GPU TECHNOLOGY CONFERENCE

NVIDIA

OPTIMIZING CoMD: A MOLECULAR DYNAMICS PROXY APPLICATION STUDY

Jamal Mohd-Yusof, LANL

Nikolay Sakharnykh, NVIDIA

OUTLINE

- Background
- Molecular dynamics
- Computational considerations
  - Machine level
  - Node level
- CoMD proxy-app

BACKGROUND

- Lessons learned from previous cutting-edge supercomputer systems
  - Programmability, resilience, etc.
- Exascale co-design projects
  - Materials, combustion, reactor design, etc.
- Enable vendors to understand the various domain science workloads while meeting exascale requirements
  - Usable flops
  - Severe power constraints
- Amenable to running on hardware emulators as well as existing hardware
  - Enable co-design of hardware

BACKGROUND

- CoMD is one of several proxy-apps developed as part of the ExMatEx project (exmatex.org)
  - 'Representative' workloads
- One of the proxy-apps included in the Mantevo suite, which won an R&D 100 award in 2013
  - Co-developers Jim Belak, Tim Germann,, Sue Mniszewski, Dave Richards, Chris Sewell, Sriram Swaminarayan
- Collaboration across many institutions and with many vendors, including Nvidia

ExMatEx

MANTEVO

Los Alamos NATIONAL LABORATORY EST.1943

Lawrence Livermore National Laboratory

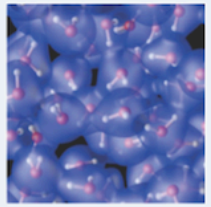
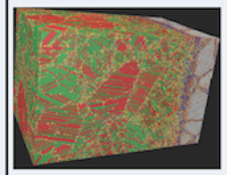
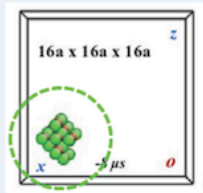
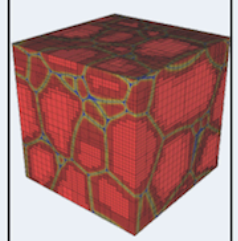
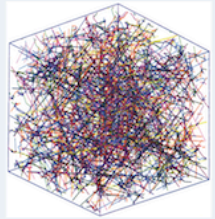
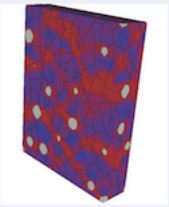
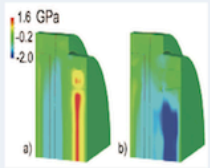
OAK RIDGE National Laboratory

Sandia National Laboratories

STANFORD UNIVERSITY

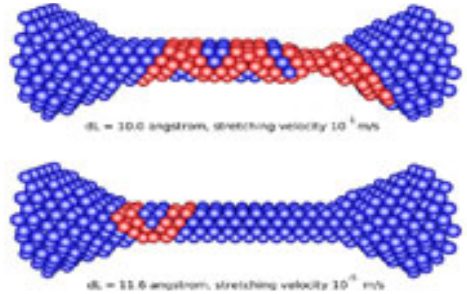
CALIFORNIA INSTITUTE OF TECHNOLOGY 1891

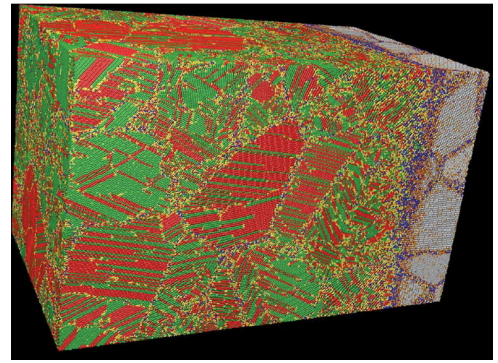
MATERIALS SCIENCE SIMULATION

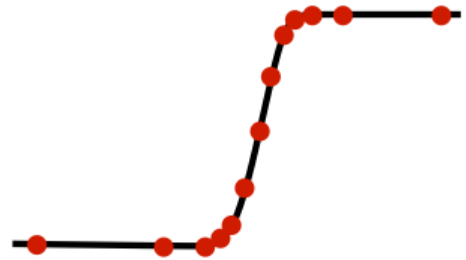
Ab-initio	MD	Long-time	Phase Field	Dislocation	Crystal	Continuum
Inter-atomic forces, EOS	Defects and interfaces, nucleation	Defects and defect structures	Meso-scale multi-phase evolution	Meso-scale strength	Meso-scale material response	Macro-scale material response
						
Code: Qbox/ LATTE Motif: Particles and wavefunctions, plane wave DFT, ScaLAPACK, BLACS, and custom parallel 3D FFTs Prog. Model: MPI + CUBLAS/CUDA	Code: SPaSM/ ddcMD/CoMD Motif: Particles, explicit time integration, neighbor and linked lists, dynamic load balancing, parity error recovery, and *in situ* visualization Prog. Model: MPI + Threads	Code: SEAKMC Motif: Particles and defects, explicit time integration, neighbor and linked lists, and *in situ* visualization Prog. Model: MPI + Threads	Code: AMPE/GL Motif: Regular and adaptive grids, implicit time integration, real-space and spectral methods, complex order parameter Prog. Model: MPI	Code: ParaDis Motif: "segments" Regular mesh, implicit time integration, fast multipole method Prog. Model: MPI	Code: VP-FFT Motif: Regular grids, tensor arithmatic, meshless image processing, implicit time integration, 3D FFTs. Prog. Model: MPI + Threads	Code: ALE3D/ LULESH Motif: Regular and irregular grids, explicit and implicit time integration. Prog. Model: MPI + Threads

USAGE SCENARIO EXAMPLES

- ‘MD-everywhere’
  - Large MD simulations running across many nodes
    - Scale effects important
- ‘MD-on-demand’
  - Smaller simulations launched in response to macro- or meso-scale simulations
    - Could still require multiple nodes
    - Must communicate constitutive values with main simulation
      - E.g. strain → stress
    - Adaptive sampling strategies

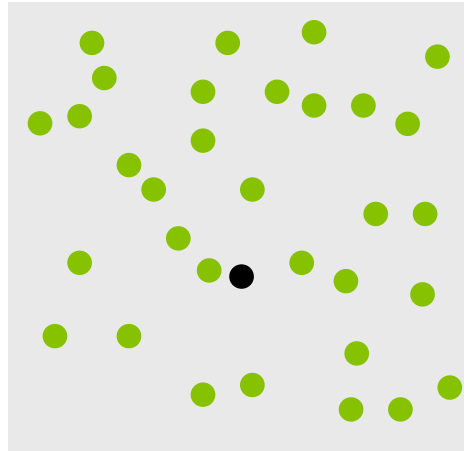


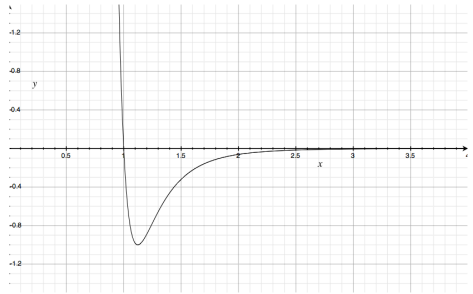




MOLECULAR DYNAMICS

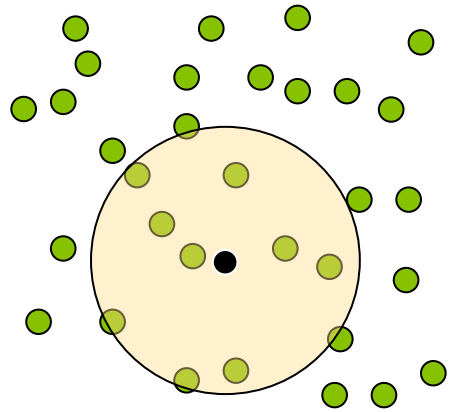
- Atomic-scale simulations of materials solves Newton's laws between particles
  - F(r) = ma = - grad(u)
  - u is some unspecified (so far) potential
- Each particle interacts with all other particles
  - N^2 problem unless we impose some constraints

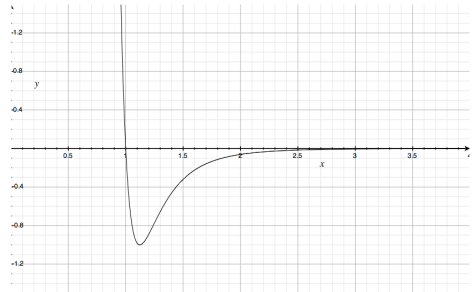




MOLECULAR DYNAMICS

- Atomic-scale simulations of materials solves Newton's laws between particles
  - F(r) = ma = - grad(u)
  - u is some unspecified (so far) potential
- Each particle interacts with all other particles
  - N^2 problem unless we impose some constraints
- For our problems
  - Targeted at materials in extreme environments
  - Short range potentials
    - Impose a finite cutoff radius
    - Appropriate for metals





MOLECULAR DYNAMICS

- Two common potentials
  - Lennard-Jones (LJ)
  - General physics

$$E_{sys} = 4\varepsilon \frac{\sigma^6}{r_{ij}^6} \sum_i \left(\frac{\sigma^6}{r_{ij}^6} - 1 \right)$$

$$F_{ij} = 4\varepsilon \frac{\sigma^6}{r_{ij}^7} \left(12 \frac{\sigma^6}{r_{ij}^6} - 6 \right)$$

  - Embedded Atom Method (EAM)
  - Fitted to specific metals

$$E_{sys} = \frac{1}{2} \sum_i \sum_{k \neq i} \phi(r_{ik}) + \sum_i F(\bar{\rho}_i) \; where \; \bar{\rho}_i = \sum_{k \neq i} \rho(r_{ik})$$

$$F_{ij} = \frac{\partial E_{sys}}{\partial r_{ij}} = \frac{\partial \phi_{ij}}{\partial r_{ij}} + \frac{\partial \rho}{\partial r_{ij}} \left(\left. \frac{\partial F}{\partial \bar{\rho}} \right|_{\bar{\rho}_i} + \left. \frac{\partial F}{\partial \bar{\rho}} \right|_{\bar{\rho}_j} \right)$$

MOLECULAR DYNAMICS

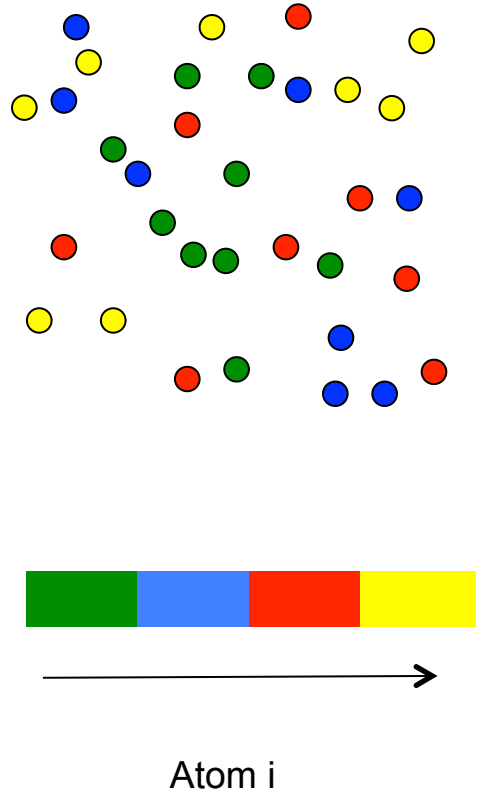
- Time-step outline
  - Compute forces between particles (~90%)
    - Variety of potentials and implementations
  - Update velocities (~5%)
  - Update positions (~5%)
  - Re-sort particle data
    - depends on implementation details
  - Analysis
    - Stress, strain, crystal structure, etc.
    - Problem dependent

COMPUTATIONAL CONSIDERATIONS

- At the machine level
  - Work decomposition
  - Communication patterns
  - I/O
- At the node level
  - Data decomposition
  - Data layout
  - Potential evaluation
  - Diagnostics

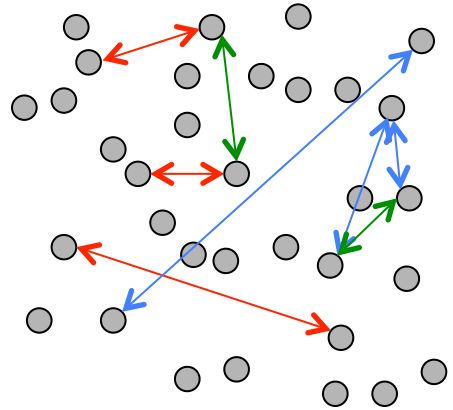
MACHINE-LEVEL CONSIDERATIONS

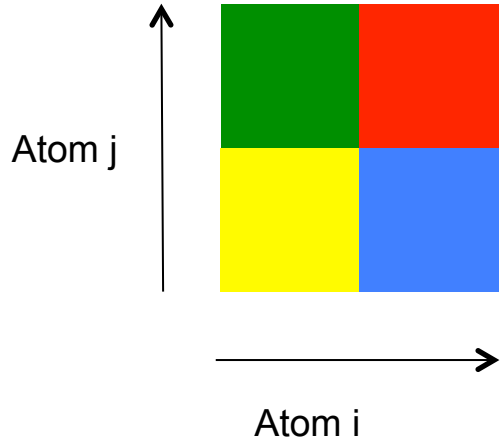
- Work decomposition
  - Atom decomposition (AD)
    - Atoms assigned to processor, regardless of spatial position



MACHINE-LEVEL CONSIDERATIONS

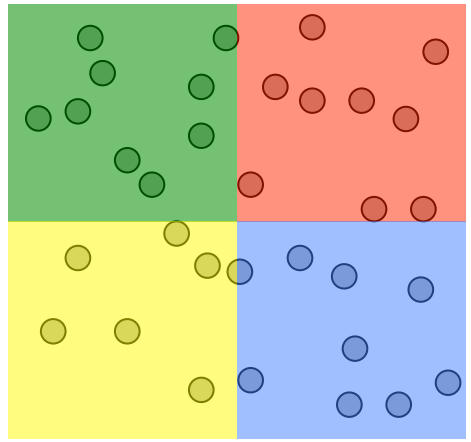
- Work decomposition
  - Atom decomposition (AD)
    - Atoms assigned to processor, regardless of spatial position
  - Force decomposition (FD)
    - Processor assigned subset of *ij* particle pairs to compute forces





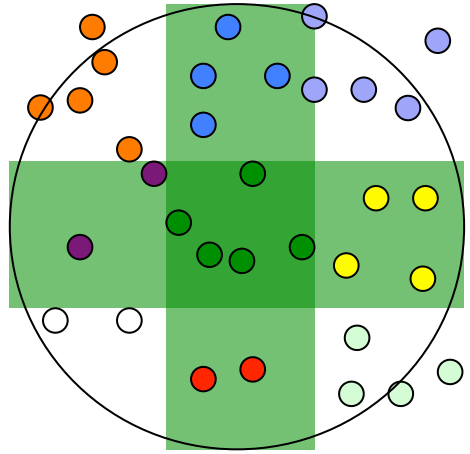
MACHINE-LEVEL CONSIDERATIONS

- Work decomposition
  - Atom decomposition (AD)
    - Atoms assigned to processor, regardless of spatial position
  - Force decomposition (FD)
    - Processor assigned subset of *ij* particle pairs to compute forces
  - Spatial decomposition (SD)
    - Processor assigned particles in some region of space



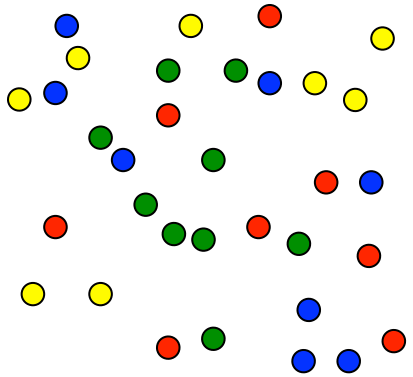
MACHINE-LEVEL CONSIDERATIONS

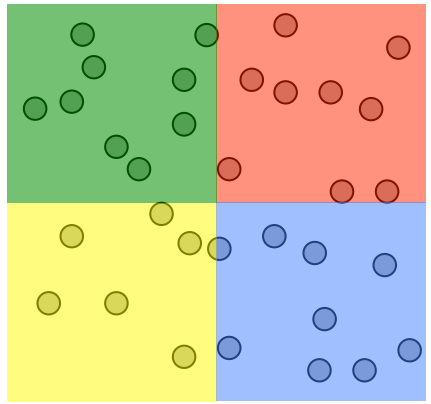
- Work decomposition
  - Atom decomposition (AD)
    - Atoms assigned to processor, regardless of spatial position
  - Force decomposition (FD)
    - Processor assigned subset of *ij* particle pairs to compute forces
  - Spatial decomposition (SD)
    - Processor assigned particles in some region of space
  - Neutral territory (NT)
    - Processor computes force for particles residing in its row/column
    - Cutoff radius > box size



WORK DECOMPOSITION METHODS

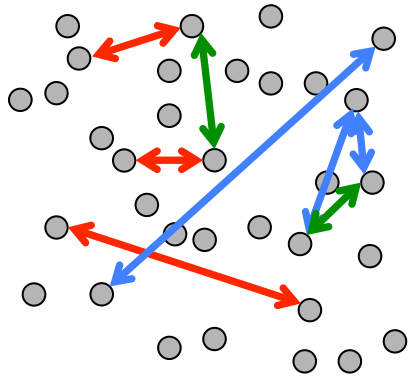
Assign Atoms to Nodes

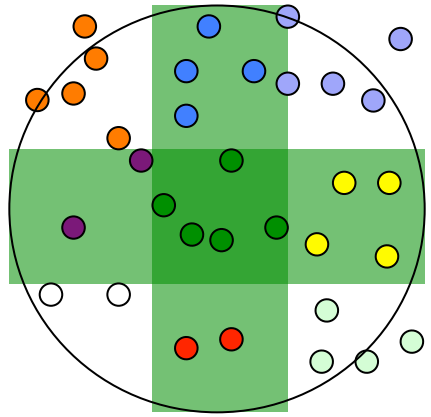




Assign Spatial Regions to Nodes

Assign Force Terms to Nodes

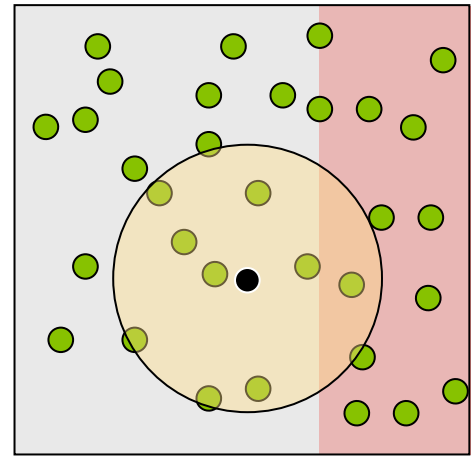




Neutral Territory Method

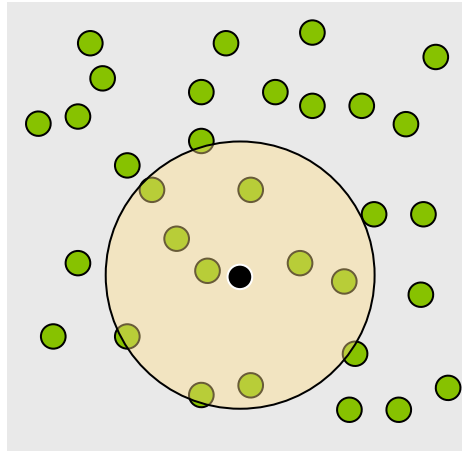
MACHINE-LEVEL CONSIDERATIONS

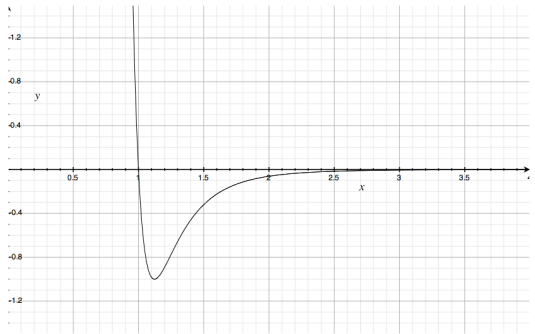
- Spatial decomposition is most often used for materials science MD applications, and is the focus (thus far) in CoMD
- Communication
  - Particles on adjacent node needed to compute force on local particle
    - Requires replication of particle data in halo region (PGAS?)
    - Data needed depends on potential used
    - For EAM, partial forces communicated during the force computation step



NODE-LEVEL CONSIDERATIONS

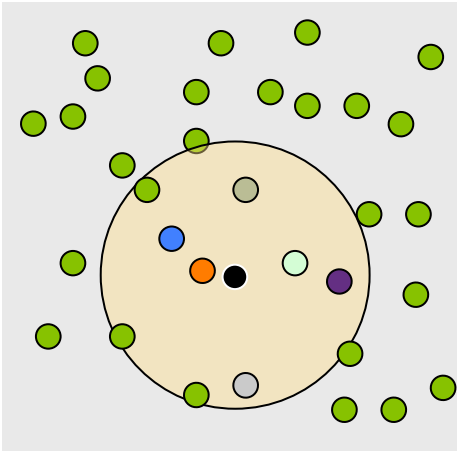
- For a finite cutoff radius potential:
  - How do we organize neighbors
    - Neighbor lists
    - Link cells
  - How do we compute potentials
    - Tabulated
    - Parameterized
- Optimal choice is architecture-dependent
  - Data locality
  - Vectorizability
  - Thread safety
  - Cache use

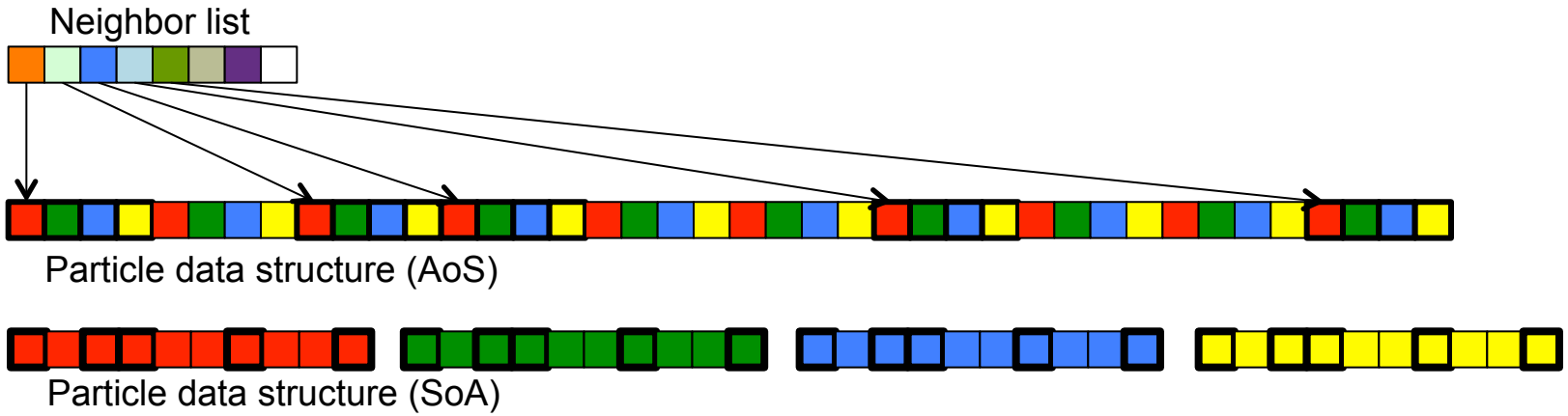




NODE-LEVEL CONSIDERATIONS

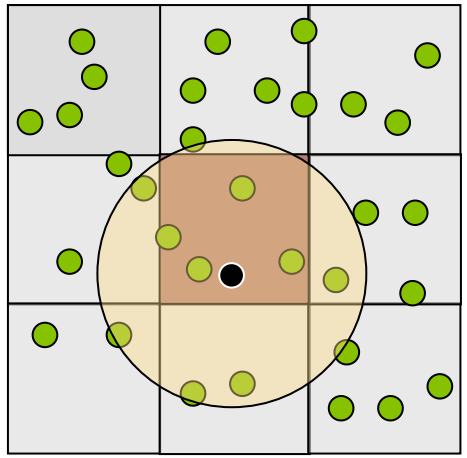
- Neighbor lists
  - Each particle has a pointer list to nearby particles
  - Pointer chasing
  - Poor cache performance
    - Mitigated with:
      - Clever sorting
      - AoS data structures

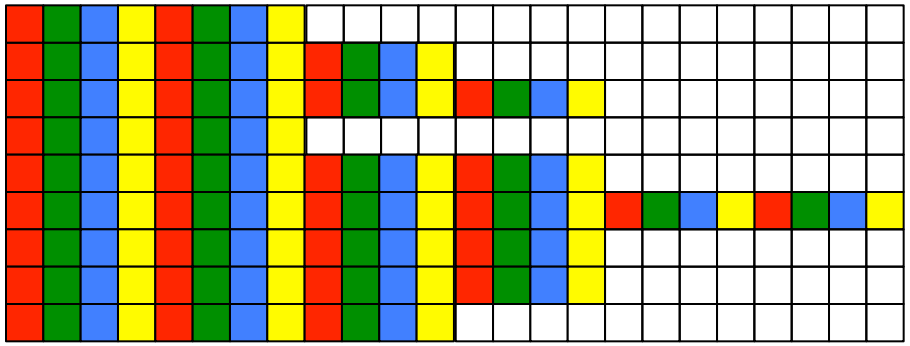


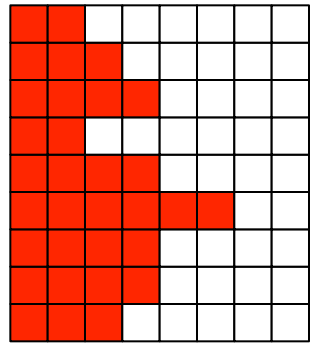


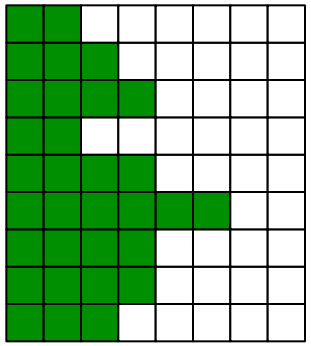
NODE-LEVEL CONSIDERATIONS

- Link cells
  - Local domain is decomposed into cells
    - Can be either Cartesian (as shown) or otherwise
    - Cell size (d) is larger than cutoff radius (r_c)
  - Data layout is improved
    - Nearby particles adjacent in memory
    - Either AoS or SoA









BASE PROXY-APP (CoMD)

- Based on SPaSM (loosely)
- Array-of-Structures (AoS) data layout e.g.
  - r = {x,y,z}
  - f = {f_x, f_y, f_z, e}
- Serial loops;

```
Loop over all cells
    Loop over all neighbor cells
        Loop over all particles i in cell
            Loop over all particles j in neighbor cell
```

  - Because loop is serialized, can increment forces, energy on both i , j to save redundant computation

PARALLEL FORCE COMPUTATION

- For EAM potential
  - Need 2 separate kernels
    - accumulate $\bar{\rho}_i = \sum_{k \neq i} \rho(r_{ik})$, store in temporary array
    - compute $\left.\frac{\partial F}{\partial \bar{\rho}}\right|_{\bar{\rho}_i}$ for each i, store in temporary array
    - (communicate partial forces)
    - compute total E_i, F_i
  - EAM tables are large (500-10,000 entries)

TIMESTEPPING

- Velocity-Verlet timestepping
  - Velocity and position are offset by half-timestep
    - Stored at full timesteps in restart file
- From a restart
  - Advance position by dt/2
  - Timestep iteration
    - Compute force (only a function of position)
    - Advance velocity by dt
    - Advance position by dt
    - Resort particles
  - Rewind position by dt/2 before saving a restart

POSITION/VELOCITY UPDATE

- Due to staggered time levels, centered in time for both velocity and position update
  - Position

$$r_{t+1} = r_t + v_{t+1/2}dt$$

  - Velocity

$$v_{t+1/2} = v_{t-1/2} + a_t dt$$

$$a_t = \frac{f_t}{m}$$

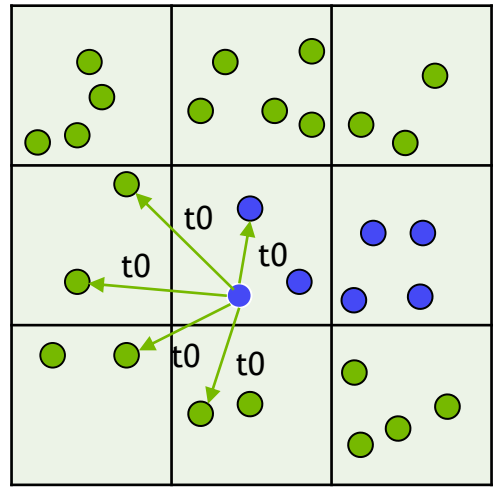
  - 1 thread per particle for both AoS and SoA OpenCL implementations

CoMD GPU ANALYSIS

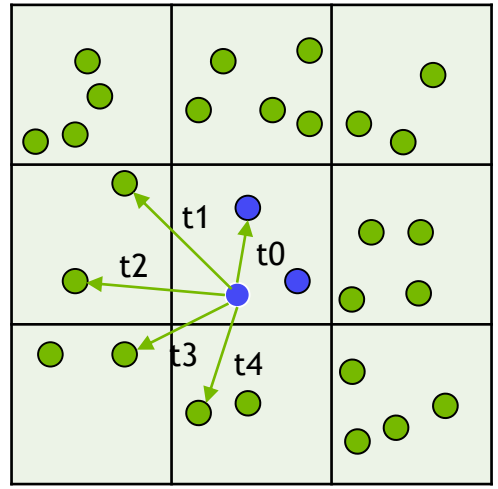
- Node-level implementations
  - Cell-based variants
  - Neighbor-lists variants
- System-scale analysis
  - Distributed implementation details
- Used in NVIDIA Research studies of future architectures
  - Explore various hardware/software trade-offs

CELL-BASED IMPLEMENTATIONS

- Thread / Atom
  - Each thread computes forces with all atoms in 3x3x3 neighbor cells
- Block / Cell
  - Each thread computes forces with a few atoms in 3x3x3 neighbor cells
  - Threads collaborate by using shared memory

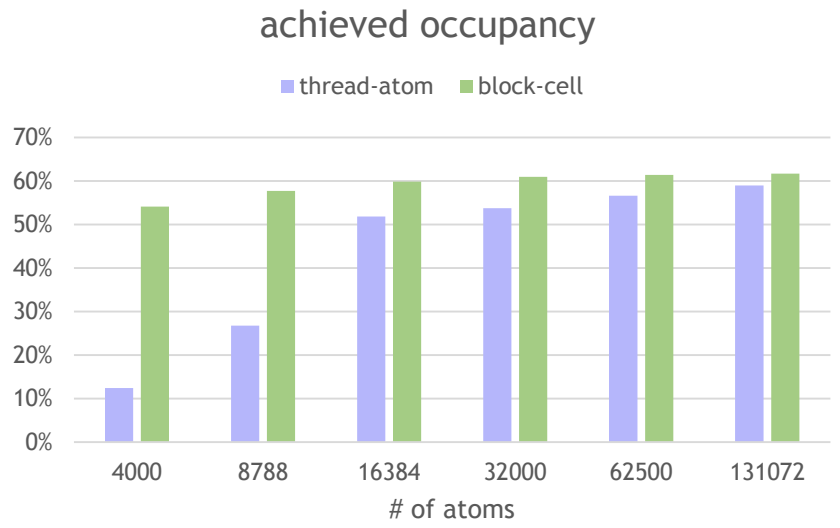


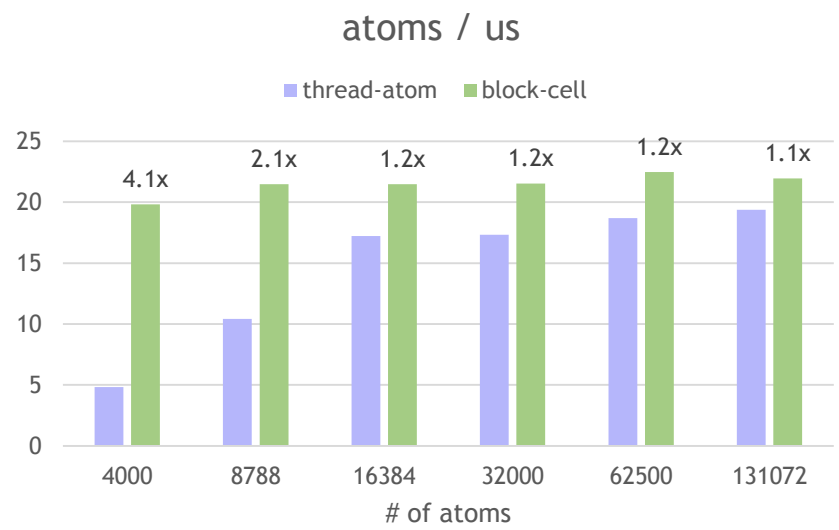
warp/block spans across multiple cells



warp/block assigned to a single cell

CELL-BASED COMPARISON

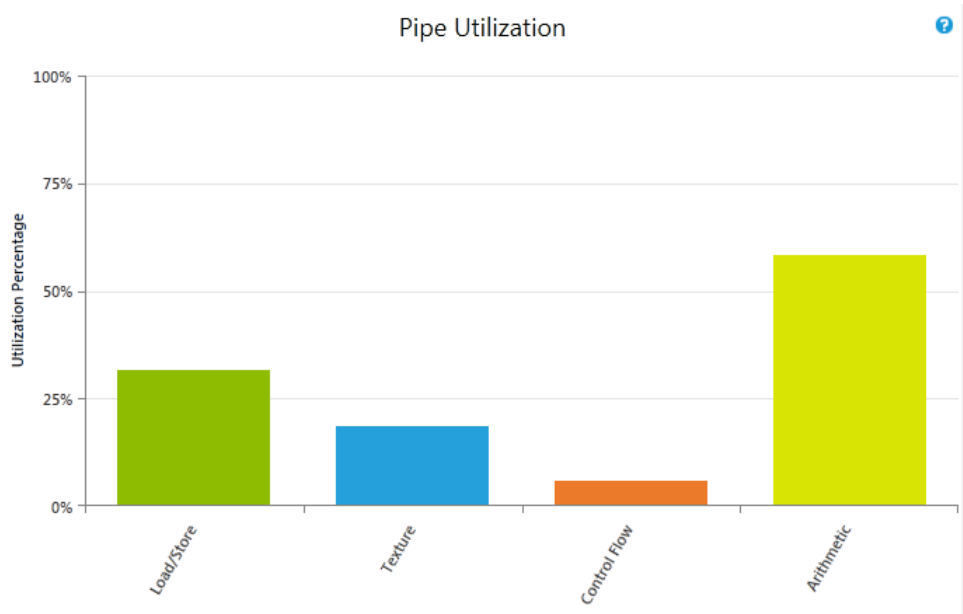




- Block-cell performs better on small grids
  - Beneficial for strong scaling
  - Better choice for expensive potentials

CELL-BASED ANALYSIS

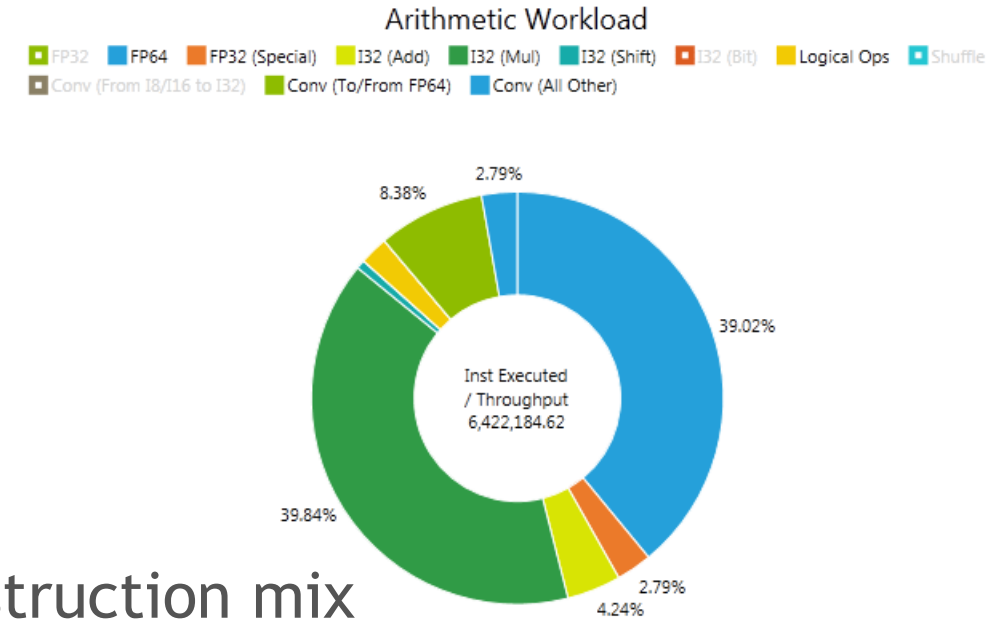
- Memory utilization
  - Low DRAM bandwidth utilization: 4%
  - High L2 cache hit rates: 87%
- SM utilization



NVIDIA Nsight
Visual Studio Edition

CELL-BASED ANALYSIS

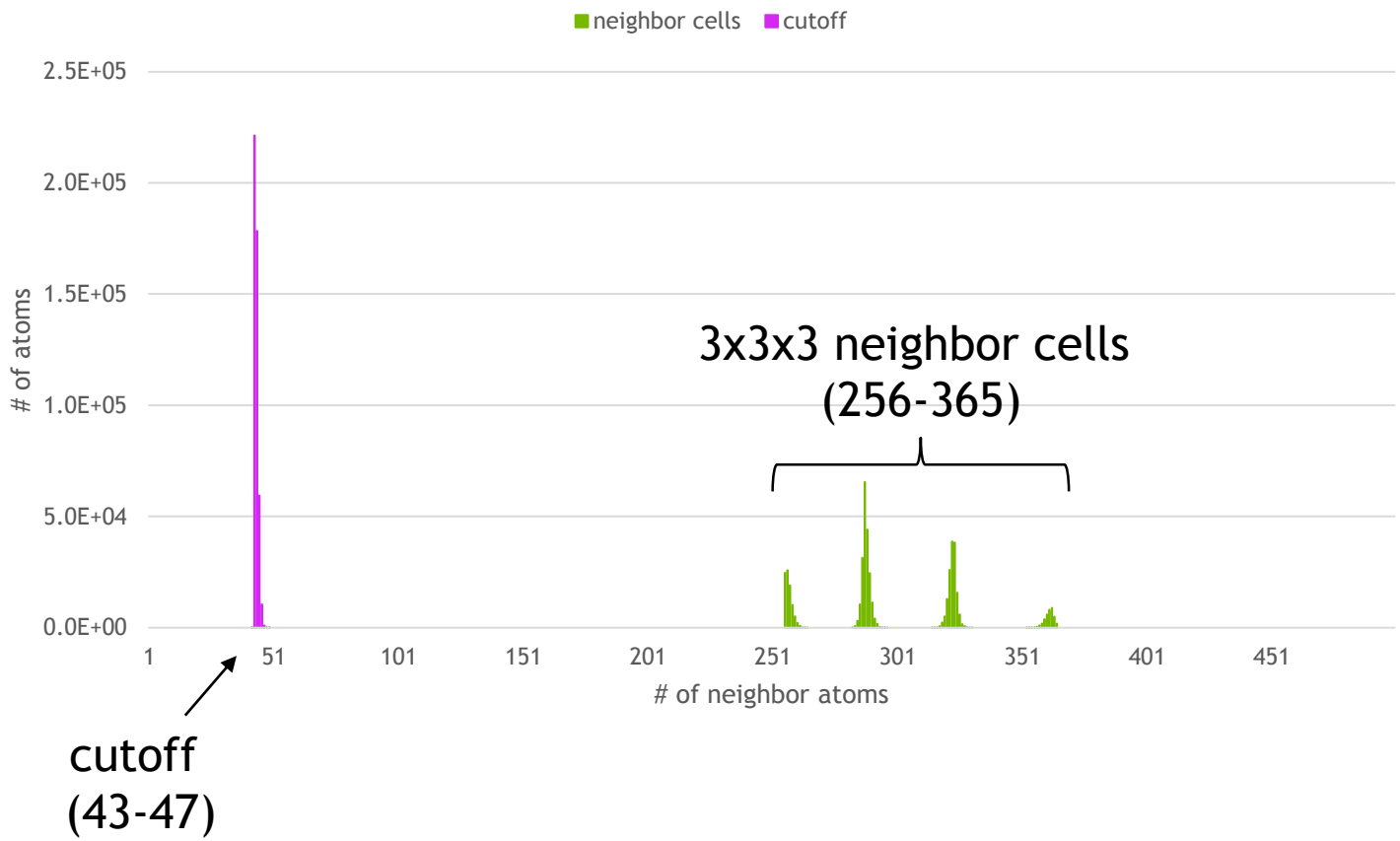
	thread/ atom	block/ cell
Executed IPC	2.22	2.56
Warp efficiency	0.40	0.73



- IPC looks good for a given instruction mix
  - Could be better if we expose more ILP in the code
- Low warp execution efficiency due to cut-off check
  - Narrow warps should improve that

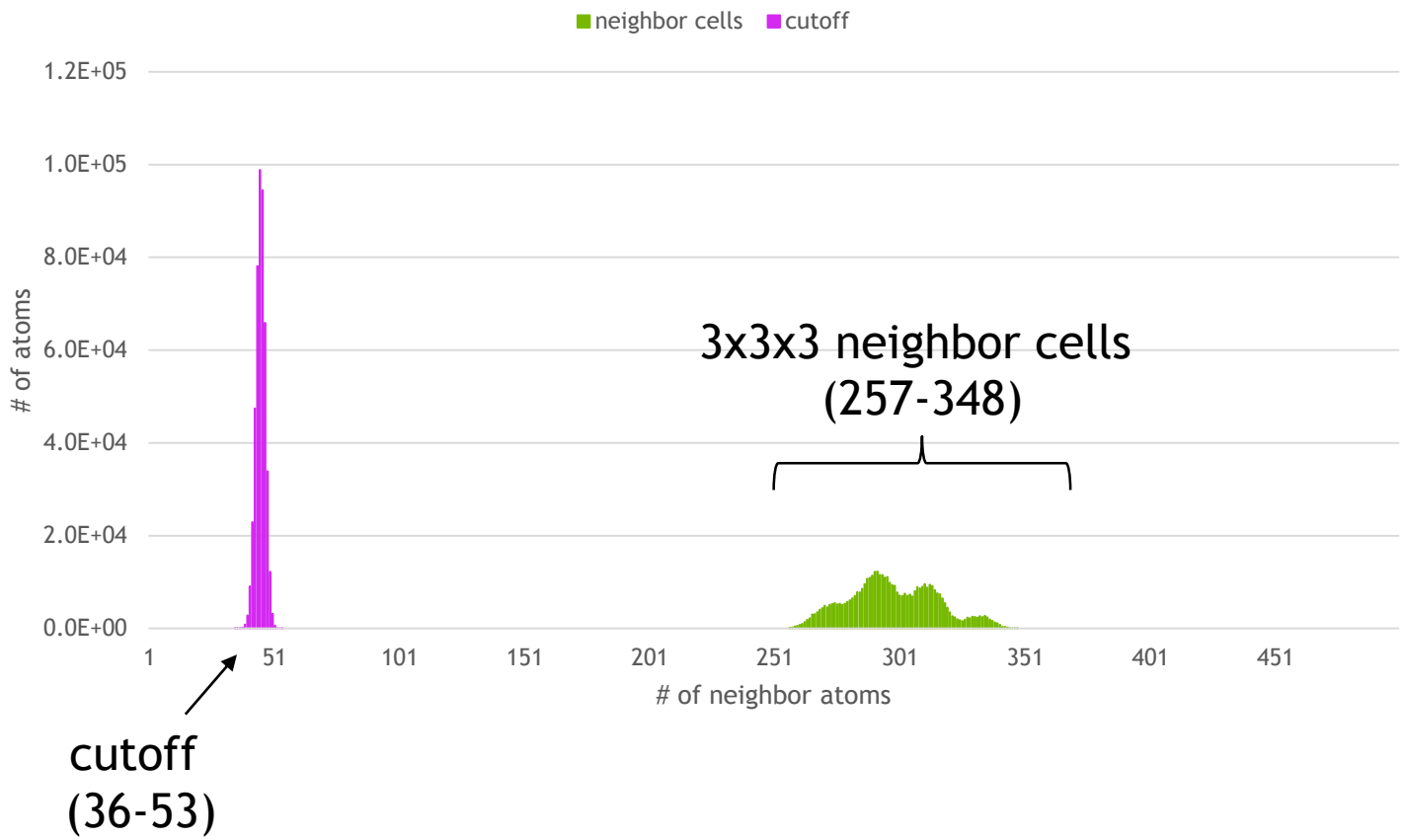
NEIGHBORS SEARCH

- Solid copper (T=273), cutoff = 4.95



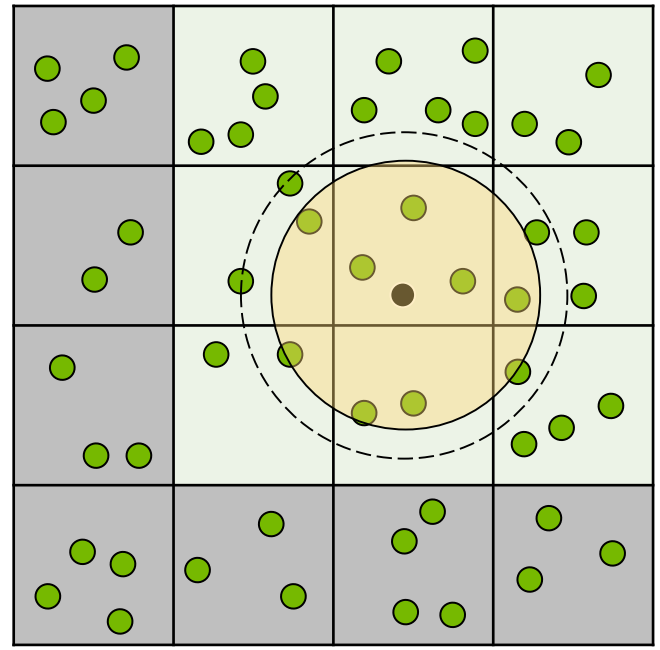
NEIGHBORS SEARCH

- Melting copper (T=1722), cutoff = 4.95



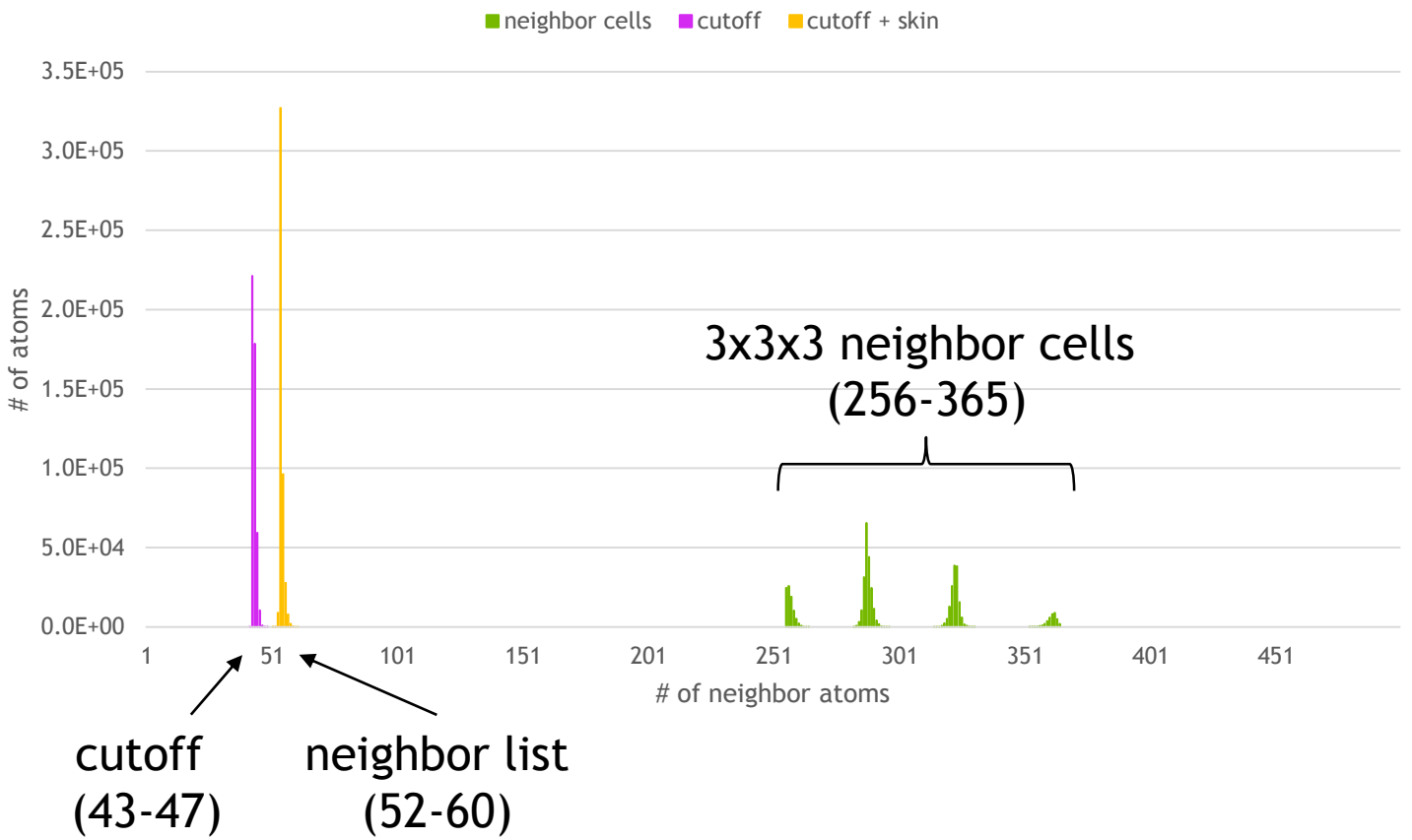
NEIGHBOR LISTS

- Cell-based
  - Low successful checks ≈15%
- Build a list of neighbors per atom
  - Cutoff + skin distance
  - Update every n-th iteration
- Force kernel speed-up **2.2x**



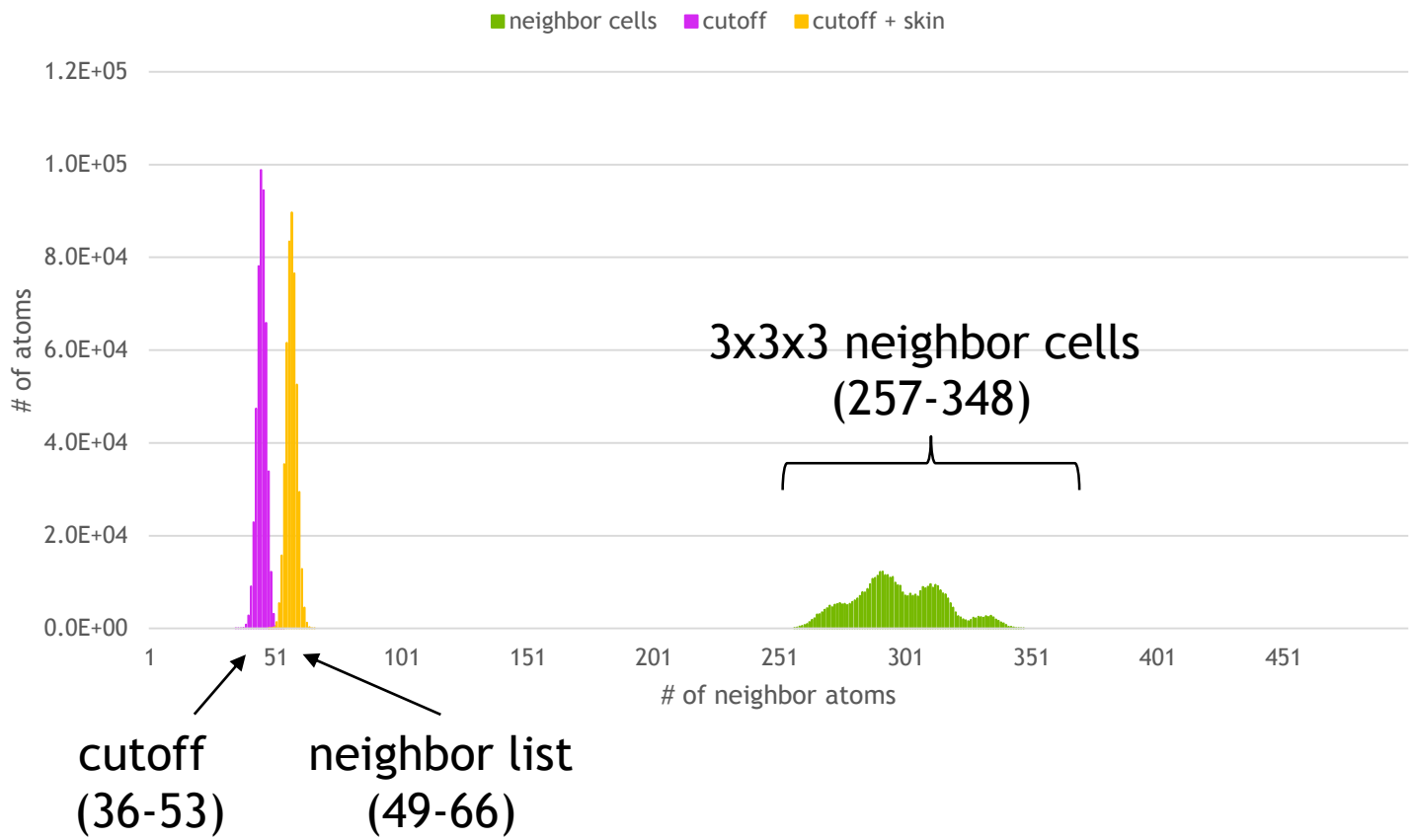
NEIGHBORS SEARCH

- Solid copper (T=273), cutoff = 4.95, skin = 10%



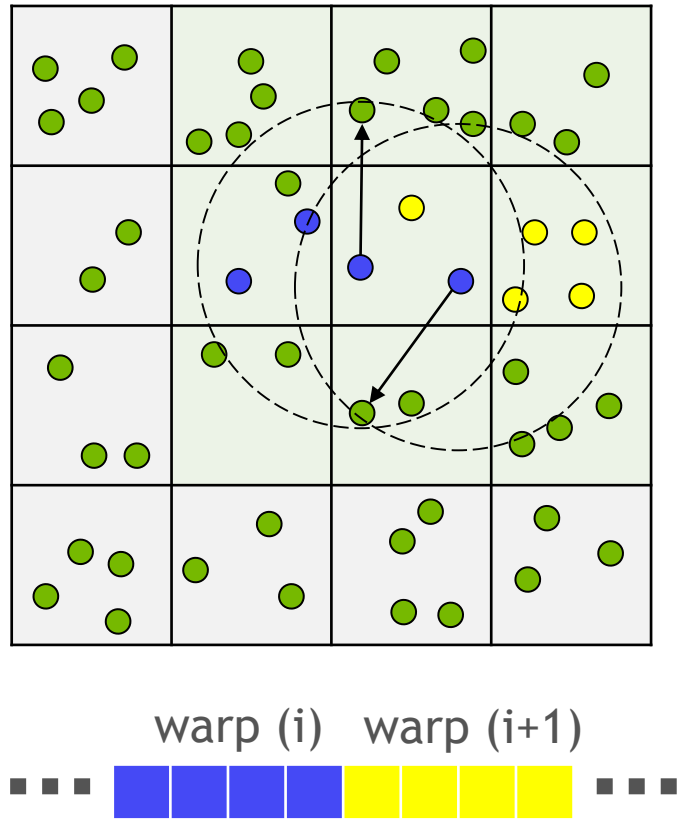
NEIGHBORS SEARCH

- Melting copper (T=1722), cutoff = 4.95, skin = 10%



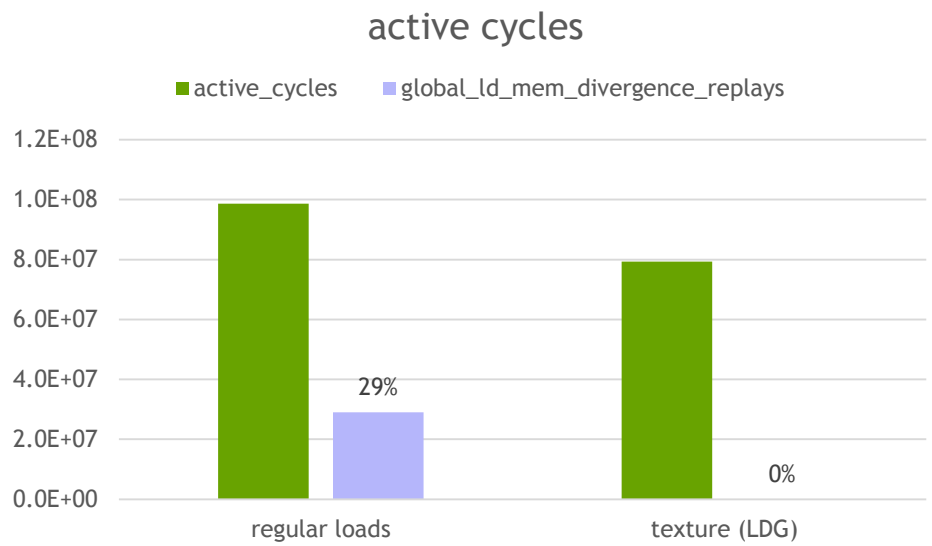
NEIGHBOR LISTS ANALYSIS

- Non-coherent memory accesses
  - POS: 10 transactions/request
  - Replays of global memory loads
- Possible solutions
  - Build neighbor lists per warp
    - Large lists – less efficient search
  - Use texture (read-only) loads



TEXTURE LOADS

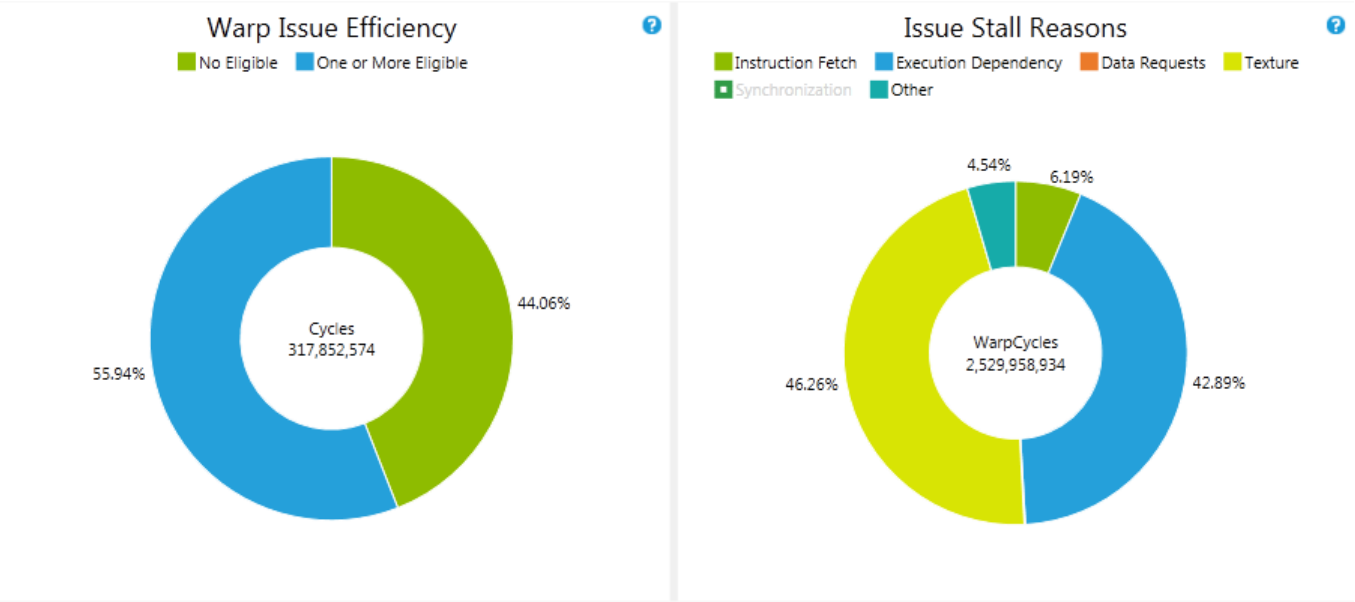
- Use texture loads (LDG) for neighbor positions on Kepler



- Kernel speed-up **1.25x**

NEIGHBOR LISTS ANALYSIS

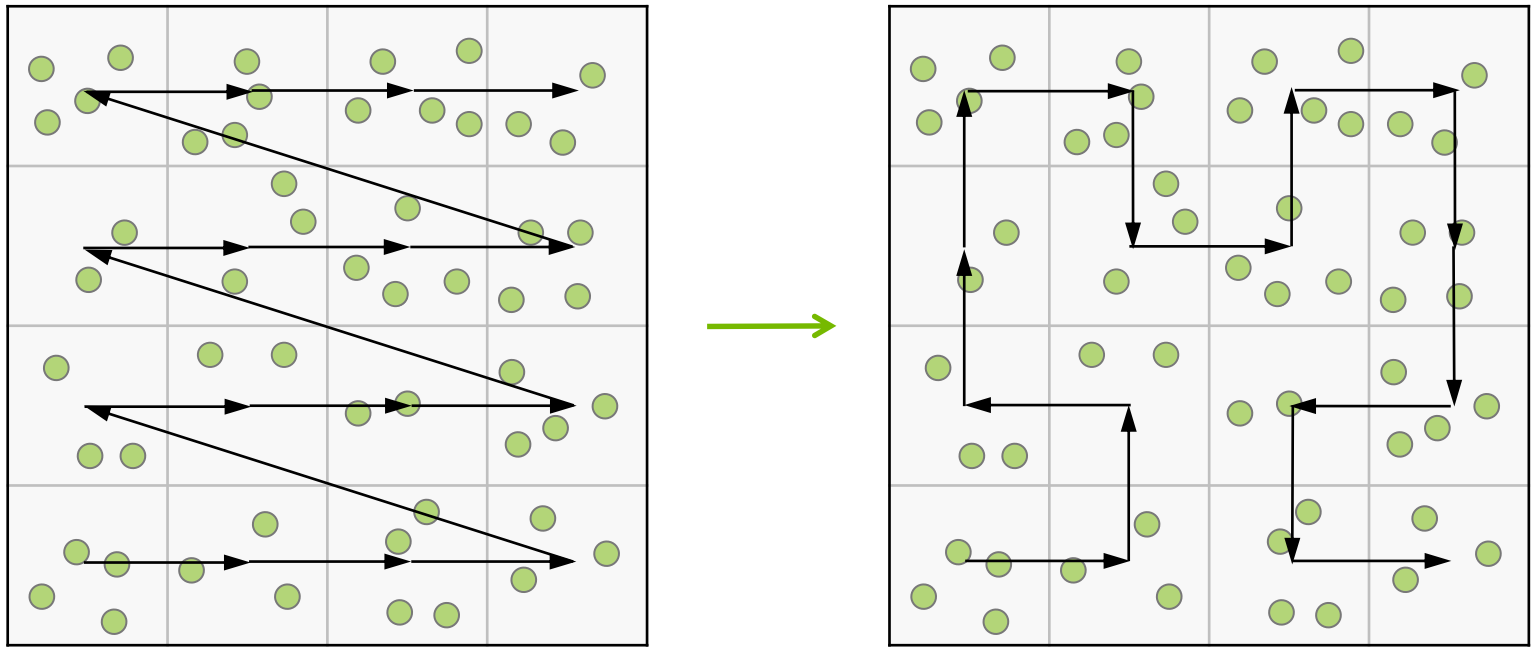
- Issue rate is limited by stalls on texture requests



- Long latency waits
  - Lots of requests missing in L2 (47-65% hit rate)

HILBERT CURVE OPTIMIZATION

- Reorder cells for better spatial locality

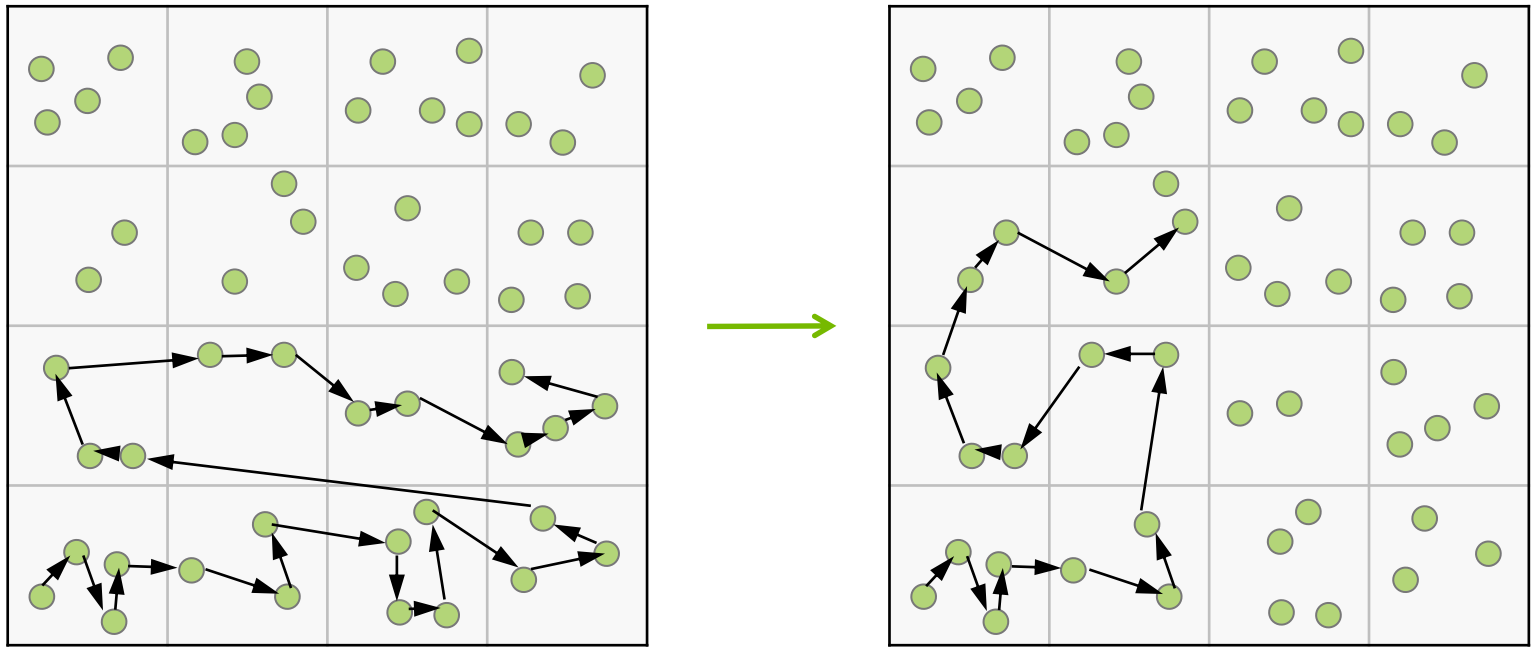


Linear mapping

Hilbert curve

HILBERT CURVE OPTIMIZATION

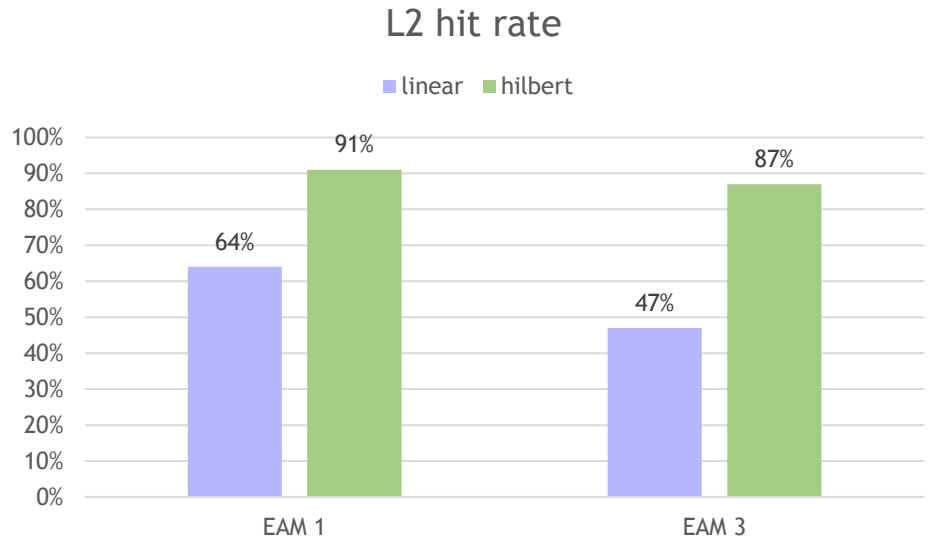
- Reorder cells for better spatial locality



Linear mapping

Hilbert curve

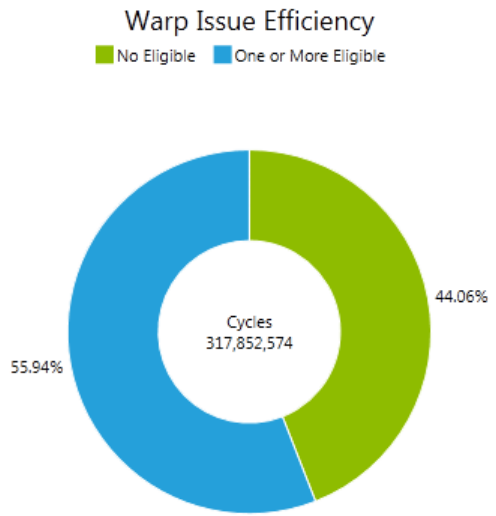
HILBERT CURVE CACHE ANALYSIS



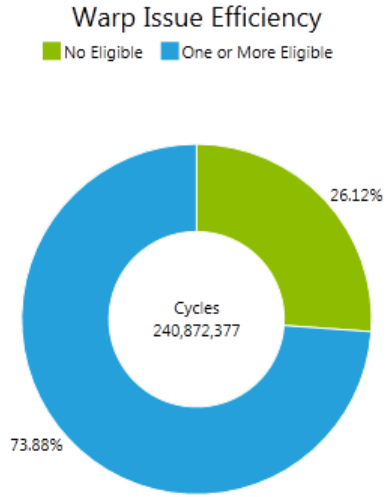
- EAM-3 hit rates increase is even better
  - More L2 requests due to embedded force loads
  - Kernel performance improvements up to **1.9x**

HILBERT CURVE ANALYSIS

- Issue rate is much better
  - IPC increased from 2.49 to 3.27



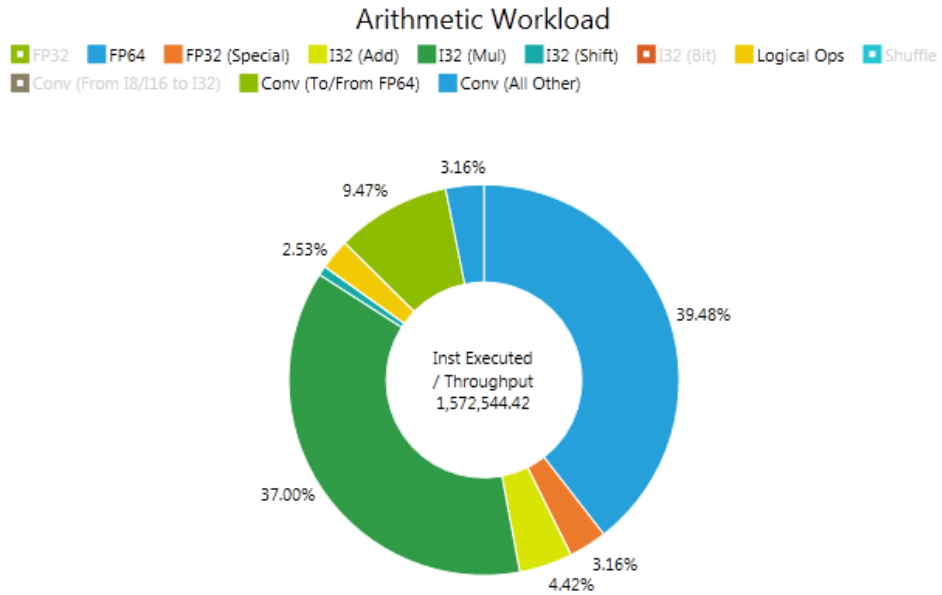
LINEAR MAPPING



HILBERT CURVE

HILBERT CURVE ANALYSIS

	NL	NL + Hilbert
Executed IPC	1.75	2.30



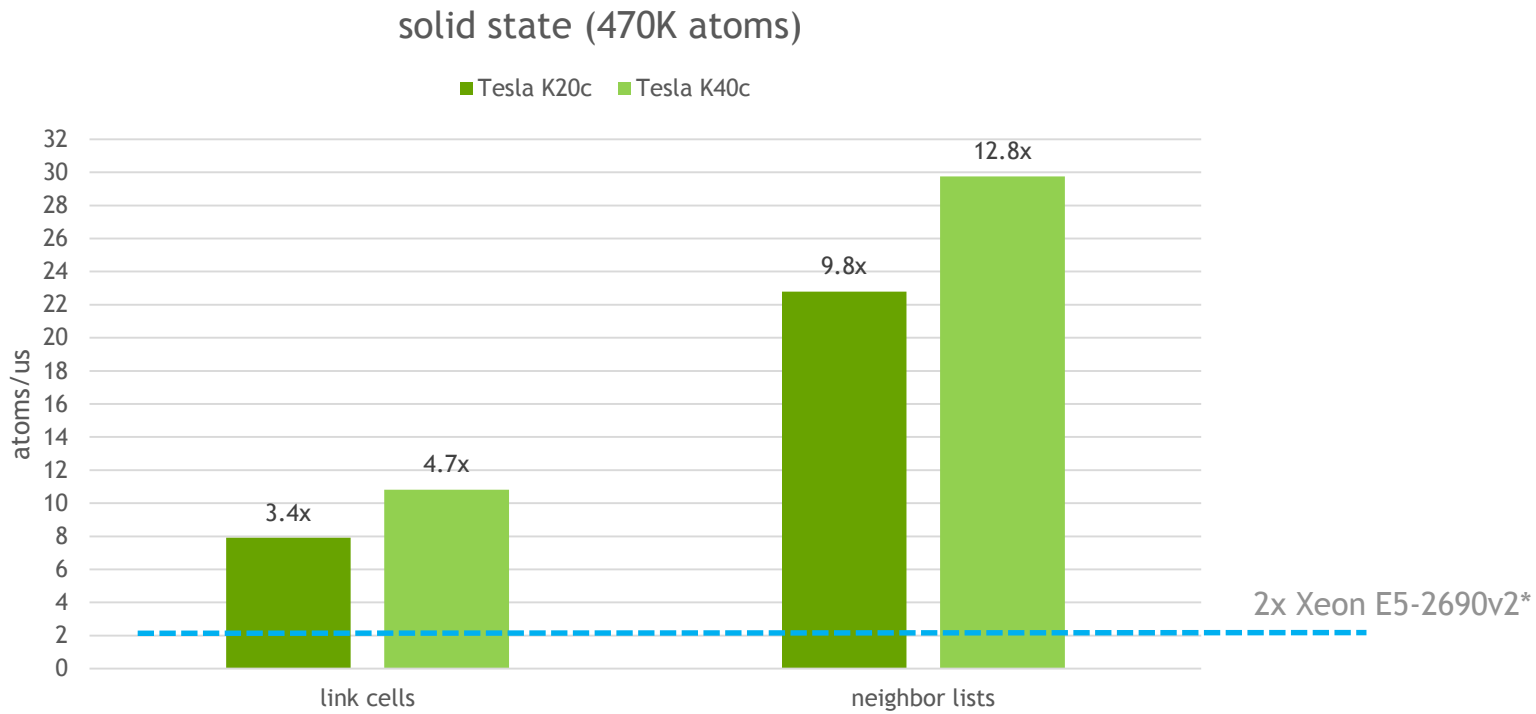
- IPC is OK for the instruction mix
  - Further improvement requires more ILP or better TEX hit rates

SUMMARY OF GPU IMPROVEMENTS

- Tesla K20c, 470K atoms

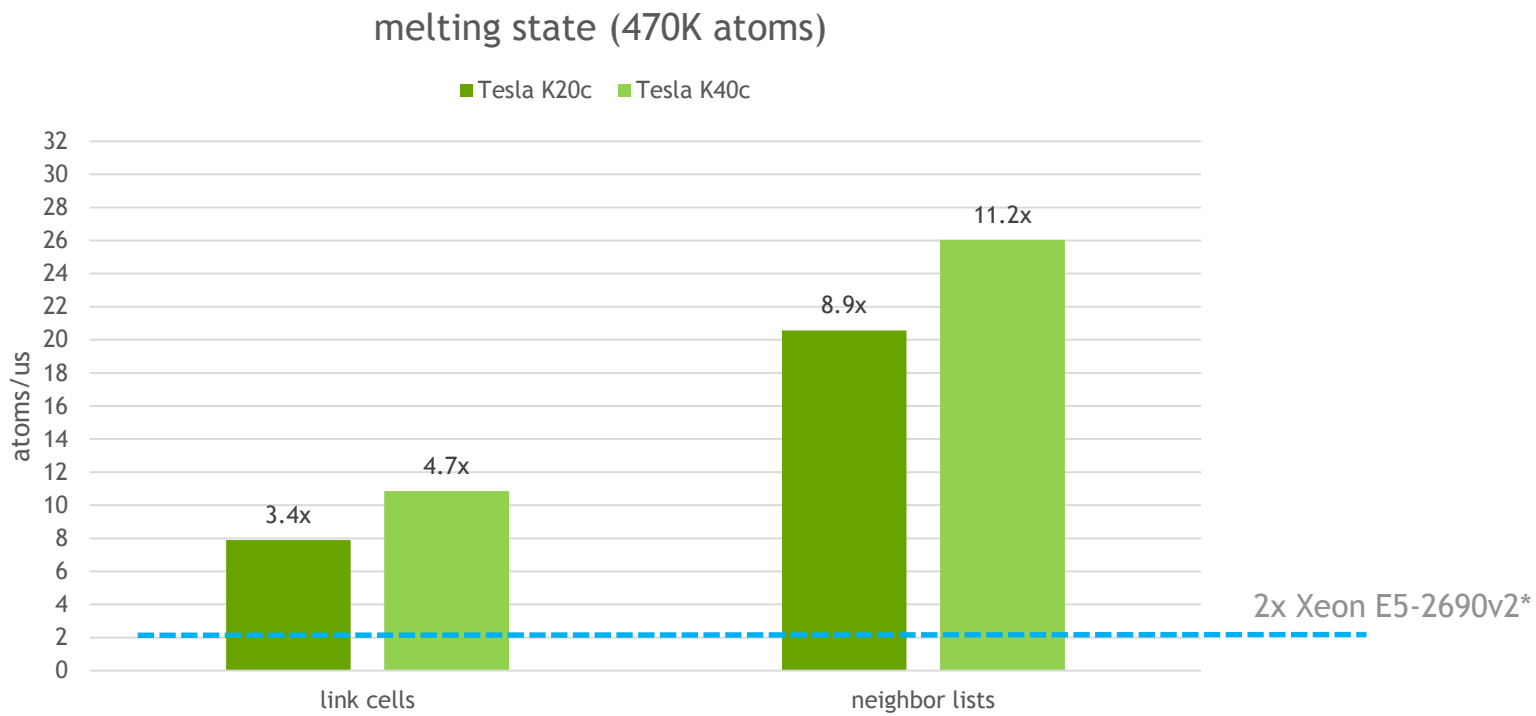
Method	Optimization	EAM_ 1 time (ms)	relative speed-up	EAM_3 time (ms)	relative speed-up
Link cells	thread/atom	26.583		23.547	
	block/cell	24.162	**1.10**	22.043	**1.07**
Neighbor lists	gmem loads	10.803	**2.24**	15.511	**1.42**
	texture loads	8.678	**1.24**	14.643	**1.06**
	hilbert curve	6.958	**1.25**	7.675	**1.91**

SINGLE NODE BENCHMARK, T=600



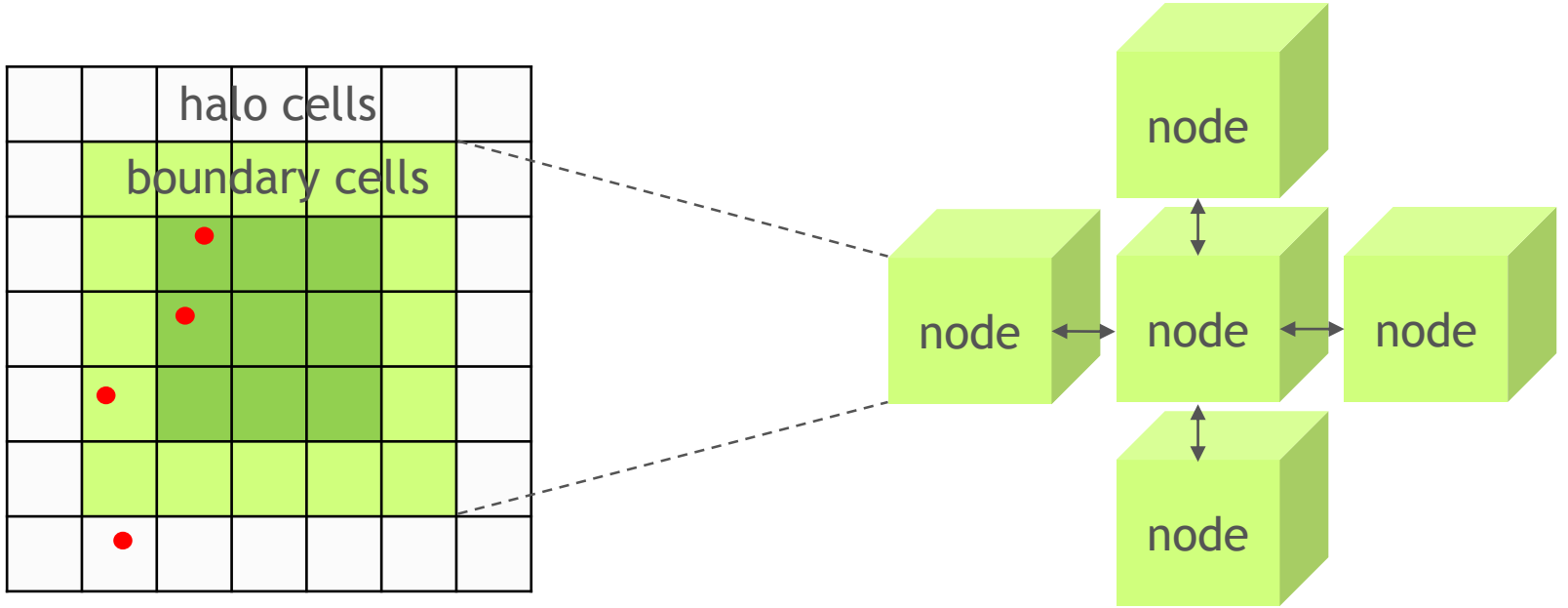
* 2x Ivy Bridge Xeon E5-2690 v2 @ 3.0GHz (20 cores, HT), OpenMP

SINGLE NODE BENCHMARK, T=3000



* 2x Ivy Bridge Xeon E5-2690 v2 @ 3.0GHz (20 cores, HT), OpenMP

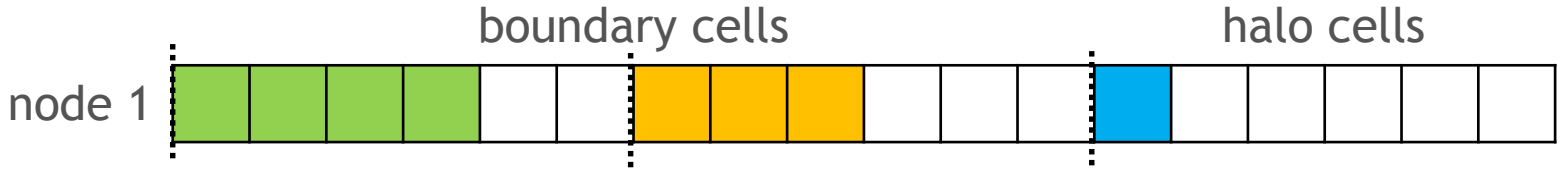
DISTRIBUTED CoMD



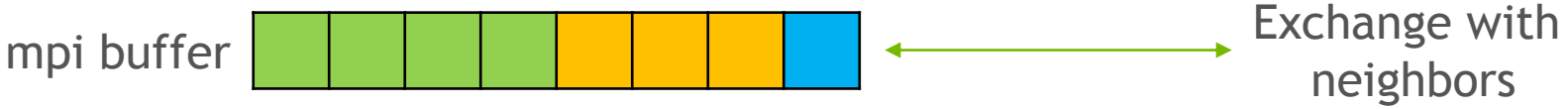
- Clear halos, advance local particles
- Update atoms in boundary/halo cells
- EAM forces: update embedded forces in halo cells

DISTRIBUTED CoMD

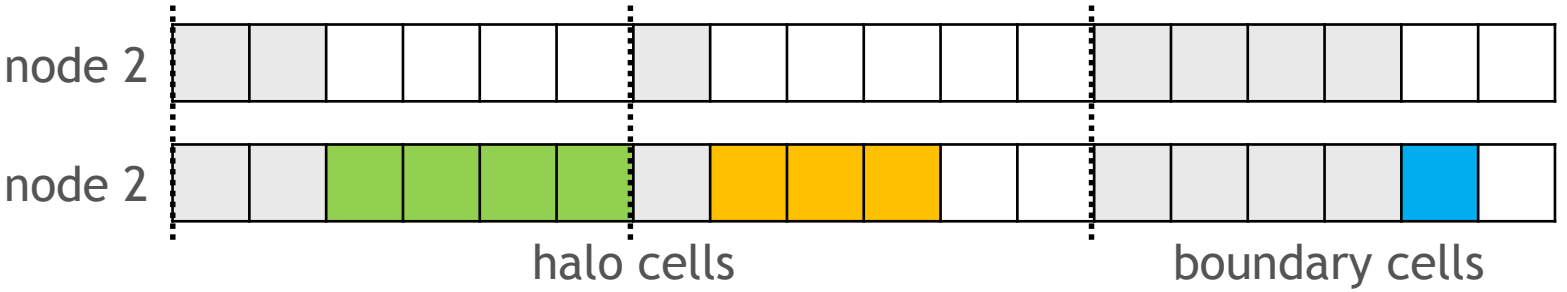
- Exchange halo atoms



  - Data packing using parallel scan

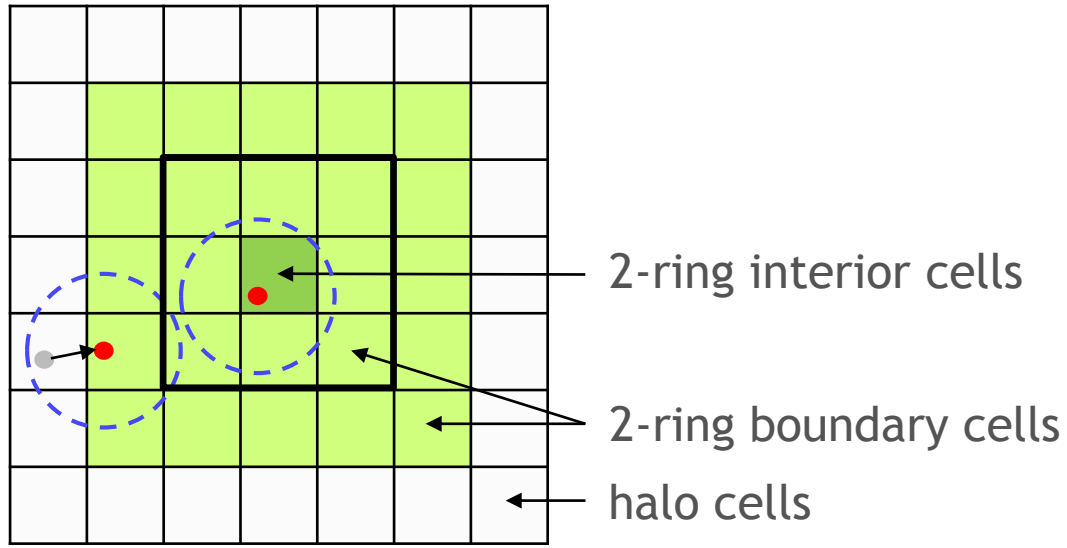


  - Data unpacking using atomics



DISTRIBUTED CoMD OPTIMIZATIONS

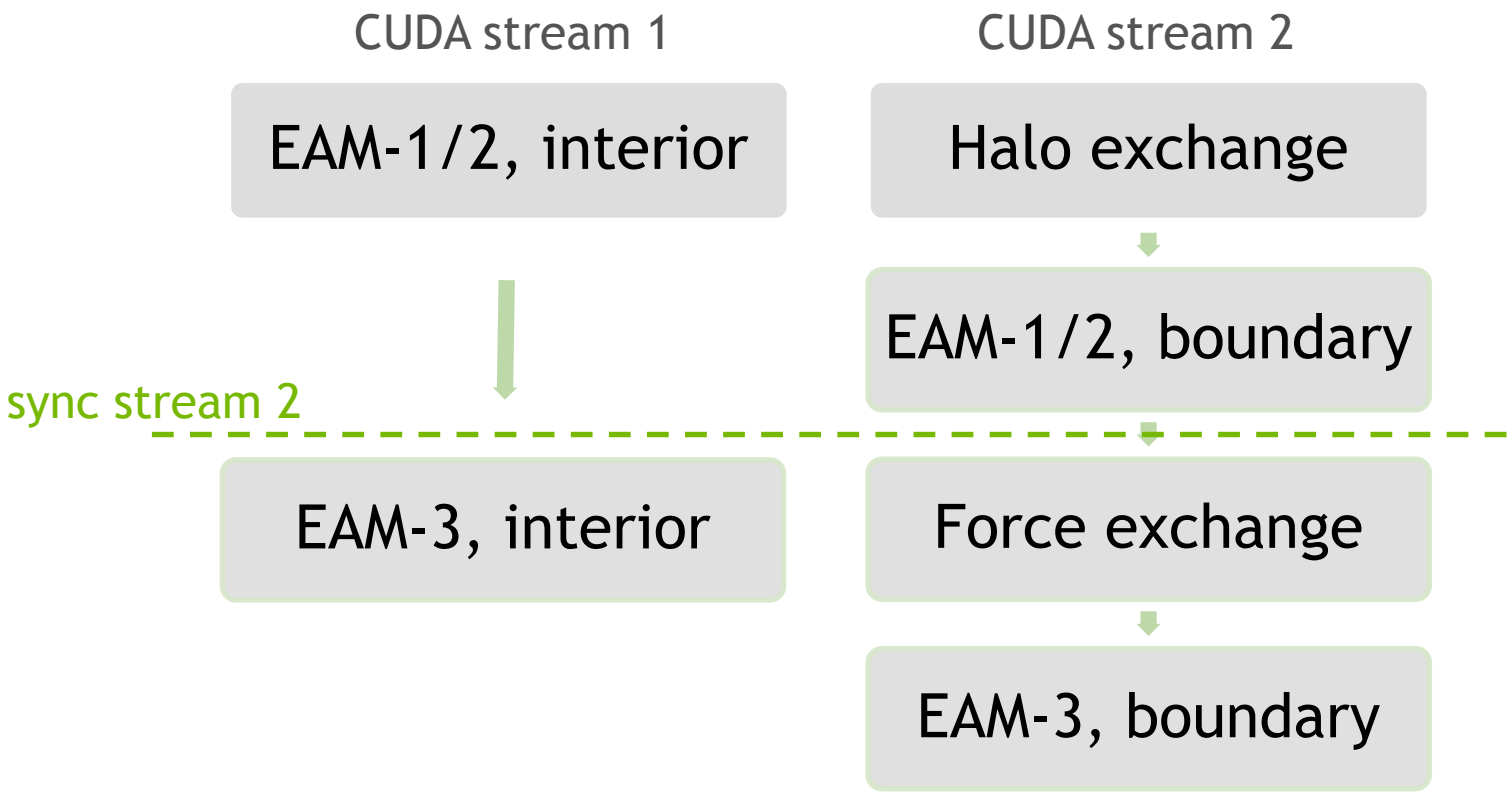
- Introducing 2-ring boundary layer



- Can work on the 2-ring interior cells independently
  - Even before halo exchange is happened

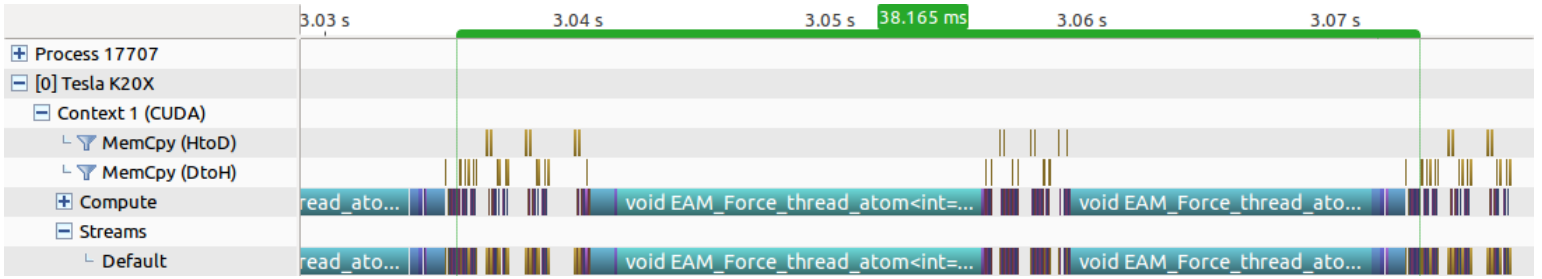
DISTRIBUTED CoMD OPTIMIZATIONS

- Asynchronous processing to hide CPU/MPI latencies

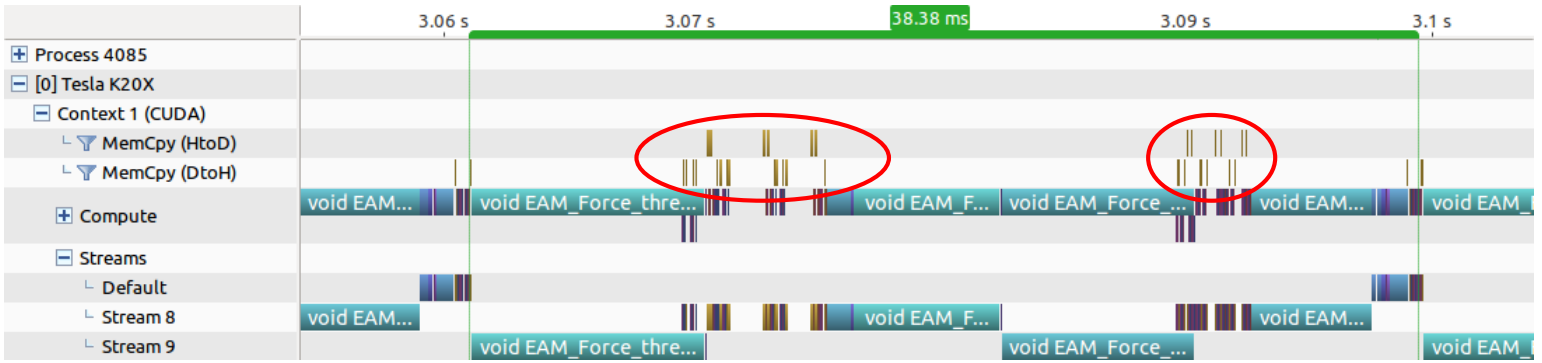


DISTRIBUTED CoMD OPTIMIZATIONS

- No speed-up ☹
  - Heavy EAM kernels take all the GPU, small assembly kernels are stuck



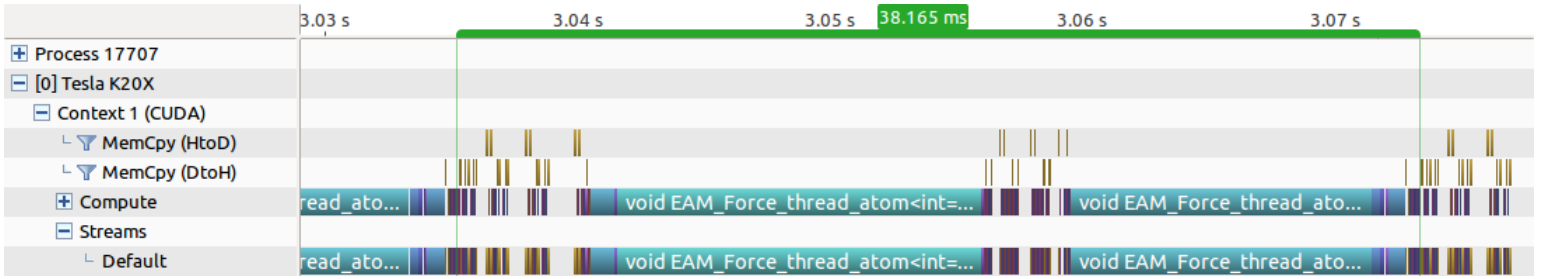
sync
38.2ms



async
38.4ms

DISTRIBUTED CoMD OPTIMIZATIONS

- Use CUDA high priority streams!
  - Preempt heavy kernels and enable faster processing of boundaries

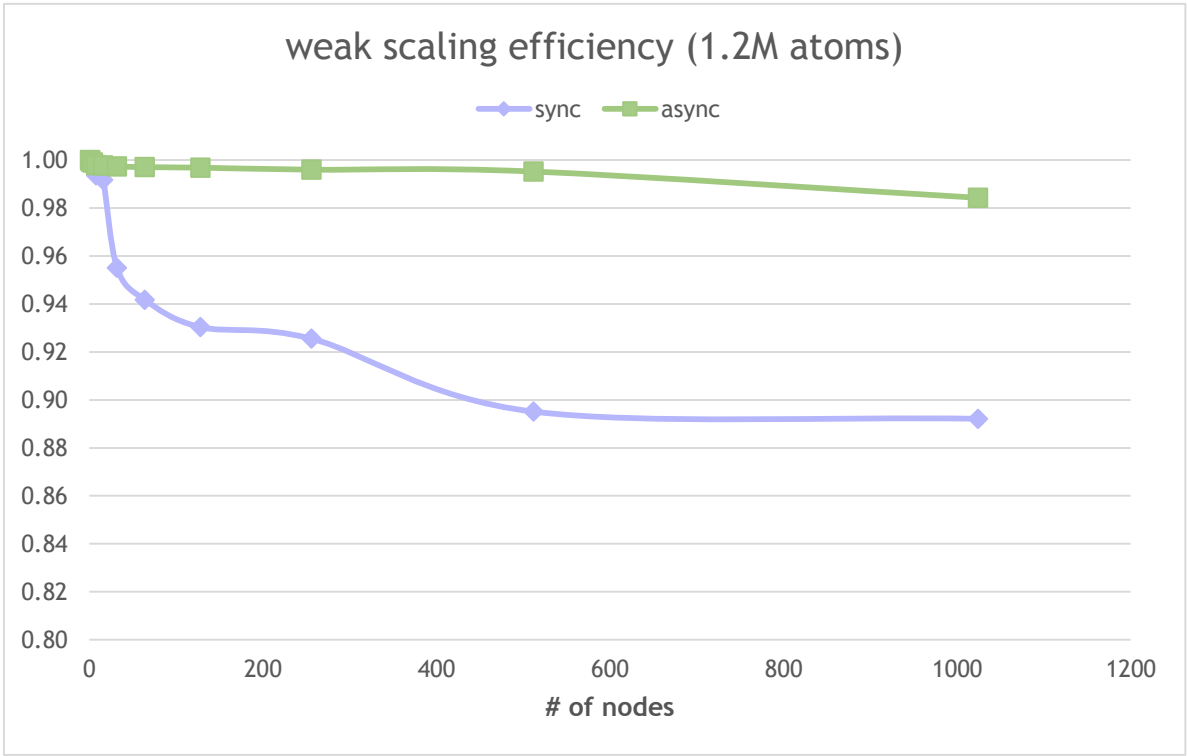


sync
38.2ms

async
29.8ms
(1.3x)

SCALING PERFORMANCE

- ORNL Titan cluster: 1xK20X per node
- >256K atoms can completely hide communications for async



CONCLUSIONS

- Neighbor lists are generally better than cell-based approaches
- Spatial locality is necessary to achieve the best performance
- Asynchronous processing of boundaries helps with the scaling

- Acknowledgements:
  - Paul Springer (NVIDIA intern fall 2013)

- Code is available online:
  - CPU: https://github.com/exmatex/CoMD
  - GPU: https://github.com/nsakharnykh/CoMD-CUDA