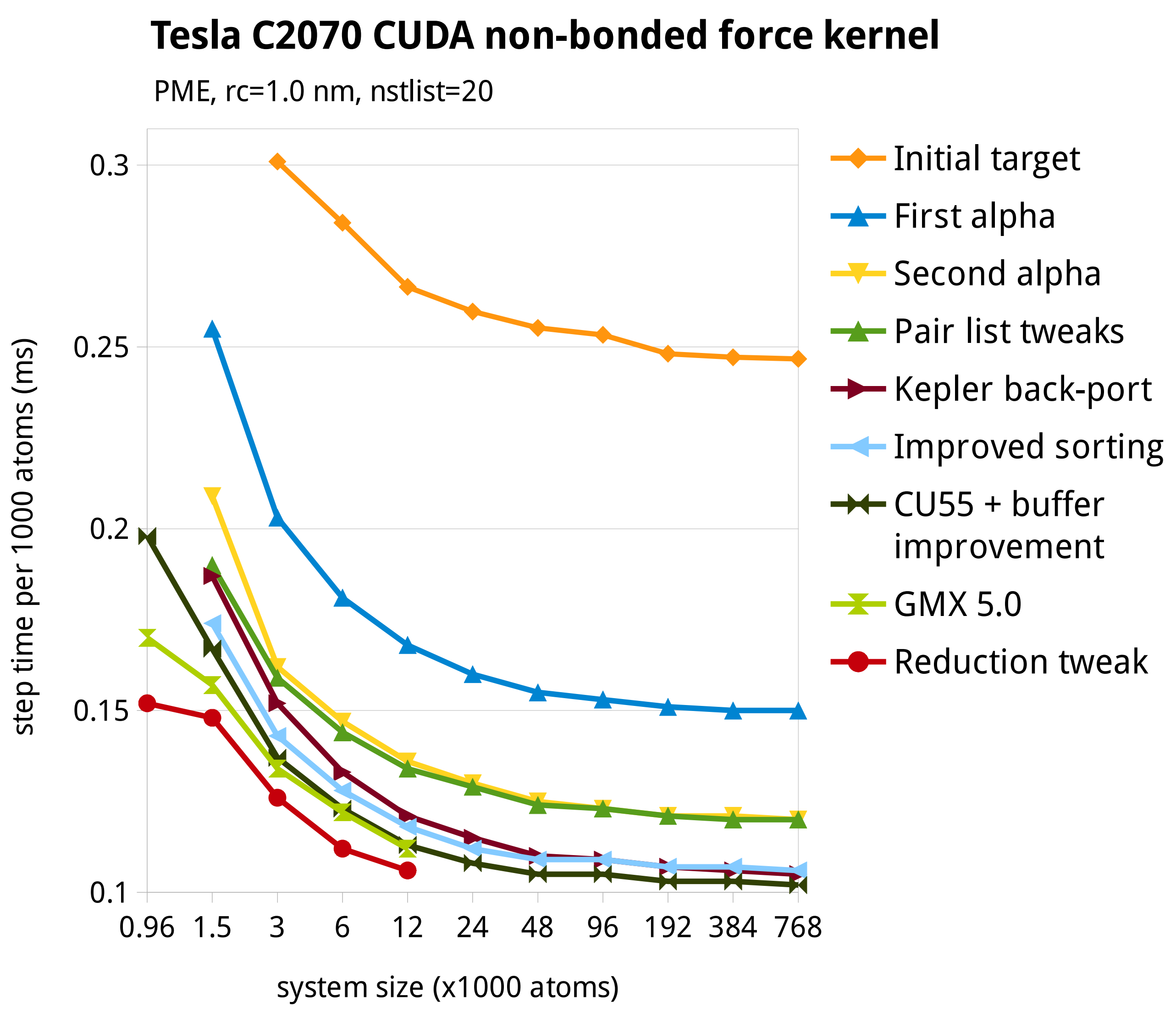


Tuning kernels: still getting faster

- Up to 2x faster wrt the first version

- We still keep finding ways to improve performance;

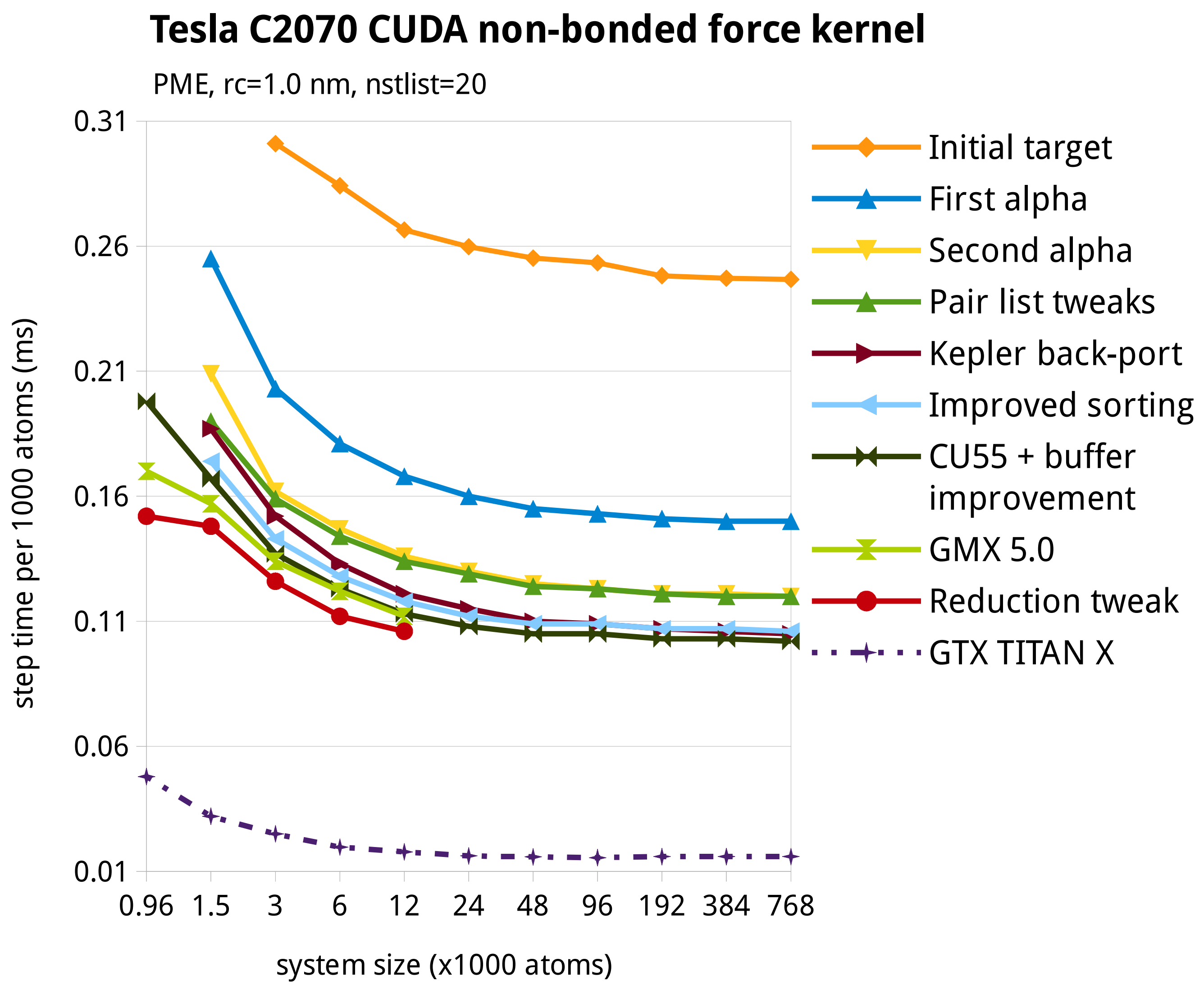
  most recently:
  - better inter-SM load balancing
  - more consistent list lengths
  - concurrent atomic operations



Tuning kernels: still getting faster

- Up to 2x faster wrt the first version
- We still keep finding ways to improve performance

  most recently:
  - better inter-SM load balancing
  - more consistent list lengths
  - concurrent atomic operations
- But NVIDIA does too!

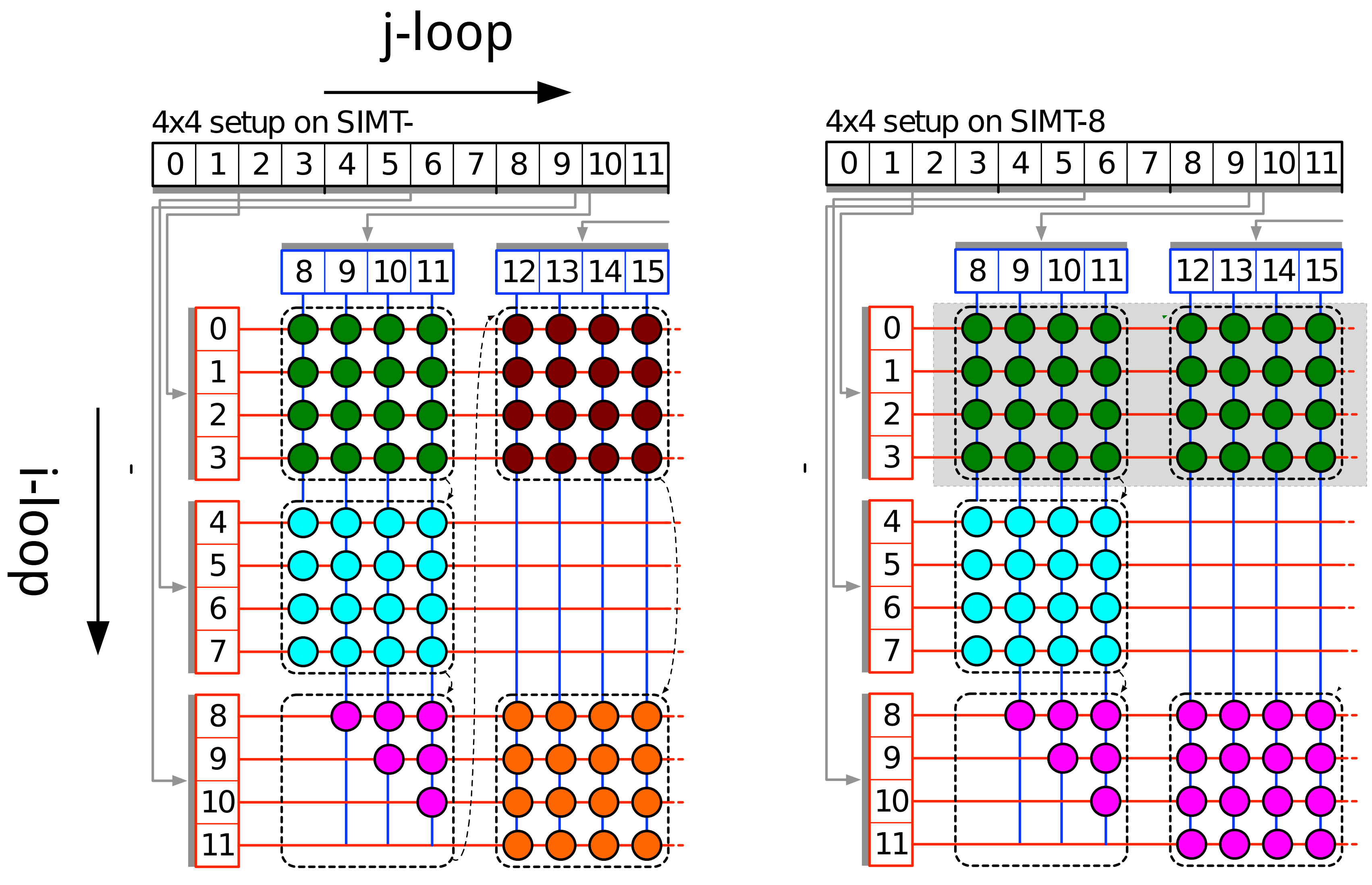


Adapting the cluster algorithm to the GK210

- Doubled register size

  => can fit 2x threads/block

  1 i-cluster vs **2 j-clusters**

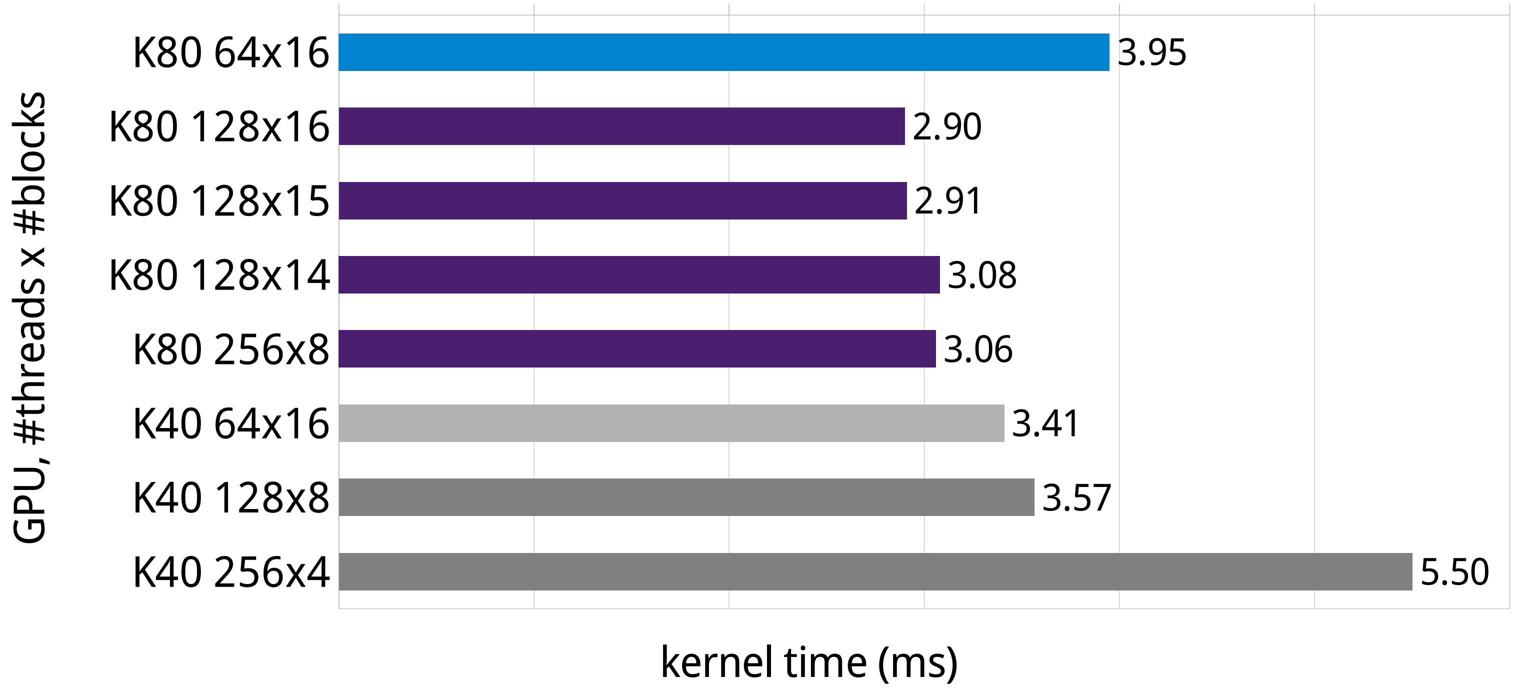


Particle cluster processing on GF, GK1xx, GM

Particle cluster processing on GK210

K80 kernel performance

- Pre-GK210: 64 threads/block, 16 blocks per SM
  occupancy: max 50%, achieved ~0.495%
- Extra registers allows 128 threads/block, still , 16 blocks per SM
  occupancy: max 100%, achieved ~0.92%



GPU Application clocks

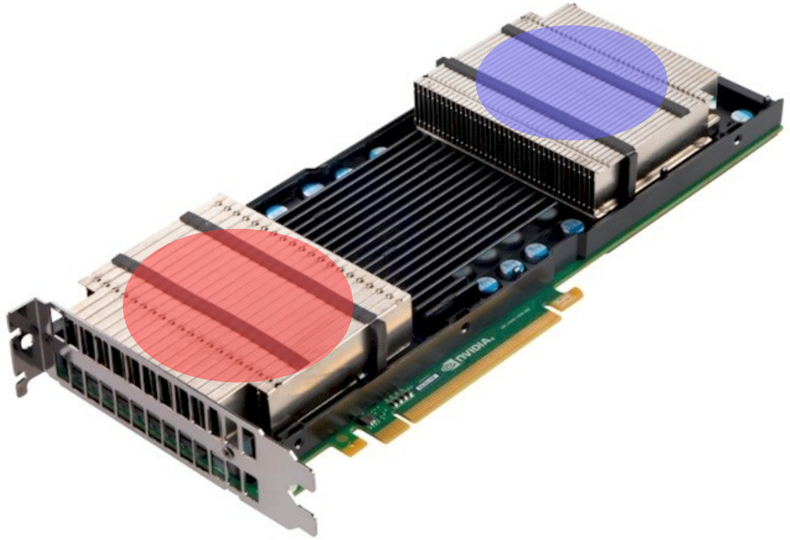
- Available on GK110[B], GK210 Tesla, Quadro (and GeForce if you're lucky)
  - First 2 gens of Kepler **moderate**: Tesla K20 7.5%, K40 17.5%
    - Lots of power/thermal headroom left: K40 peak ~155W (max 230W)
  - Tesla K80 **aggressive** boost: 562 Mhz → 875 Mhz (55.7%)
    - good for heterogeneous codes with alternating GPU utilization MD
    - can get close to the power limit (145 W)
    - Throttling-prone → load balance concern
- Fully supported in GROMACS 5.1 [contribution by: Jiri Kraus (NVIDIA)]
  - adjust clock at run-time or warn user if:
    - permissions don't allow
    - not linked against NVML

GPU Throttling

- K80: aggressive boost close to TDP + dual chip

  power consumption asymmetry

  → performance asymmetry



- Desktop cards: GeForce & Quadro
  - fan speed capped by default to <60%
  - throttle-prone: temperature limit (~80C)

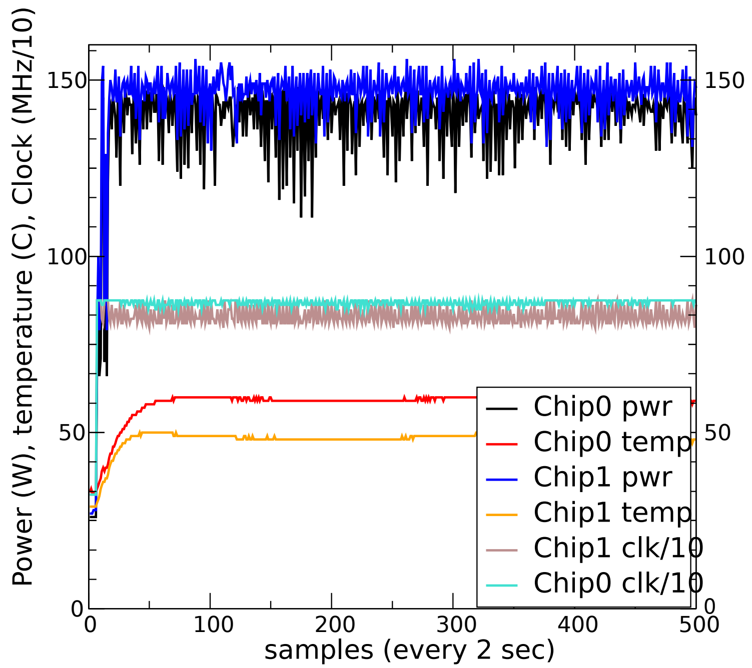
```
+-----------------------------------------------------+
| NVIDIA-SMI 346.47     Driver Version: 346.47        |
|-------------------------------+---------------------+
| GPU  Name        Persistence-M| Bus-Id       Disp.A |
| Fan  Temp  Perf  Pwr:Usage/Cap|        Memory-Usage |
|===============================+=====================+
|   0  GeForce GTX TITAN    On  | 0000:02:00.0    Off |
| 58%   80C    P0   191W / 250W |   107MiB /  6143MiB |
+-------------------------------+---------------------+
|   1  Quadro M6000         On  | 0000:03:00.0    Off |
| 54%   83C    P0   199W / 250W |   533MiB / 12287MiB |
+-------------------------------+---------------------+
```

GPU Throttling

- K80: aggressive boost close to TDP + dual chip

  power consumption asymmetry

  → performance asymmetry

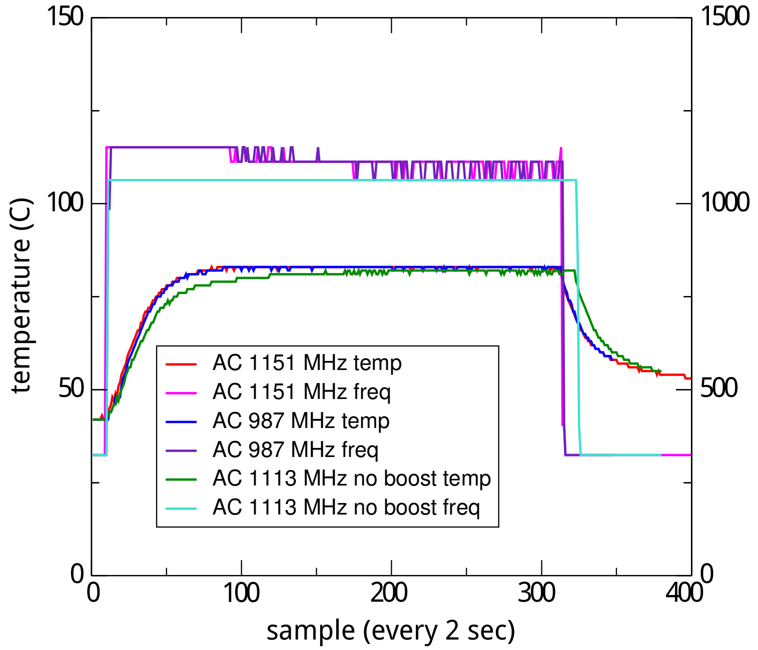


K80 Chip1:
hotter
slower
more hungry

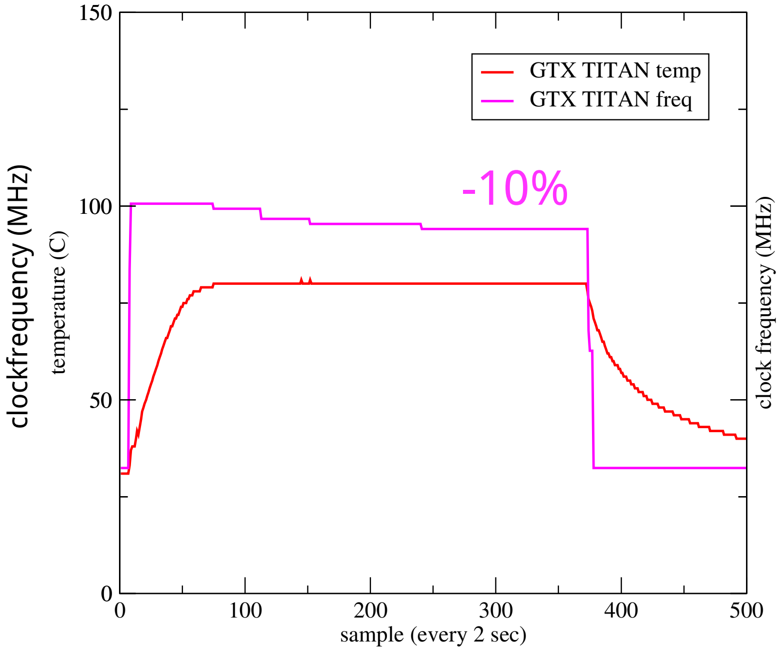
- Desktop cards: GeForce & Quadro
  - fan speed capped by default to <60%
  - throttle-prone: temperature limit (~80C)

=> load balancing issues

The Quadro M6000 throttles regardless of AC if auto-boost is on



The GTX TITAN can throttle by 10-20% in well-cooled chassis (non OC cards)



Tip: ~~set~~ force GPU fan speed manually

xorg.conf:

```
Section "ServerLayout"
        Identifier "dual"
        Screen 0 "Screen0"
        Screen 1 "Screen1" RightOf "Screen0"
EndSection

Section "Device"
        Identifier      "nvidia0"
        Driver          "nvidia"
        VendorName      "NVIDIA"
        BoardName       "GeForce GTX TITAN"
        Option          "UseDisplayDevice" "none"
        Option          "Coolbits" "4"
        BusID           "PCI:2:0:0"
EndSection

Section "Device"
[...]
EndSection

Section "Screen"
        Identifier      "Screen0"
        Device          "nvidia0"
EndSection

Section "Screen"
[...]
EndSection
```

Start the X server

```
export DISPLAY=:0

/usr/bin/X $DISPLAY -nolisten tcp vt7 -novtswitch &

nvidia-settings -a [gpu:0]/GPUFanControlState=1 -a
[fan:0]/GPUCurrentFanSpeed=80

nvidia-settings -a [gpu:1]/GPUFanControlState=1 -a
[fan:1]/GPUCurrentFanSpeed=80
```

```
+------------------------------------------------------+
| NVIDIA-SMI 346.47     Driver Version: 346.47         |
|-------------------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage |
|===============================+======================+
|   0  GeForce GTX TITAN   On   | 0000:02:00.0     Off |
| 80%   69C    P8   205W / 250W |    549MiB /  6143MiB |
+-------------------------------+----------------------+
|   1  Quadro M6000        On   | 0000:03:00.0     Off |
| 80%   66C    P0   203W / 250W |    533MiB / 12287MiB |
+-------------------------------+----------------------+
```

Thanks to Stefan Fleischmann for figuring out the details!

OpenCL port

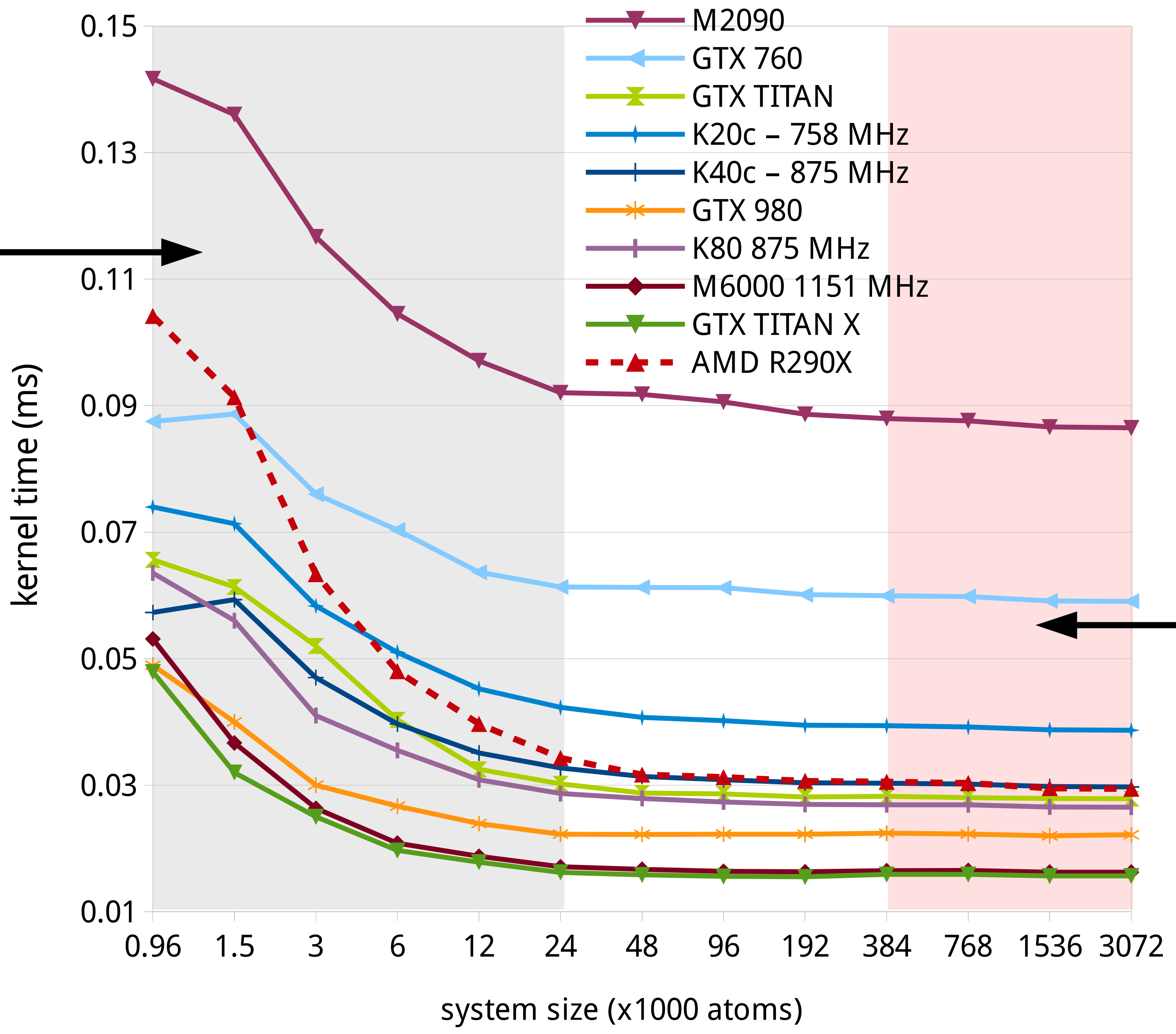
- **Collaboration with Streamcomputing**
- GROMACS is highly portable across CPUs but **not across accelerators**
- **Goals:**
  - Improve portability
  - Wide user-base: allow using the hardware
- AMD & NVIDIA GPUs supported
- Status: merge into v5.1 pending

- **Lessons learned:**
  - AMD OpenCL:
    - AMD R9 290X ~ GTX 970
      ~200W vs 145W
      (-25% wrt 980)
  - NVIDIA OpenCL **severely lacking**
    - v1.1?
    - performance: 2-3x lower than CUDA

Kernel performance and scaling

Strong scaling regime:

This is where most of our efforts go!

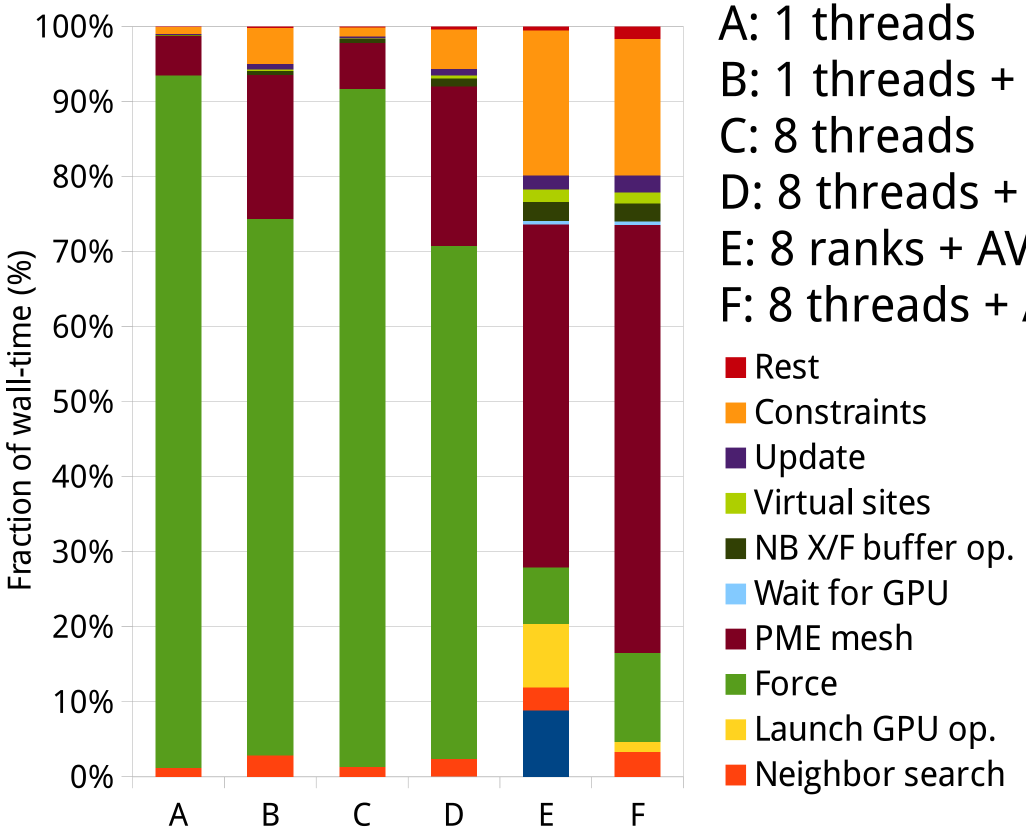


Benchmark “show-off” regime:

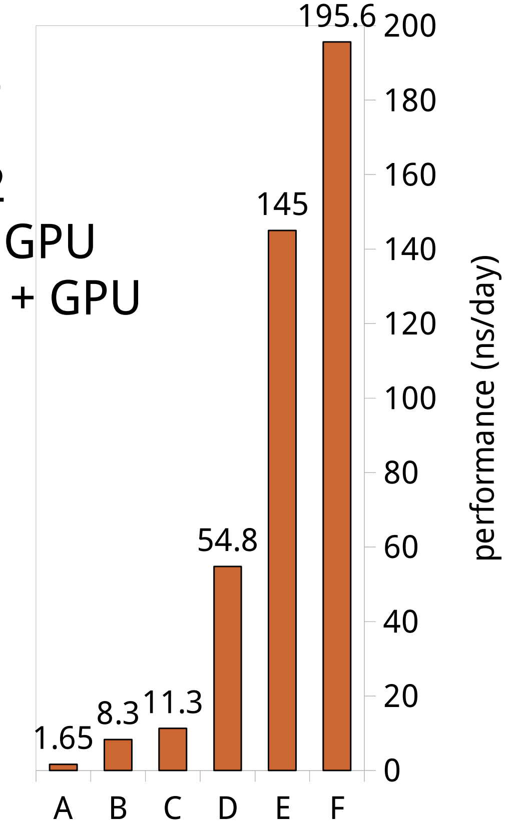
This is where the “free lunch” from new hardware comes in full effect

Intra-node:
CPU+GPU

CPU SIMD and threading



A: 1 threads
B: 1 threads + AVX2
C: 8 threads
D: 8 threads + AVX2
E: 8 ranks + AVX2 + GPU
F: 8 threads + AVX2 + GPU



Input: VSD ion channel embedded in a membrane 47k atoms
Settings: PME, cut-off >=1 nm, 5 fs, all bonds constrained

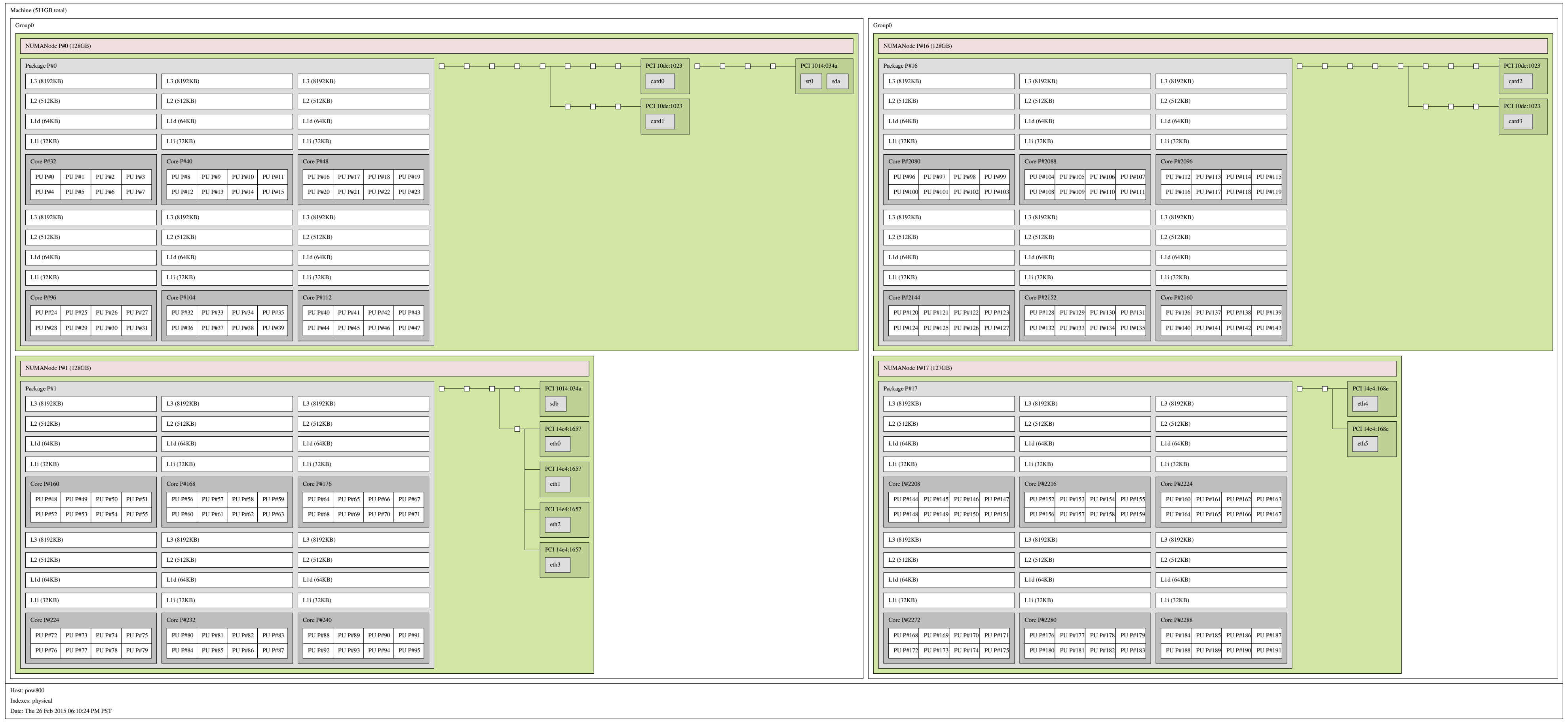
Hardware:
Core i7 5960X &
GeForce GTX 980

- SIMD support:
  - x86: SSE2, SSE4.1, AVX (+FMA4), AVX2, AVX-512
  - ARM: Neon, Neon-ASIMD,
  - IBM: QPX, VSX, VMX
  - Sparc64

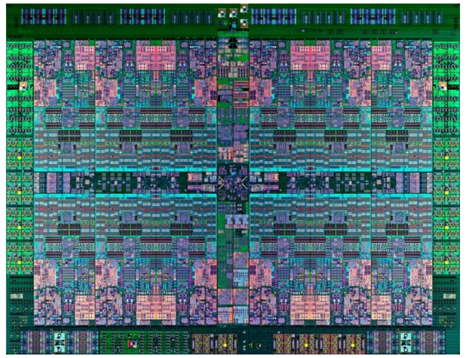
  → Facilitated by GROMACS' generic SIMD layer

- Threading: OpenMP - the lesser evil
  - challenges: increasing core/hardware thread count

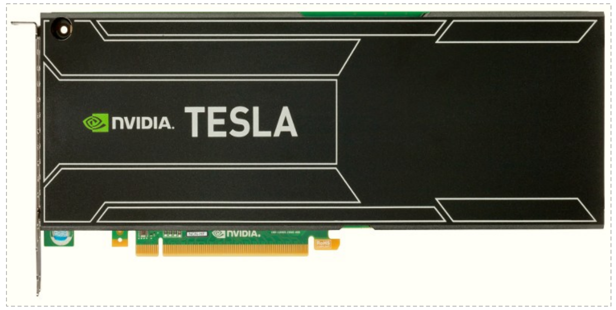
Power8



POWER8



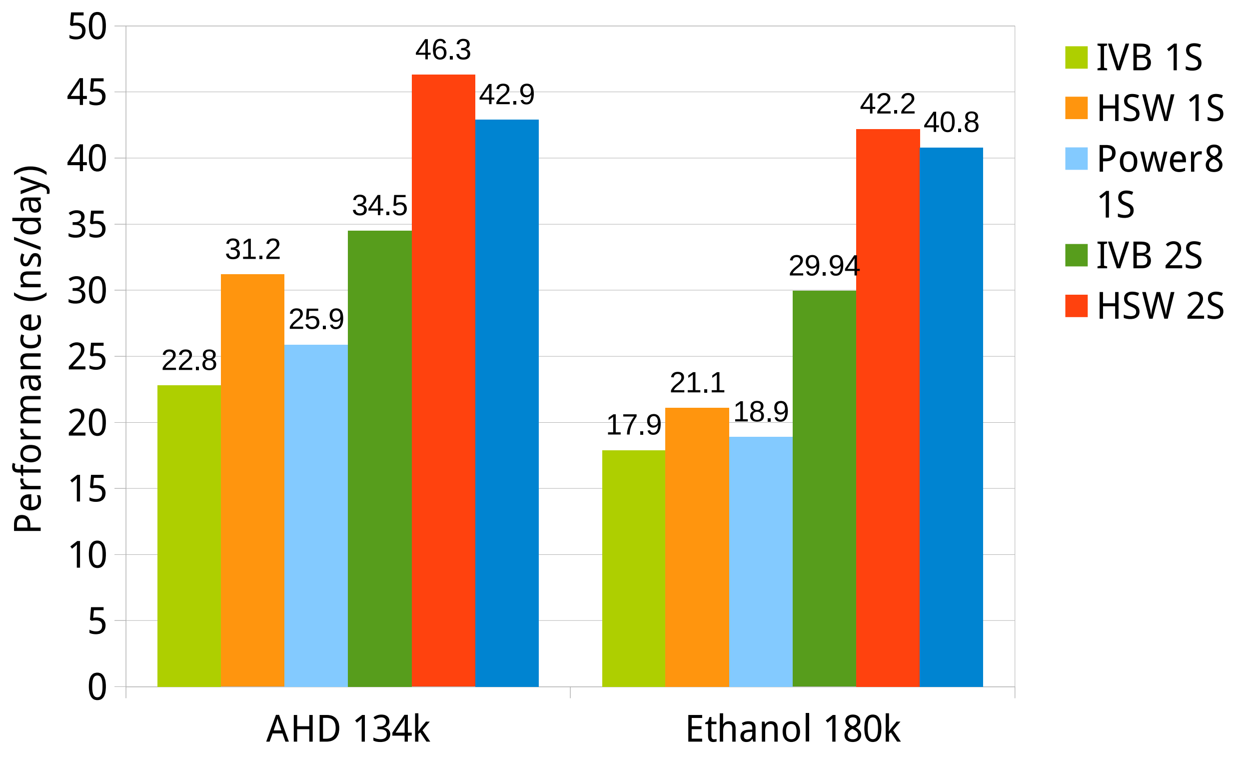
+

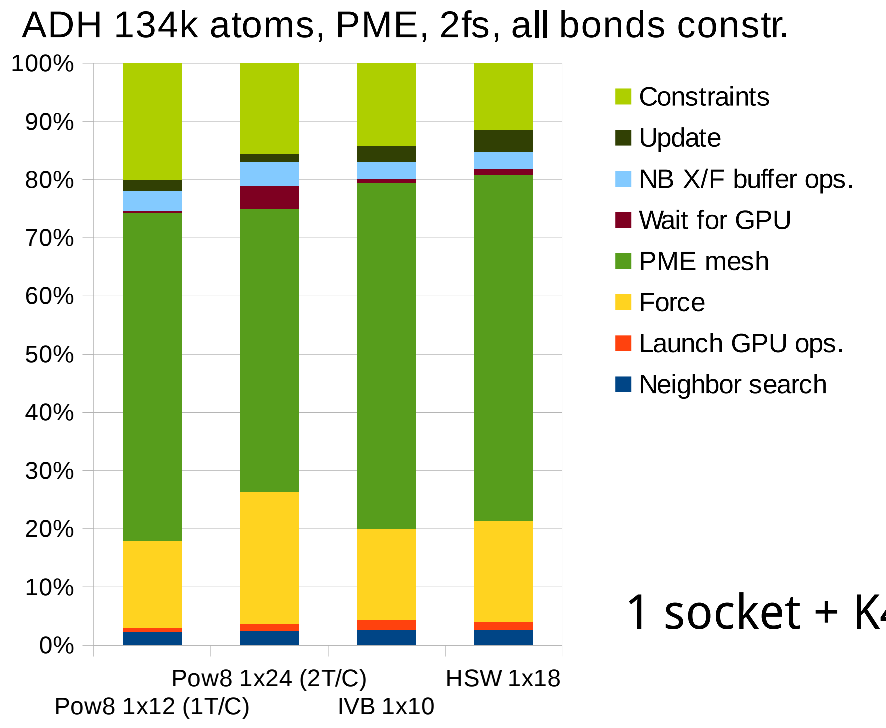


- Power8/8+ and OpenPower: ~~a~~ **the only** promising competition for Intel!
- I hope that the lessons of the past are used to the best possible extent

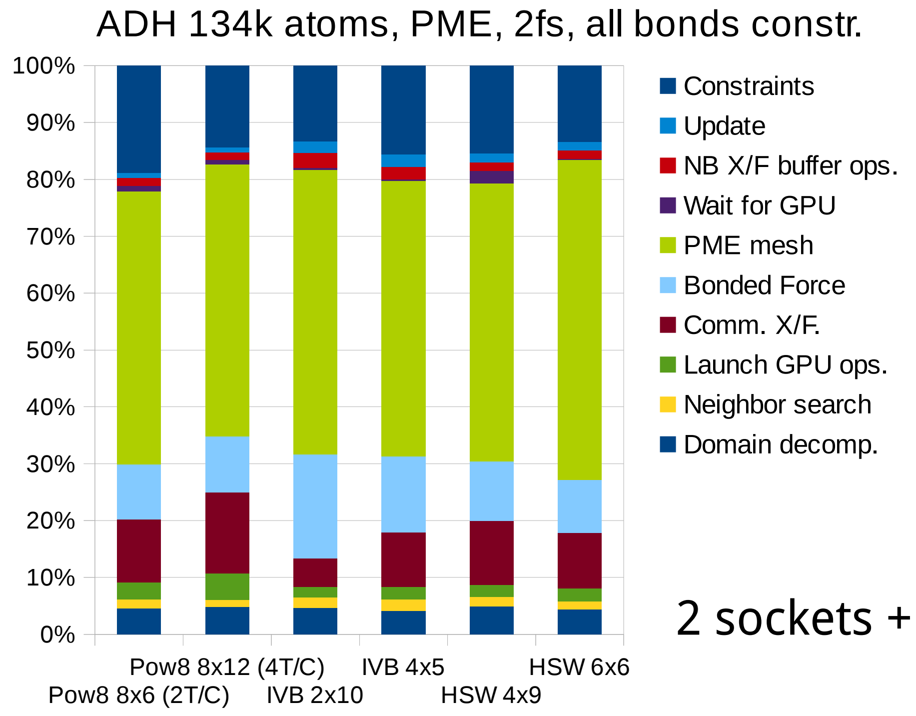
  (Blue Waters, BLue Gene, x86 servers)

Power8 vs IVB / HSW





1 socket + K40



2 sockets + 2 K40s

Hardware: 2 socket nodes

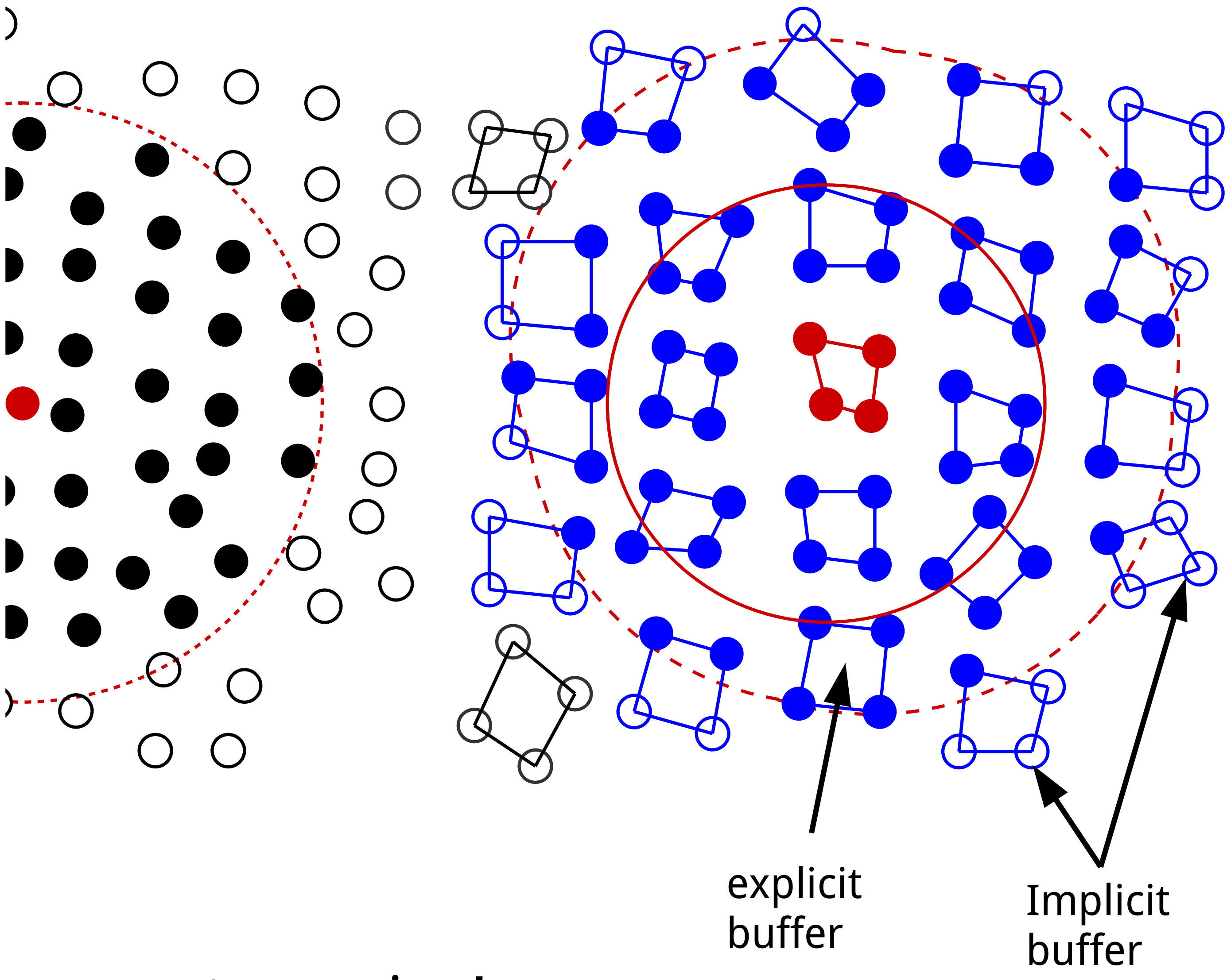
- Power8: Power8 PSG node 12 cores @ 4 GHz (? W)
- IVB: Intel Xeon E5 2690 v2 10 cores @ 3.0 GHz (2x130W)
- HSW: Intel Xeon E5-2699 v3 18 cores @ 2.3 GHz (2x145W)

Buffering & Calculating useful zeros

- Given: target drift & interaction cut-off
- Automated buffer estimate based on:
    - atomic displacement distribution
    - potential at cut-off,
    - constraints, vsites,...
- Clusters crossed by the cut-off

  → **implicit buffer**

  allows r_{list} = 0 in some cases*



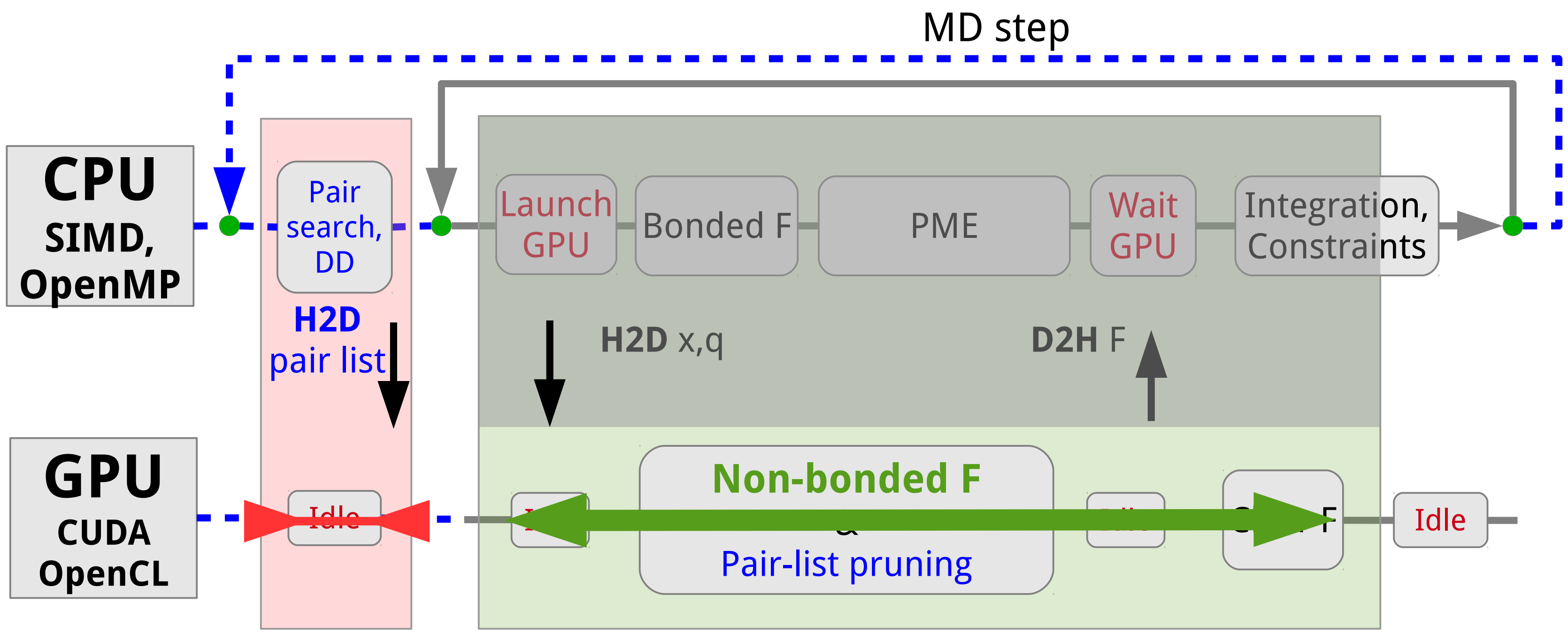
- Much tighter estimates in 5.0, further improvements coming!

Páll, S., & Hess, B. (2013). Com. Phys. Comm., 184(12), 2641–2650. http://doi.org/10.1016/j.cpc.2013.06.003

*Buffer size or list cut-off in GROMACS terminology

Pair list rebuild frequency: **tunable!**

- **Goal**: increase CPU-GPU overlap
  - search less often => larger buffer
  - cost tradeoff: pair search + domain decomposition vs non-bonded work
- Based on physics/math not guessing!



More CPU-GPU overlap:
higher GPU utilization

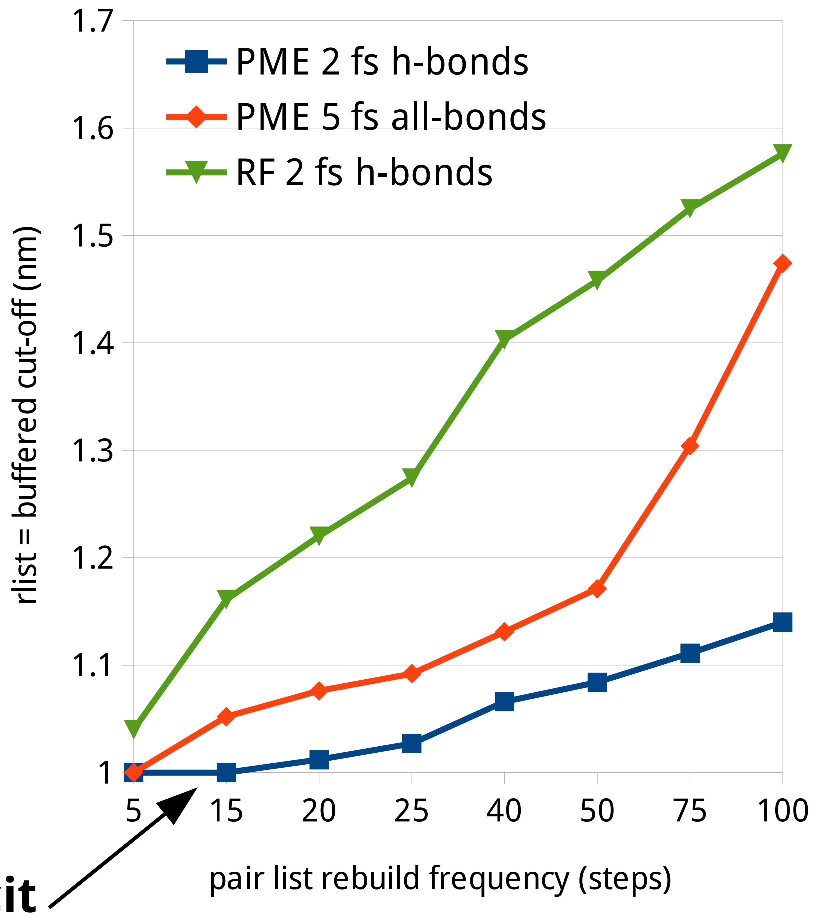
Pair list rebuild frequency: **tuning**

The cost of less frequent list updates:
- increasing buffer size

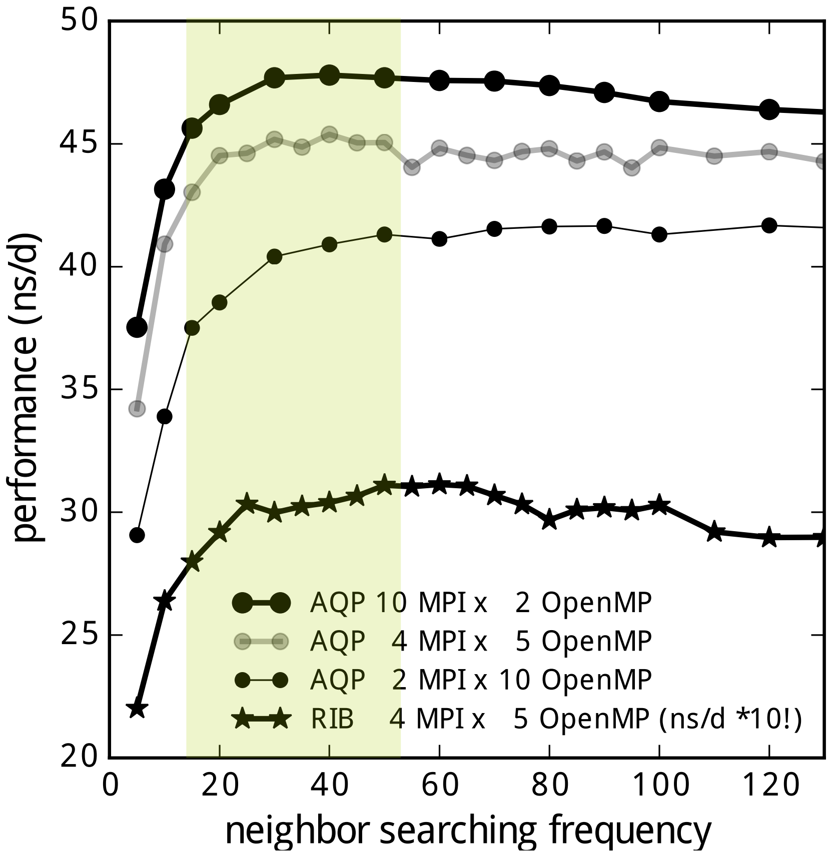
→ increasing non-bonded cost

In practice:
- GPUs are fast,
- rel. throughput increases with nstlist

  → optimal value: 15-50

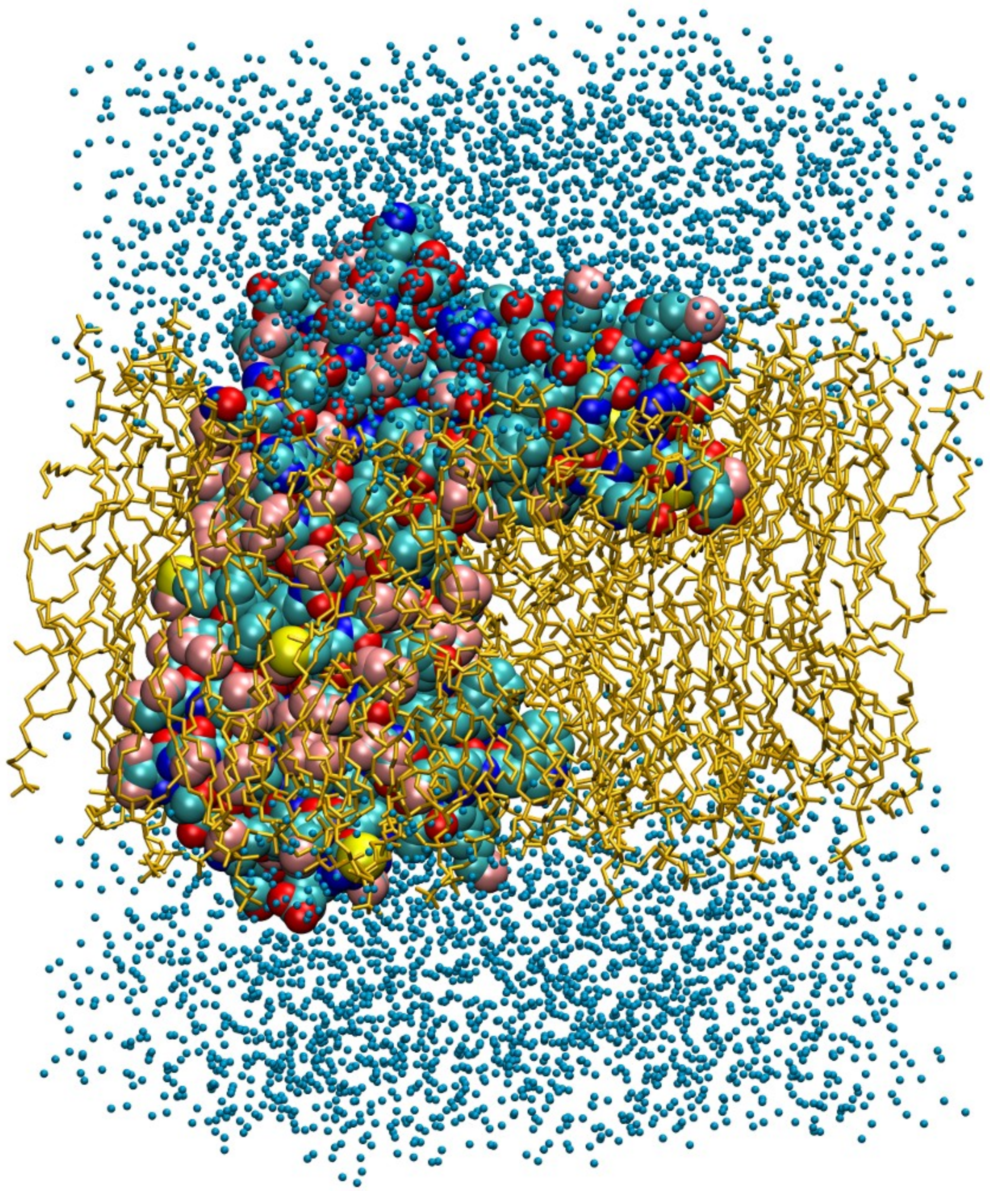


No **explicit** buffer needed

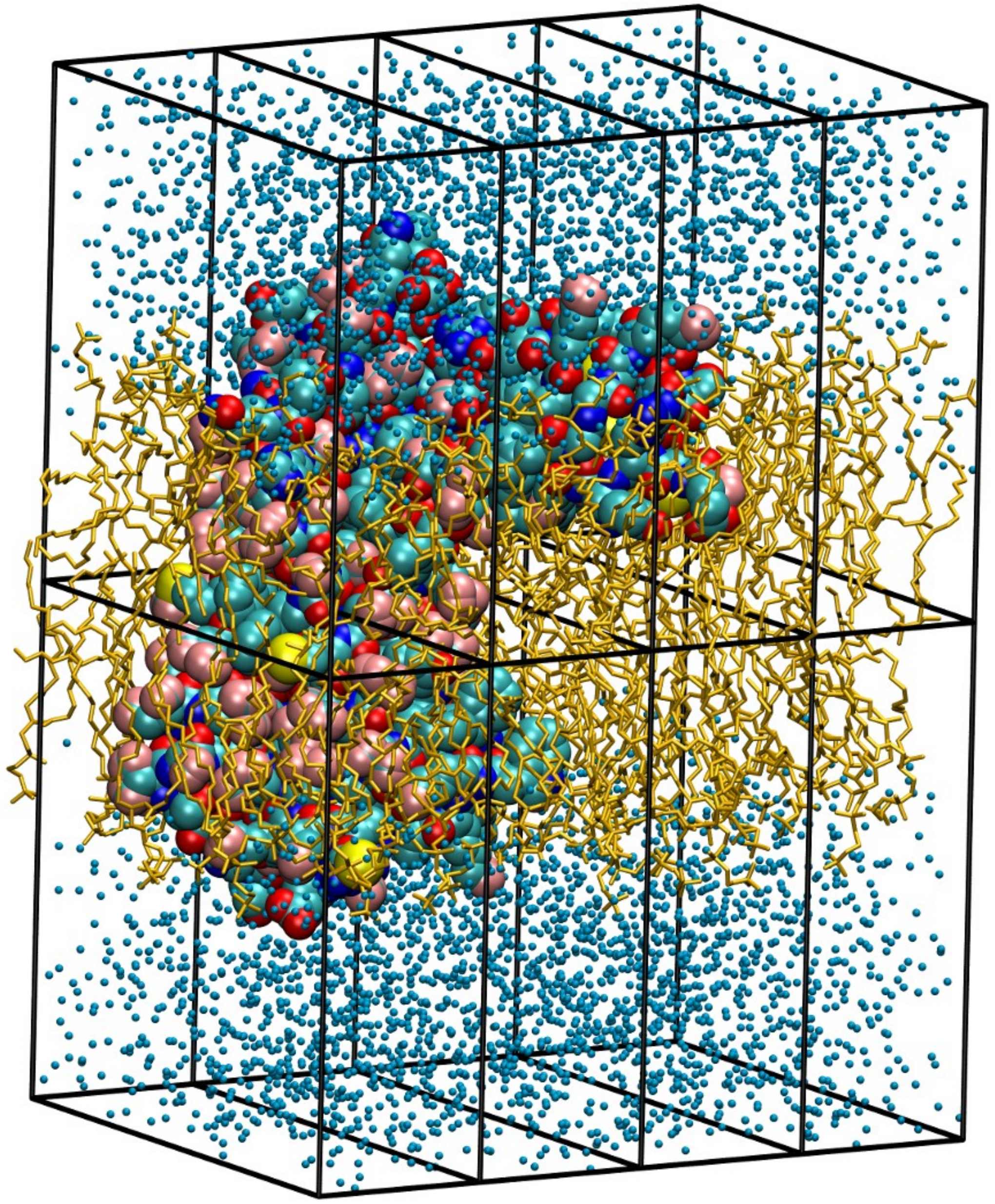


Multi-
CPU+GPU

Domain decomposition

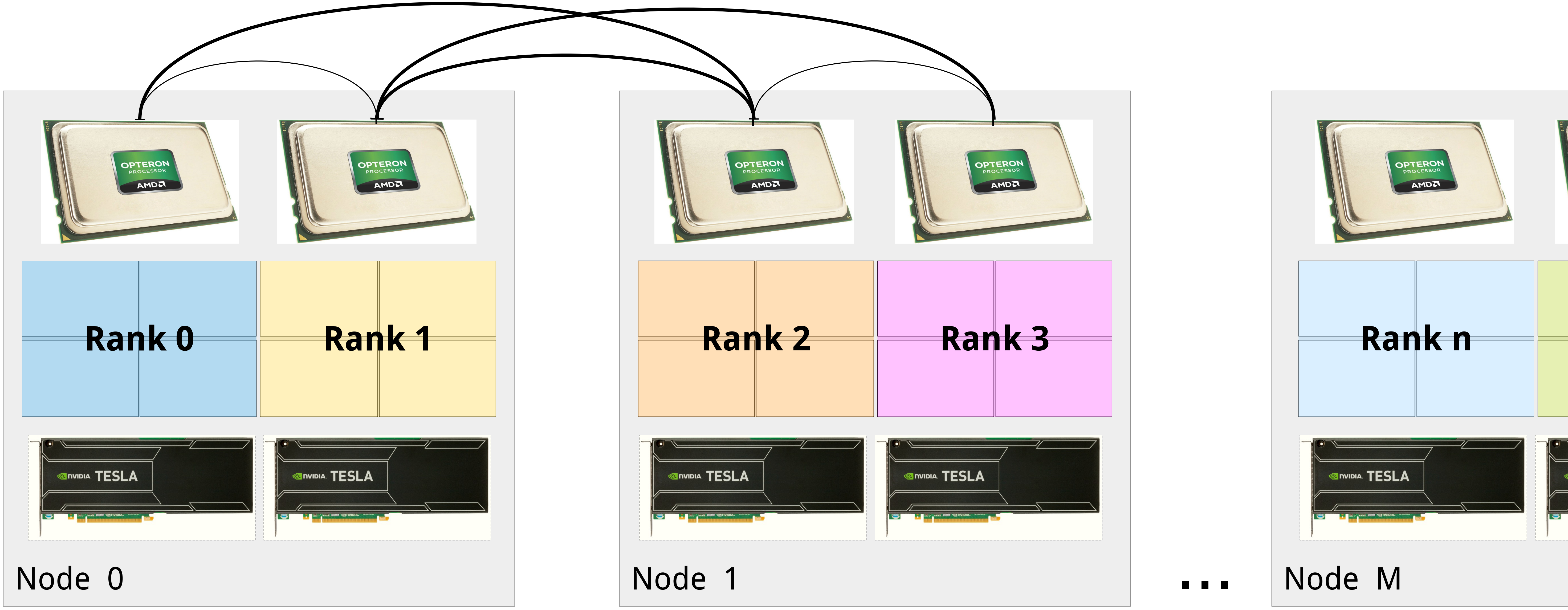


DD
initialization

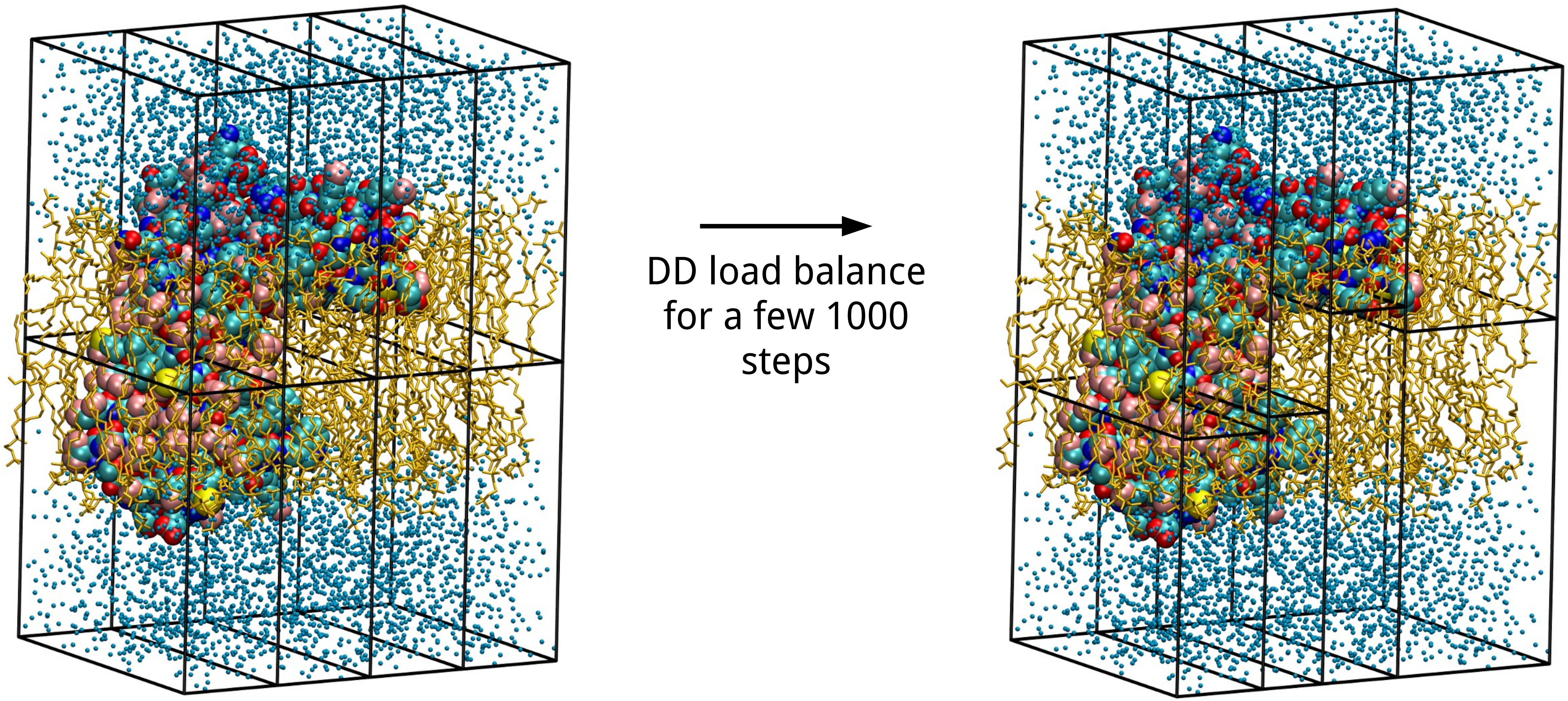


Kv2.1 Voltage Sensor Domain (VSD)

Schematics of CPU-GPU assignment



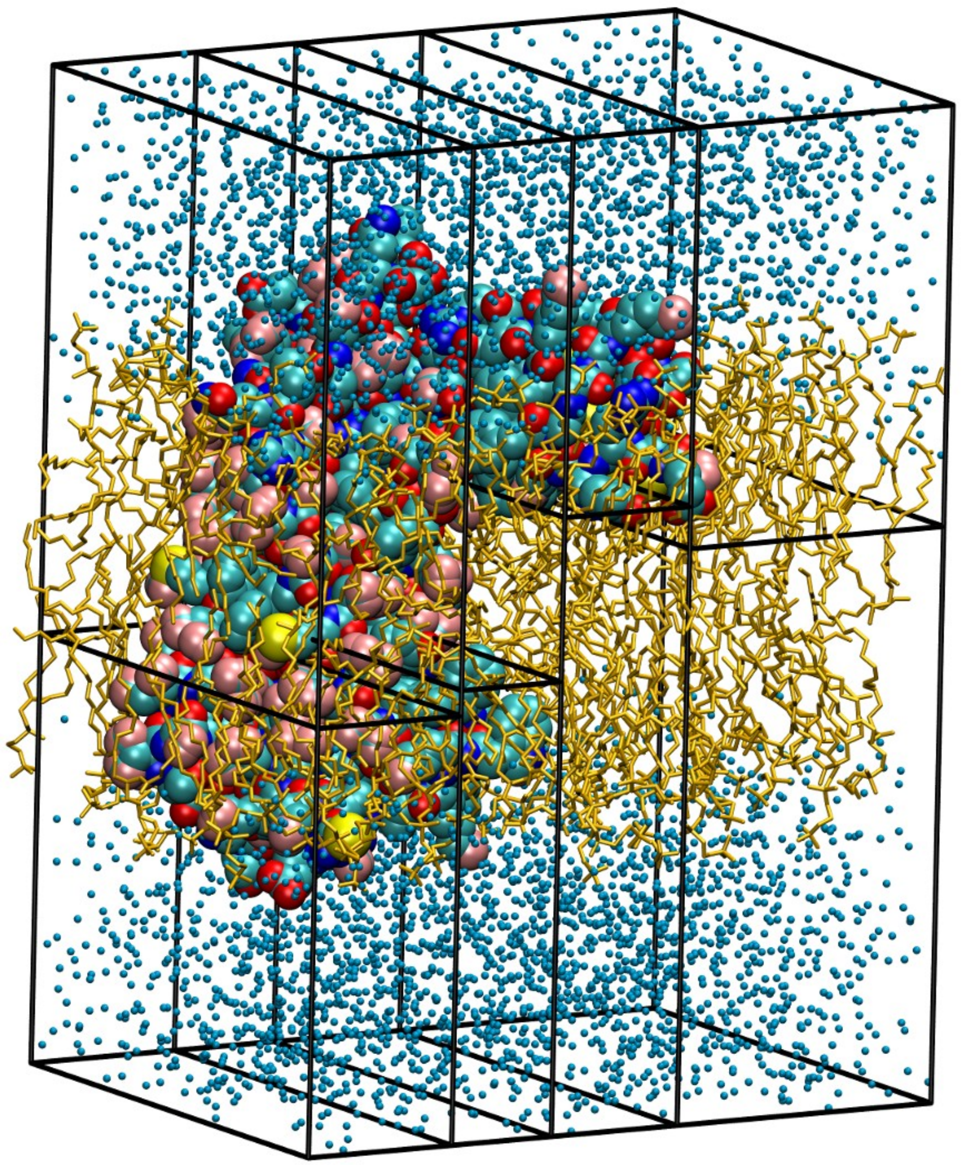
Dynamic load balancing (DLB)



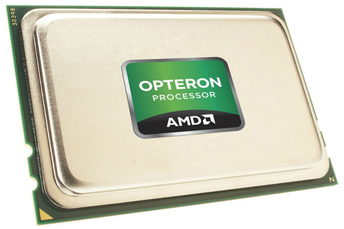
Dynamic load balancing (DLB)

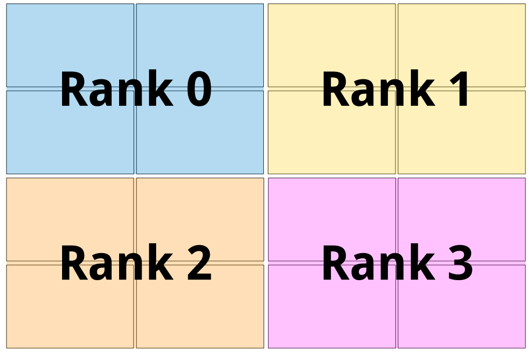
- Fully automated
  - measure force load per rank
  - shift cell boundaries every N steps
  - fast & eager algorithm

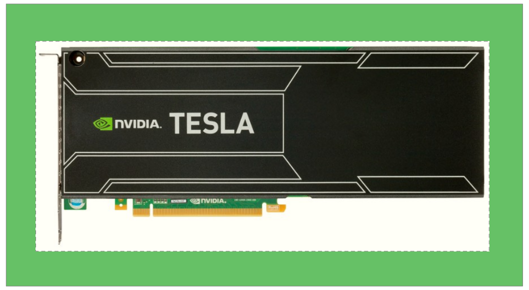
- Uses cycle counters
  - rdtsc/rdtsc
  - sync or actual clock needed



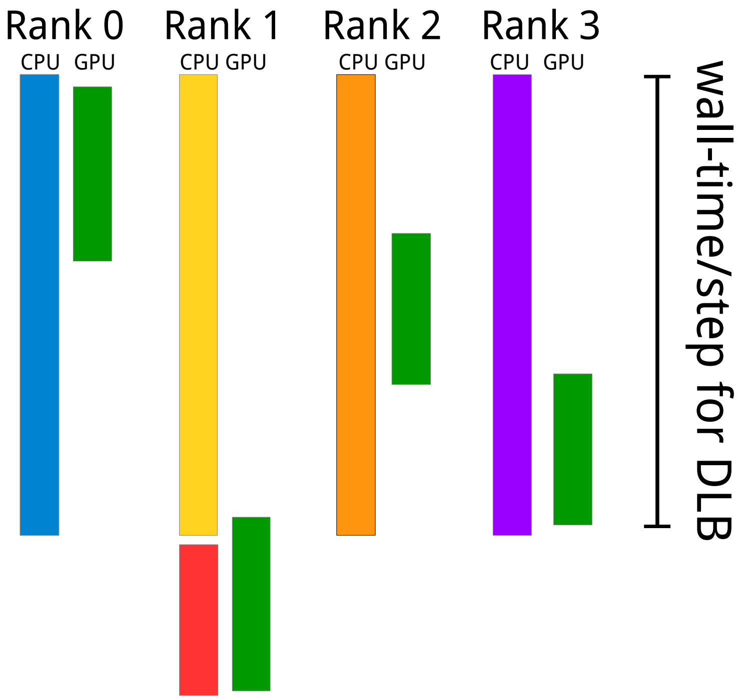
DLB with ranks sharing a GPU







4 ranks x 4 cores each
sharing a single GPU



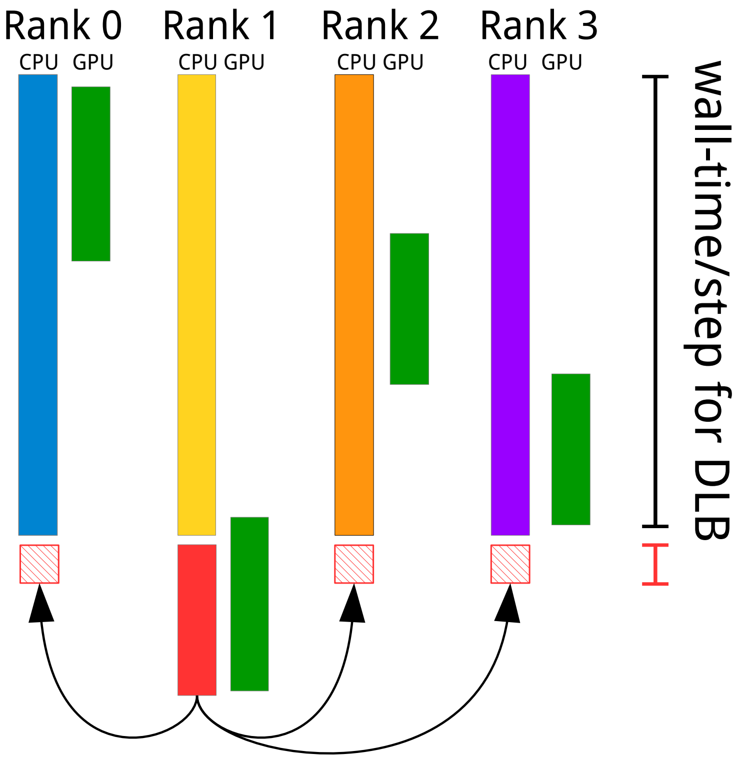
The unlucky rank waits!

But should we balance on this?

- **Can't split the GPU like a CPU**
  - executed ~serialized
    => lucky rank gets scheduled 1st
  - use custom block scheduling?
- **Benefits:**
  - #NUMA regions > #GPUs
  - reducing multi-threaded bottlenecks
  - overlap comm/kernel from different ranks
- **Artificial load imbalance** on ranks sharing a GPU

DLB with ranks sharing a GPU

4 ranks x 4 cores each sharing a single GPU



The unlucky rank waits!

But should we balance on this?

Consistently unlucky ranks would get overloaded!

- **Solution:** make it look like they are not sharing
- **Challenges**:
  - Can't measure actual kernel times with multiple streams!

    (CUDA ~~feature~~ bug)
  - Need to resort to the crude method: redistribute the wait

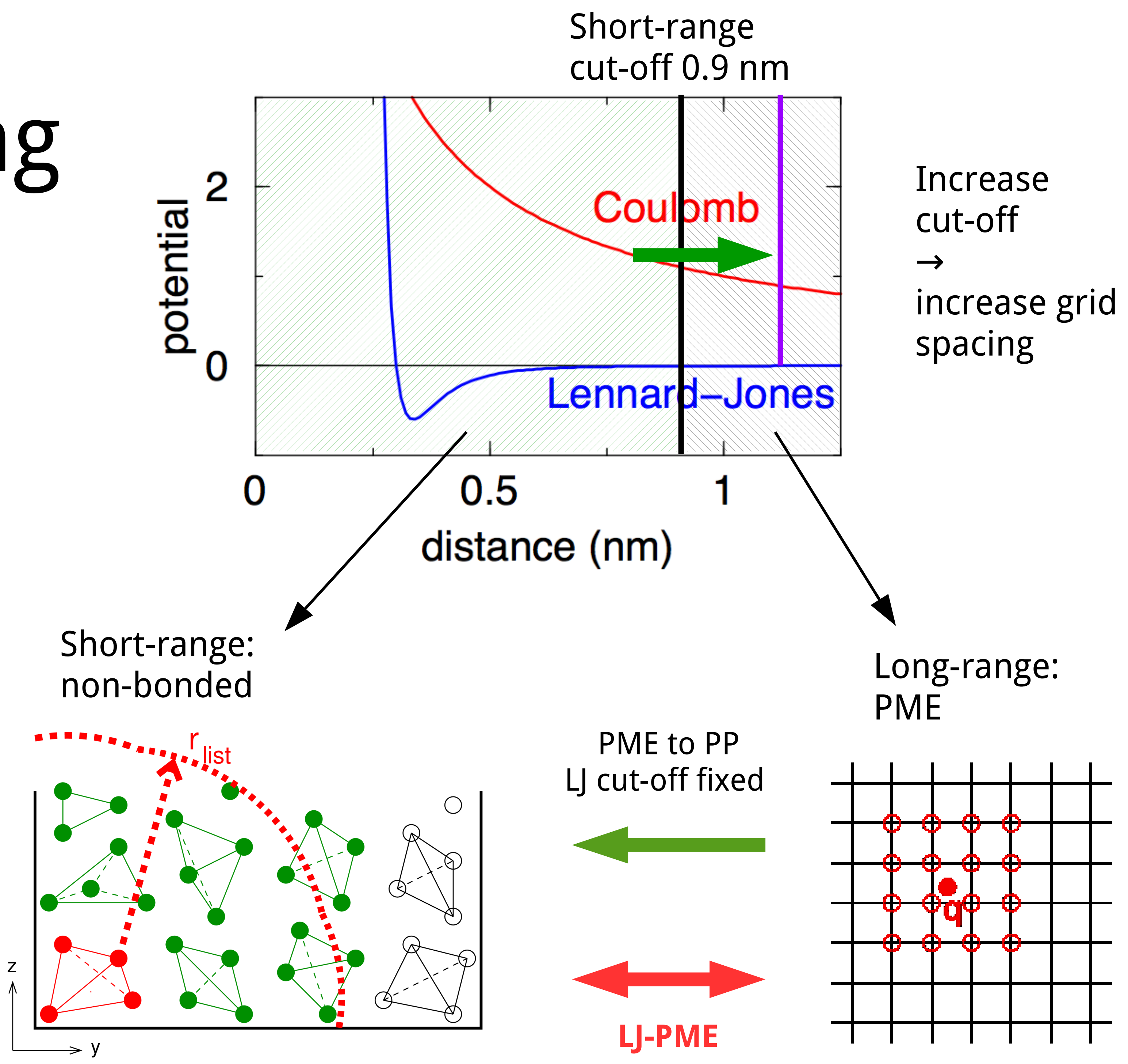
PP-PME load balancing

- Task load balancing:

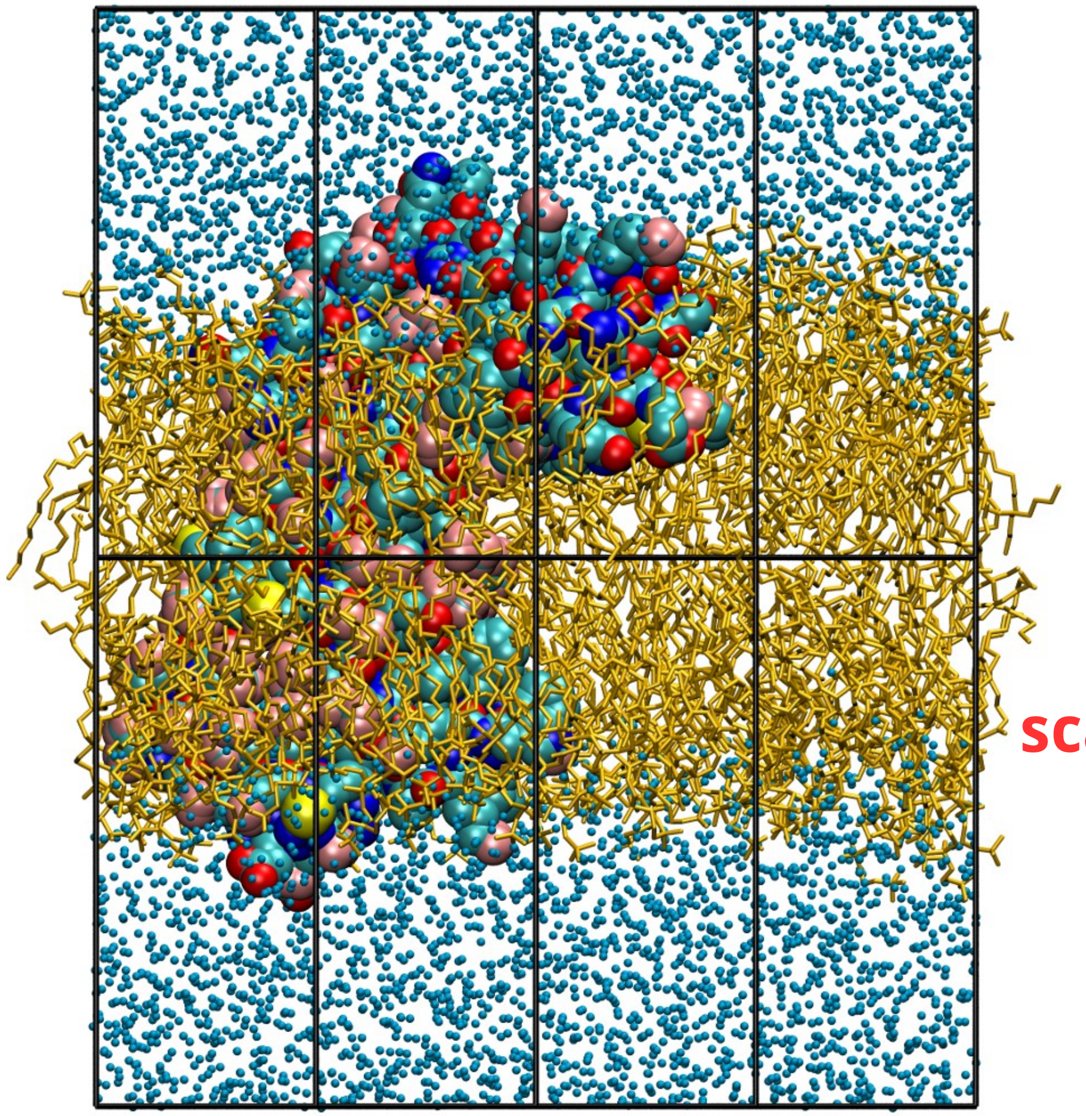
  shifts work from long- to short-range
  - MPMD : PP – PME ranks
  - non-bonded offload: CPU-GPU

- Need to keep LJ cut-off fixed!
  - Twin cut=off kernel (3-5% slower)
  - LJPME in v5.0 allows scaling both

- Scaling tradeoff!



DLB + PP-PME load balancing

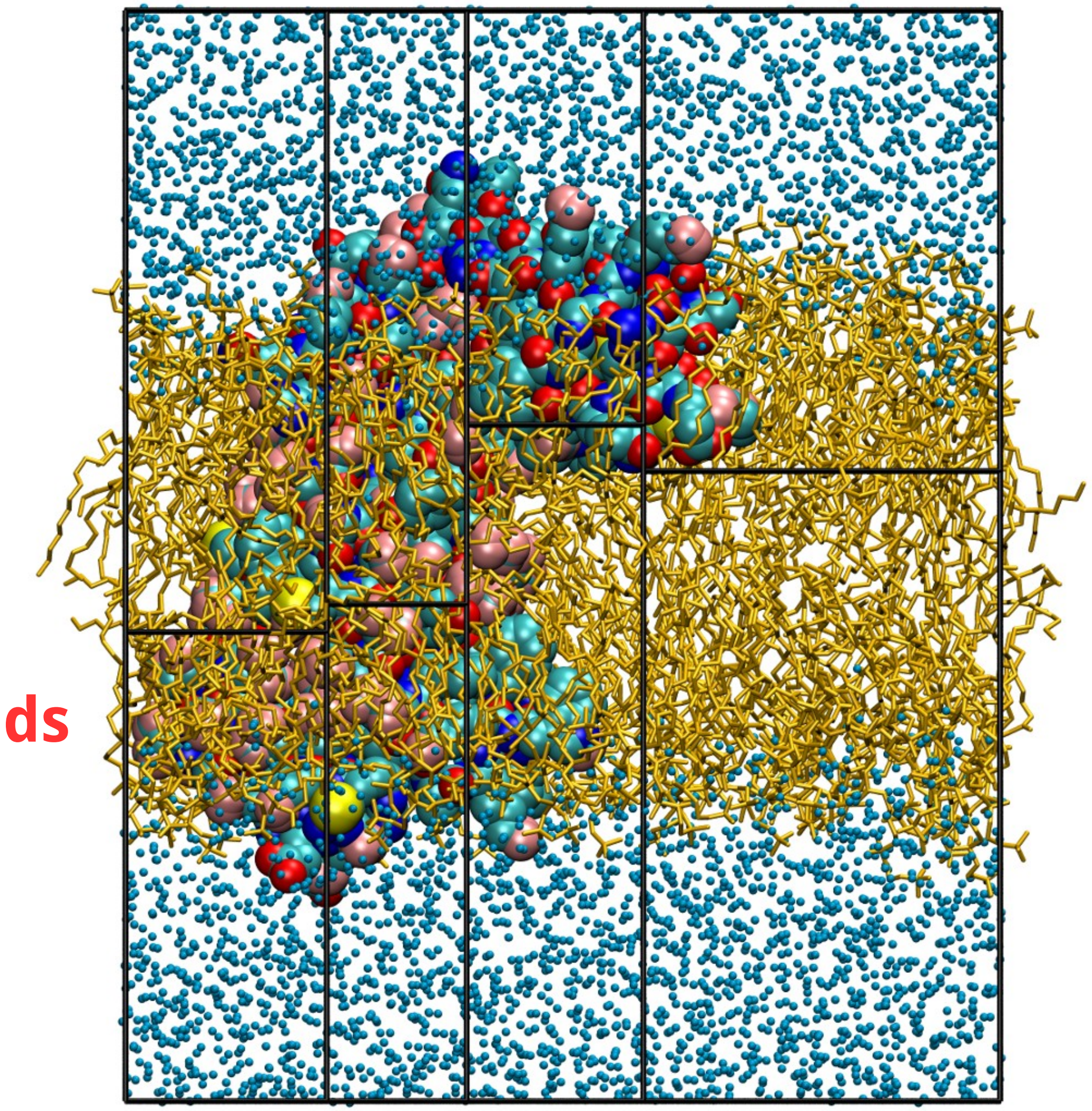


step 0 uniform DD grid

DD load balance
for a few 1000s
of steps

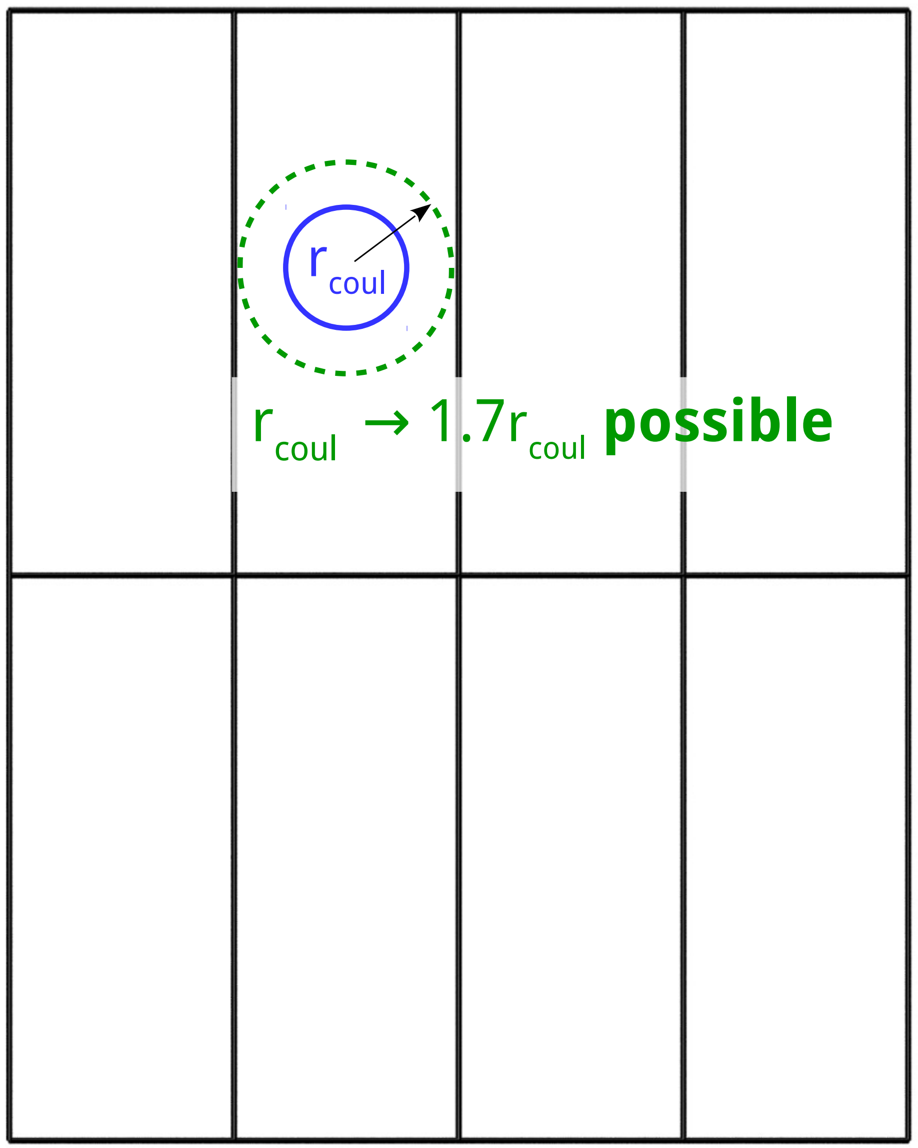
**At the same time,
scan the possible
scaled cut-offs/PME grids**

Let's try!



step 5000 DD grid: non-
uniform, staggered

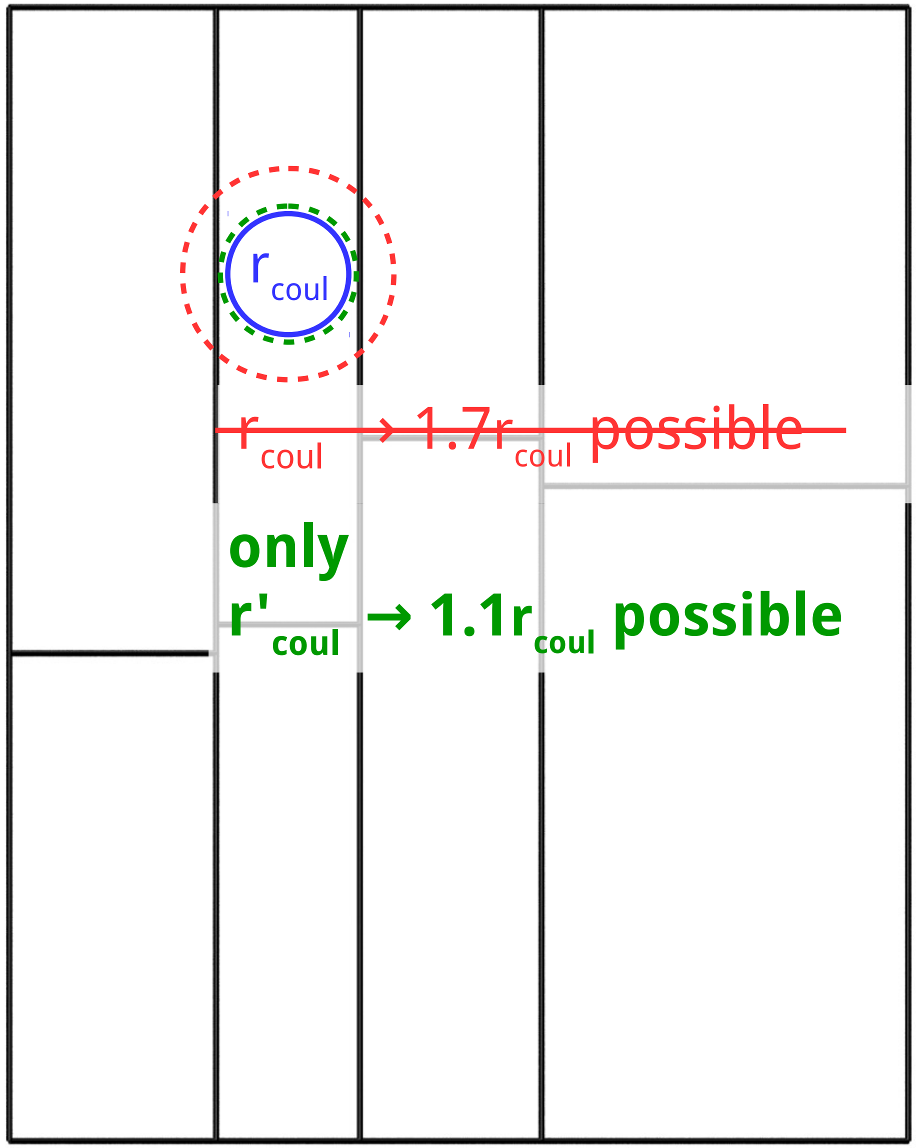
DLB + PP-PME load balancing: unwanted interaction



step 0
Uniform DD grid

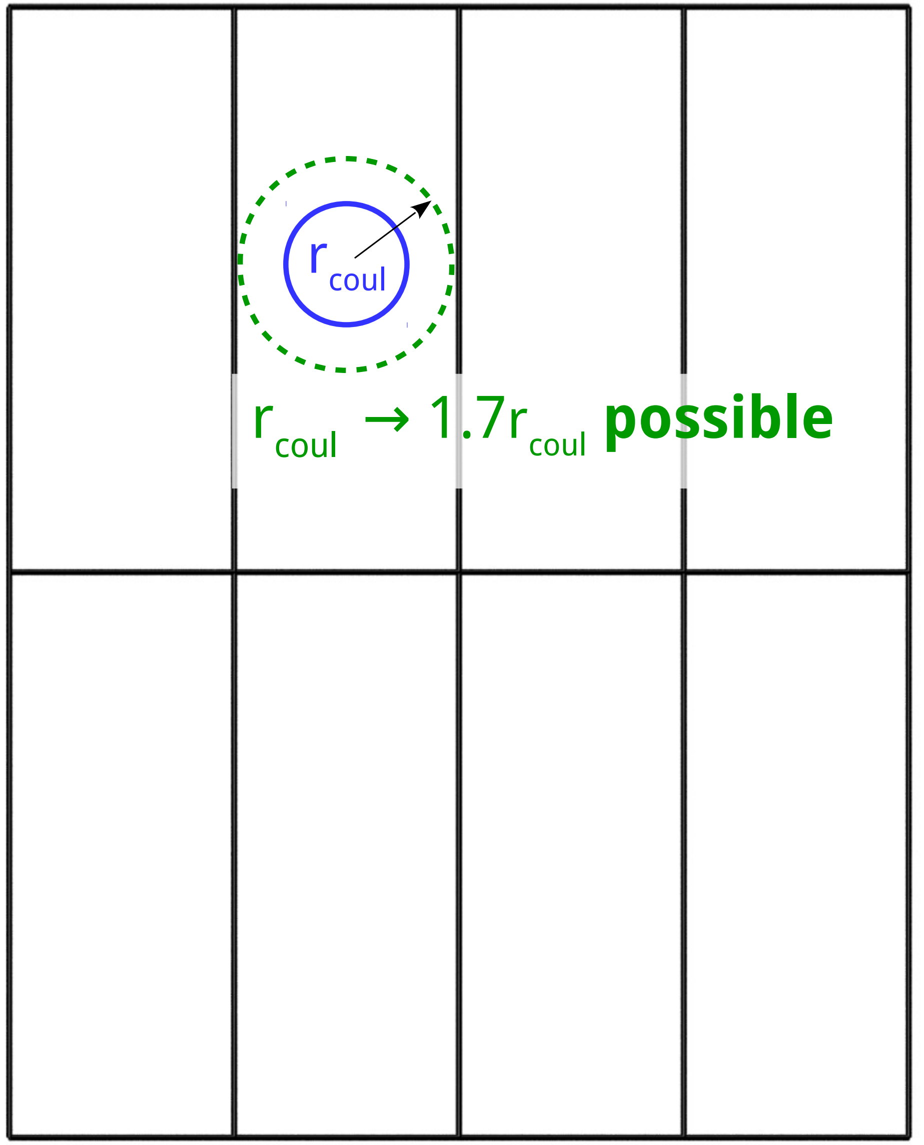
DD load balance for a few 1000s of steps

Scanning of the possible increased the cut-offs gets **limited by the new cell size**!



step 5000
staggered DD grid

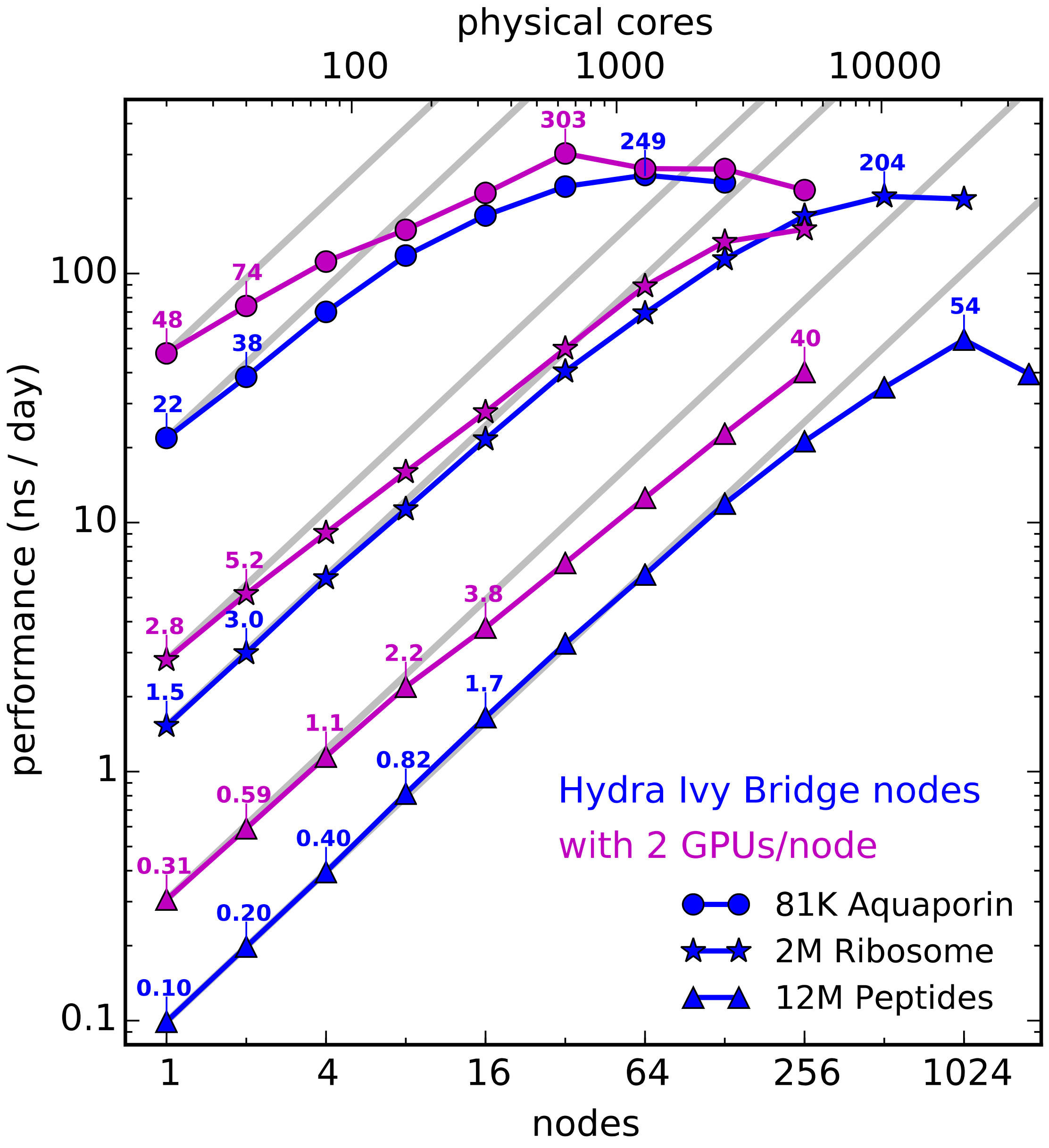
DLB + PP-PME load balancing: eliminating unwanted interaction



step 0
Uniform DD grid

- **Solution: multi-stage load-balancing**
  1. lock DLB
  2. scan for scaled cut-off/PME grid setups
  3. unlock DLB
  4. Re-scan cut-off/PME grid setups with preserving minimum cell size
  5. (repeat 4 periodically/if CPU-GPU load imbalance is observed)

GROMACS scaling: Hydra



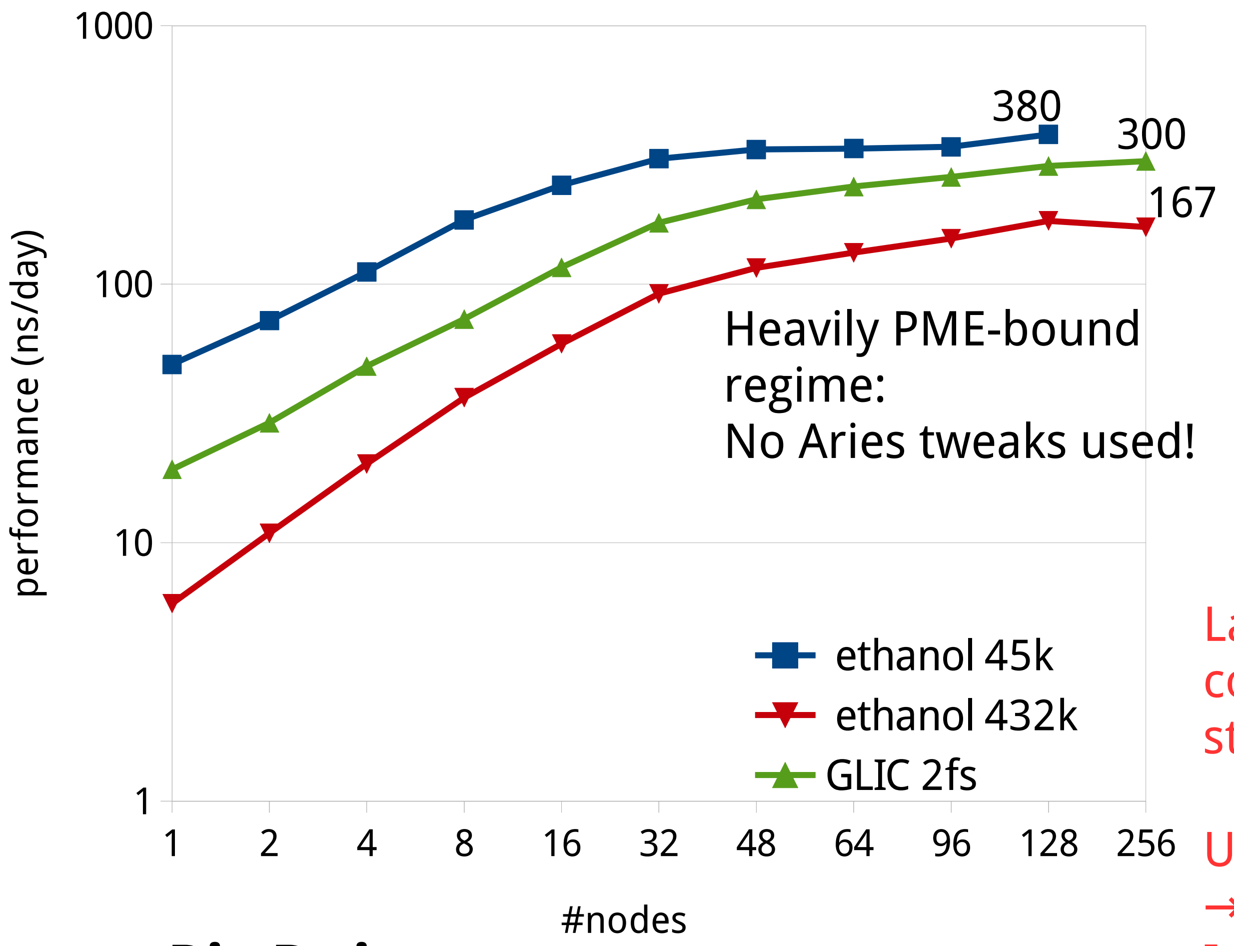
- Using GROMACS 4.6

- Hardware:

  Xeon E5-2680 v2 + K20X

- Load balancing issues well-illustrated: partly addressed since!

Cray XC30 performance: Piz Daint

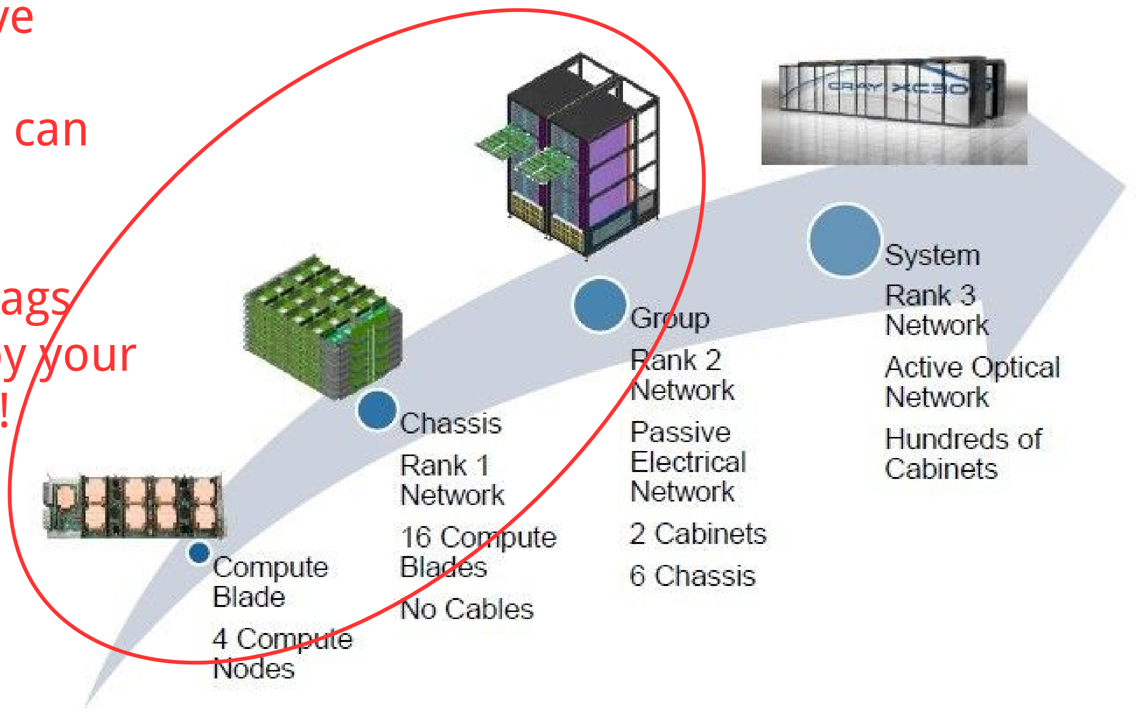


Piz Daint:
Xeon E5-2670 2.6 GHz (SNB)
Tesla K20X
(w/o application clock support)

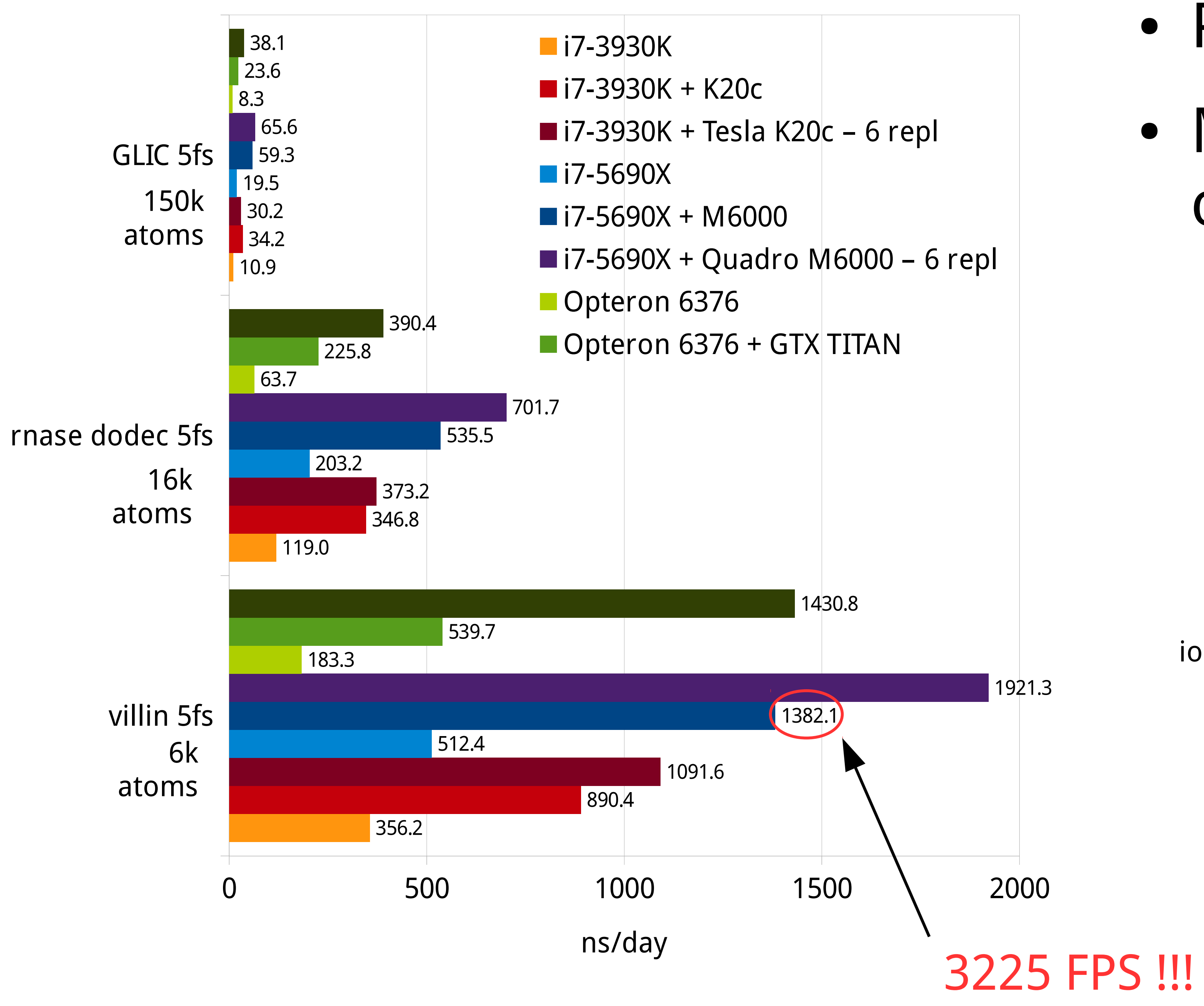
- Cray Aries challenges: up to 2-4x performance fluctuation
  - Network/routing is bandwidth optimized
  - Need to use feature flags
  - Increase GNI eager buffer size

Latency sensitive codes:
stay here if you can

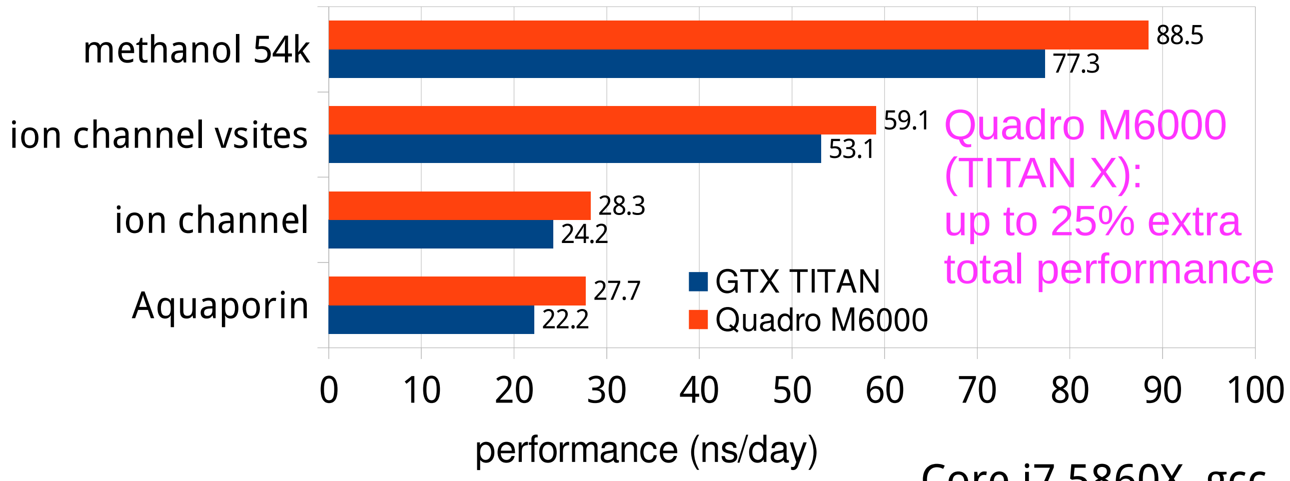
Using SLURM?
→ get feature flags
Implemented by your admins or Cray!



Performance & acceleration: single node

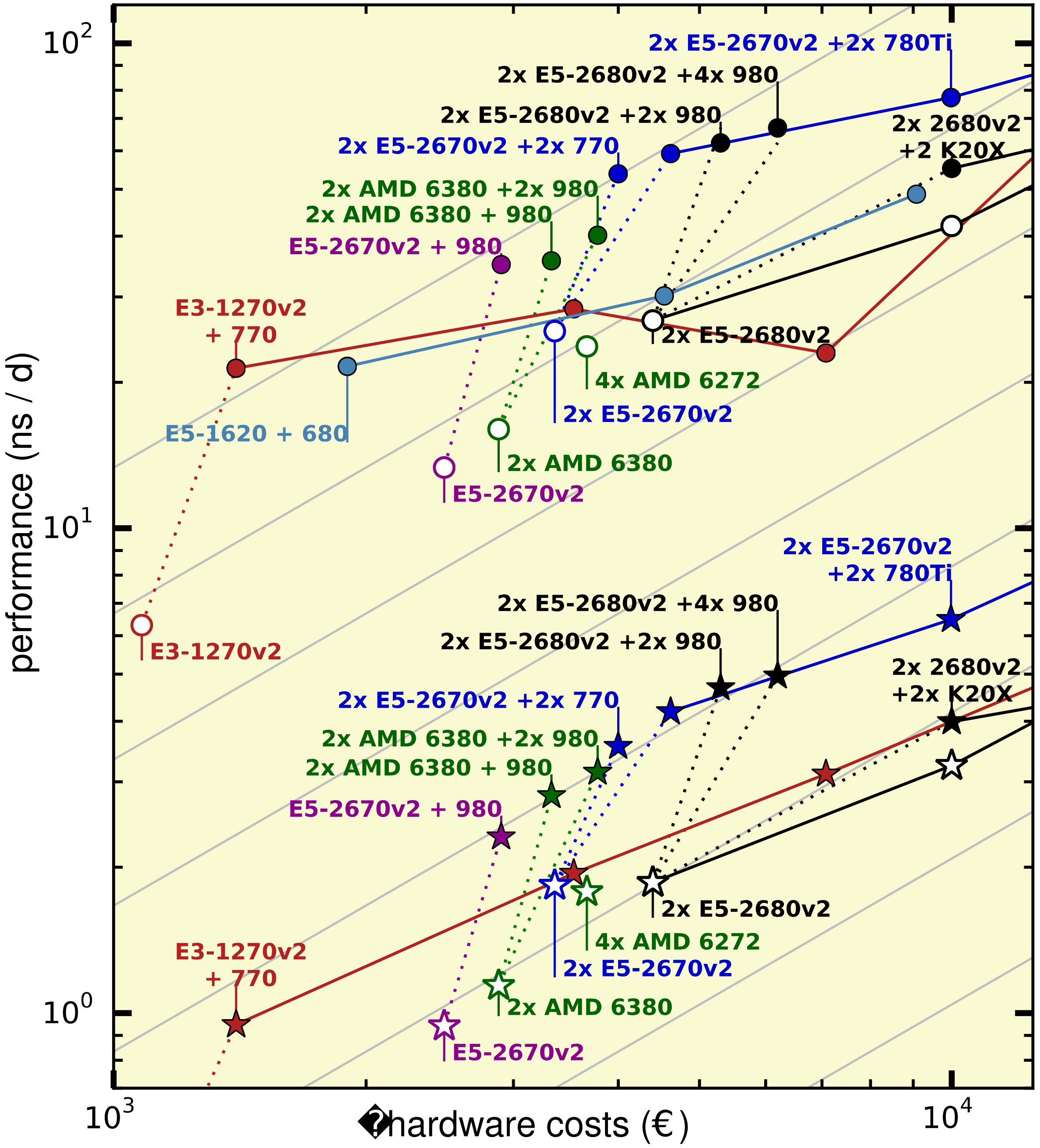


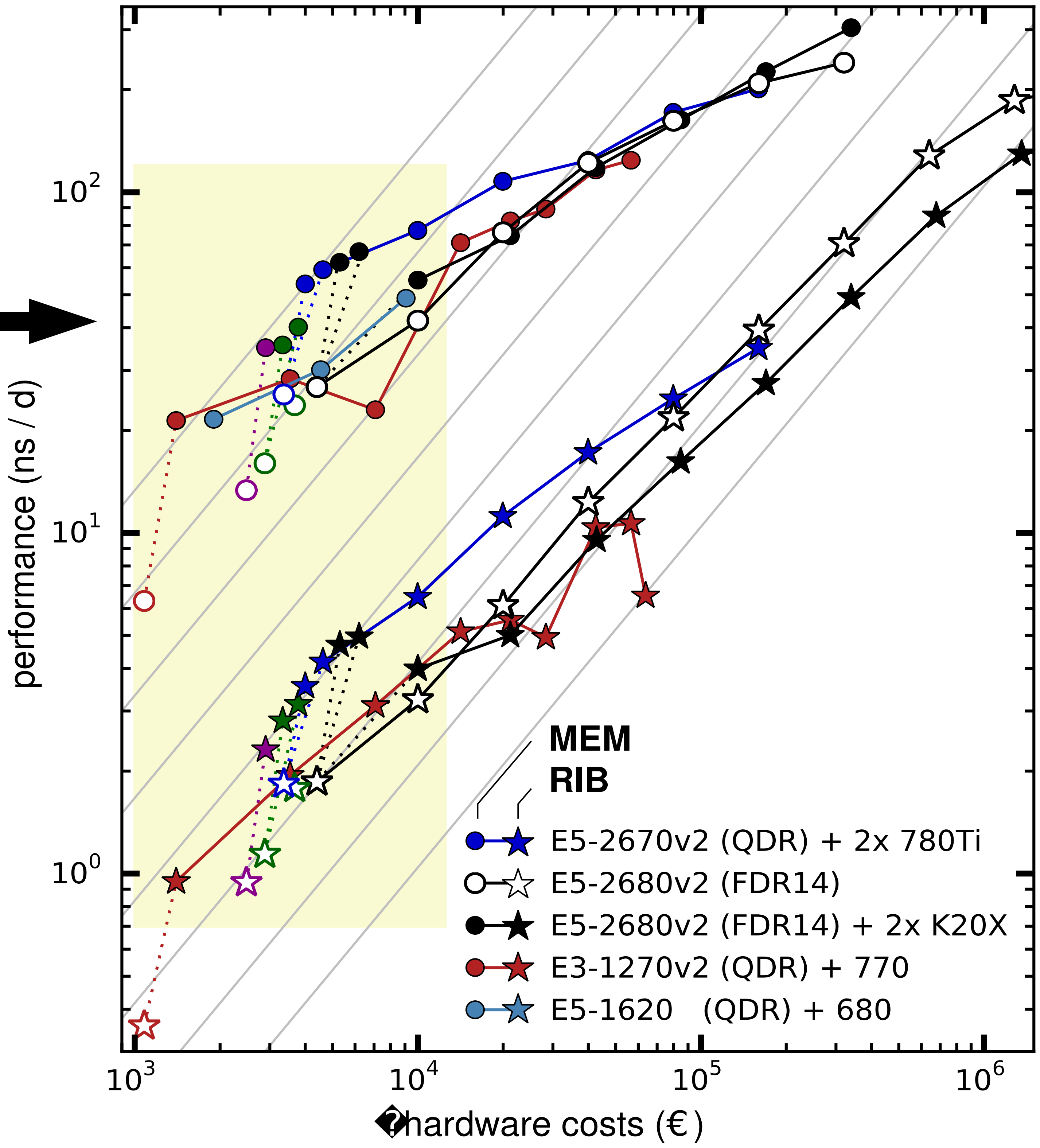
- Peaking at 1.4 /day = 300 us/step
- Multi-simulations make better use of the hardware!
  - Can use it in parallel runs and interleave ranks



Core i7 5860X, gcc 4.9, CUDA 6.5 GROMACS 5.1-dev

Power efficiency





Experiments done by Carsten Kutzner, MPI

Conclusions for MD and not only

- Bottom-up performance engineering:
  - strong benefits (enabling all users)
  - challenges (scaling is less pretty)
- Heterogeneous code:
  - efforts required on all levels (and it will get harder)
  - more offload, more load balancing, more automation needed
- GPUs are a revolution → shifting into evolution mode
- GeForce rules the (MD) research world
- IBM Power8 + CUDA = the opportunity
- ARM64 + CUDA = the promise
  - Where is the big Denver chip?
  - Integration, integration, integration (I hope AMD APUs become a model)

Acknowledgements

Berk Hess, Erik Lindahl

Mark Abraham

- **GMX developers**

    Carsten Kutzner

    Roland Schulz

    The GROMACS

    developers & community

STREAM COMPUTING
Performance Engineers

Vincent Hindriksen
Anca Hamuraru
Teemu Virolainen

- **NVIDIA:**
    - Jiri Kraus, Mark Berger
    - and the many many engineers

VMD rendering: Viveca Lindahl

Hardware / support

NVIDIA

AMD

CSCS
Swiss National Supercomputing Centre

Funding

ScalaLife

SEVENTH FRAMEWORK PROGRAMME

Vetenskapsrådet

erc European Research Council