JÜLICH
FORSCHUNGSZENTRUM

Multi-Threaded Construction of Neighbour Lists for Particle Systems in OpenMP

08. September 2015 | Rene Halver, Godehard Sutmann

Mitglied der Helmholtz-Gemeinschaft

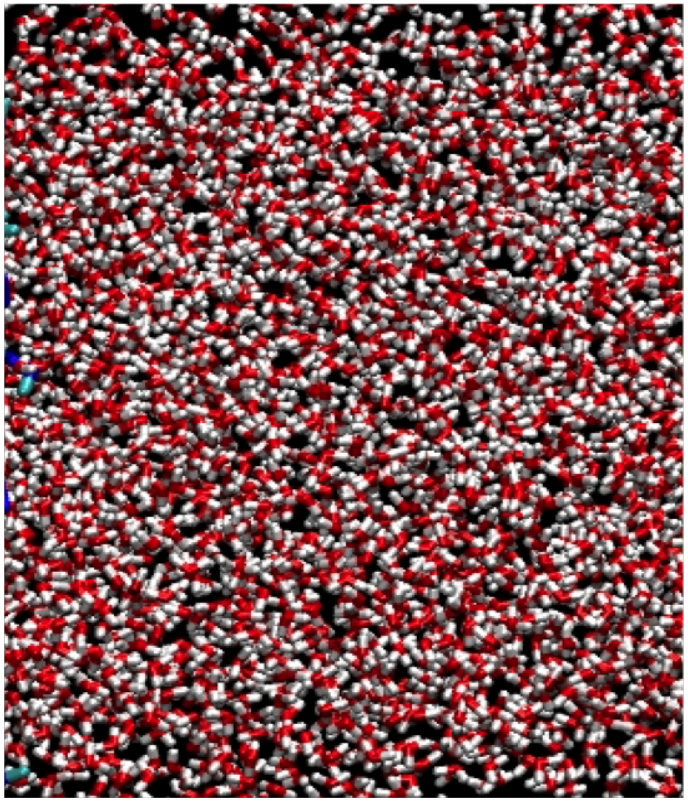
Overview

- Introduction
- Implementation of Neighbour Lists
- Performance
- Outlook

Mitglied der Helmholtz-Gemeinschaft

Molecular Dynamics

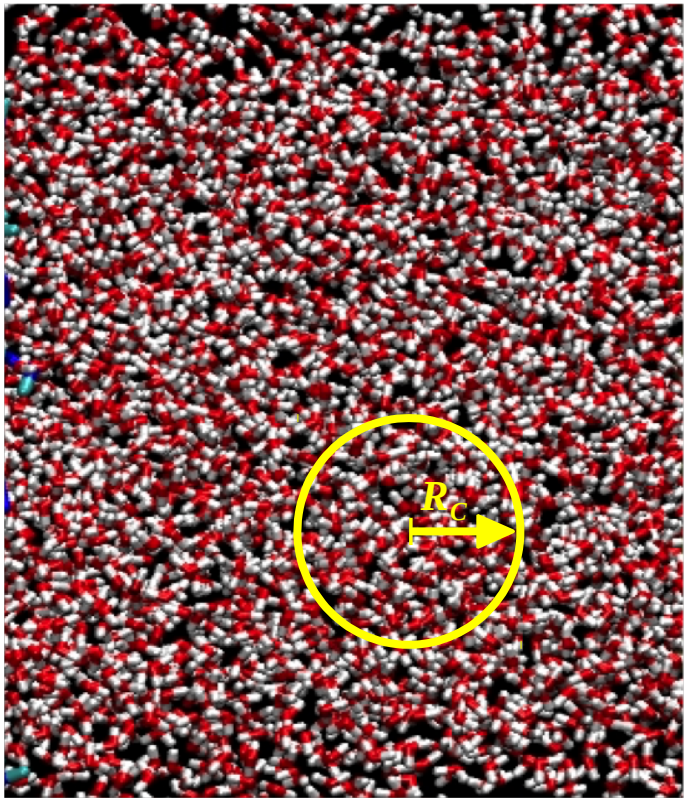
- Simulation of atomistic systems (e.g. polymers, crystals)
- Description of interacting particles
  - here: only short-ranged interactions
    → interactions only if $\|\vec{r}_{ij}\| < r_c$



Mitglied der Helmholtz-Gemeinschaft

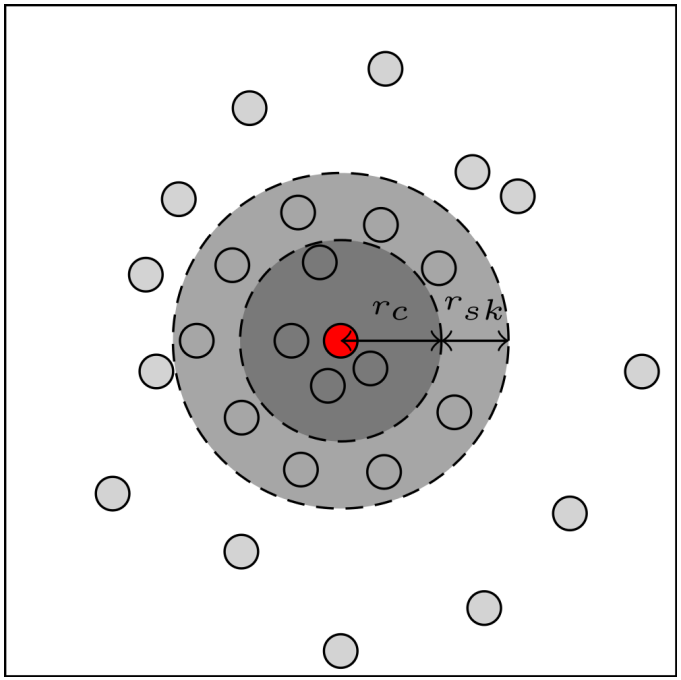
Molecular Dynamics

- Simulation of atomistic systems (e.g. polymers, crystals)
- Description of interacting particles
  - here: only short-ranged interactions
    → interactions only if $\|\vec{r}_{ij}\| < r_c$
- Classical O(N^2) complexity problem



Verlet Lists

- For each particle keep a list of neighbours within cutoff-radius
  - Advantage:
    - ⟶ reduce distance comparisons between non-local particles to once per list creation
  - Disadvantage:
    - ⟶ need to update list regularly in order to account for particle mobility
    - ⟶ reduces only prefactor of complexity (still $O(N^2)$)

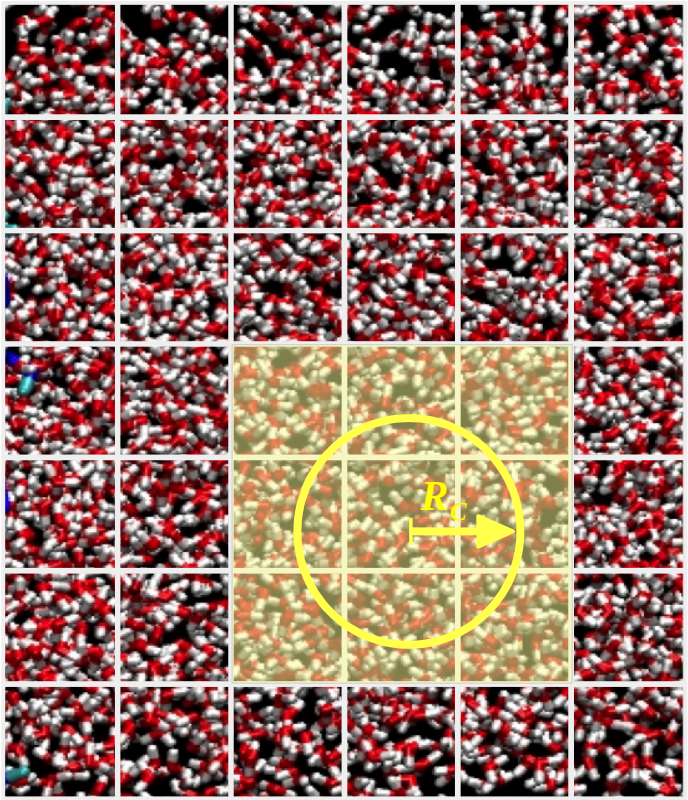


Mitglied der Helmholtz-Gemeinschaft

Molecular Dynamics

- Simulation of atomistic systems (e.g. polymers, crystals)
- Description of interacting particles
  - here: only short-ranged interactions
    → interactions only if $\|\vec{r}_{ij}\| < r_c$
- Classical $O(N^2)$ complexity problem
- Strategy to reduce $O(N^2)$ complexity: neighbour-lists

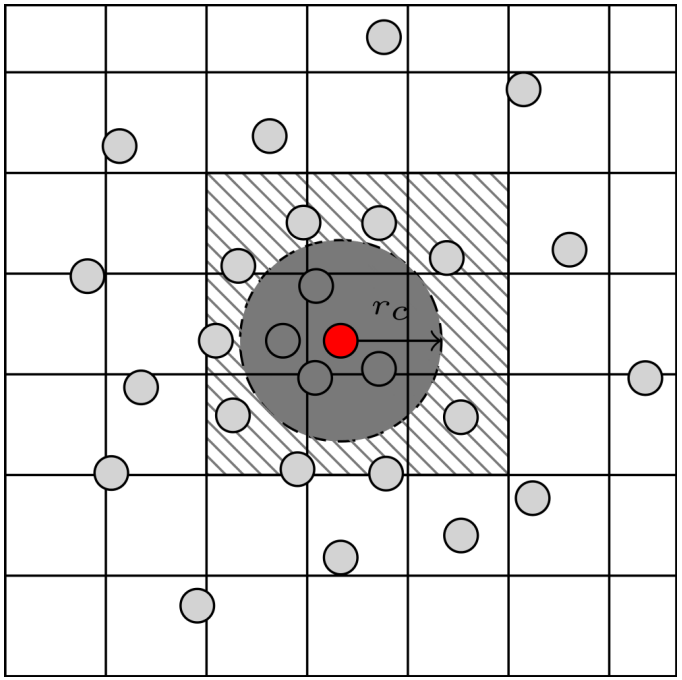
→ Verlet lists with linked-cell pre-sorting



Mitglied der Helmholtz-Gemeinschaft

Linked Cells

- Create a grid of cells with length r_c and sort particles into grid cells, O(N)
- Check only particles in local and direct neighbour cells for possible interactions
  - Advantage:
    - ⟶ complexity of calculation of O(N)
  - Disadvantage:
    - ⟶ particles sorted every timestep
    - ⟶ random memory access to particle data in each cell when addressing particles

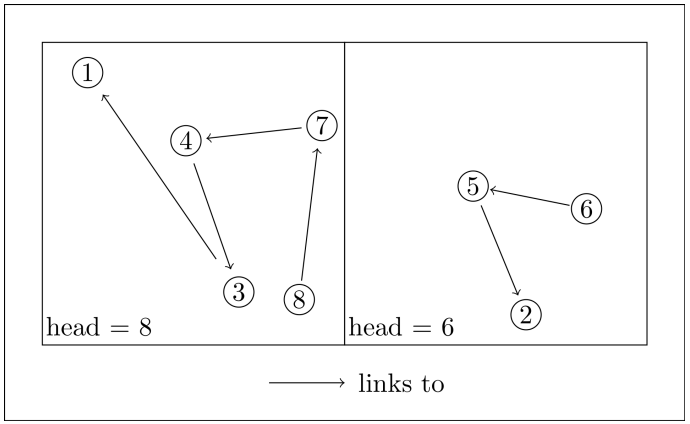


Mitglied der Helmholtz-Gemeinschaft

Sequential implementation of linked cells

- For each particle find the correct cell and construct a linked list:

```
head(:) = 0; list(:) = 0;
do i = 1,n_part
 cell_coords = get_idx(part_pos(:,i),cell_length)
 list(i) = head(cell_coords)
 head(cell_coords) = i
end do
```



Mitglied der Helmholtz-Gemeinschaft

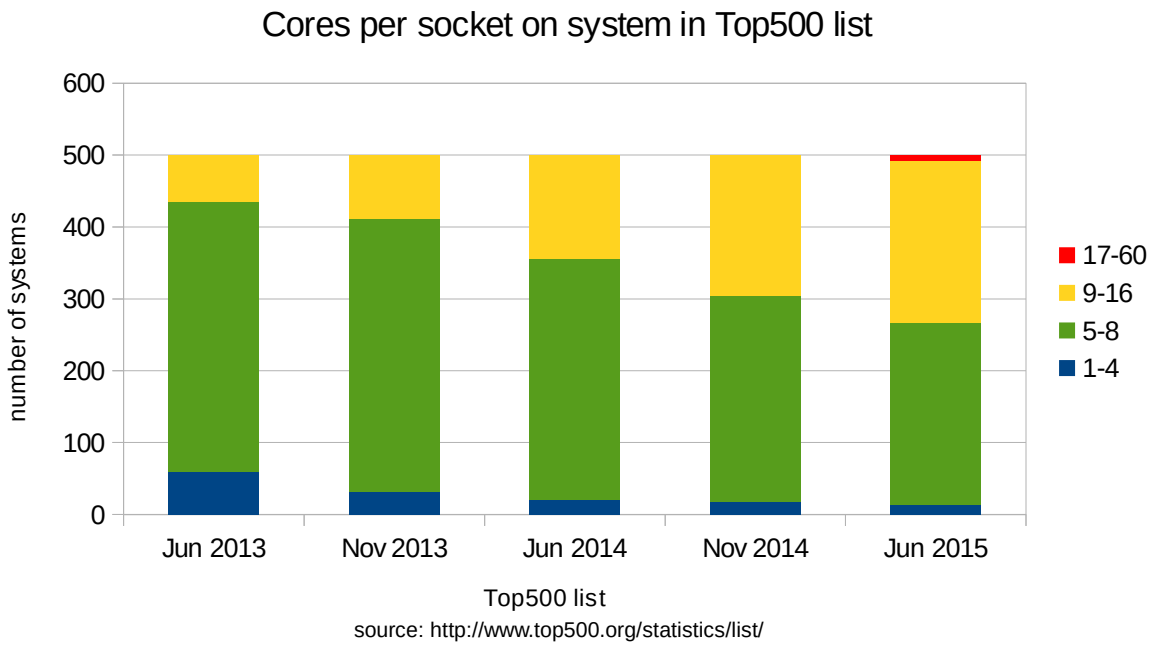
Parallelization Strategies

- Parallelization strategies in Molecular Dynamics:
  - Inter-node parallelization (e.g. MPI)
    - Domain decomposition
    - Force decomposition
    - Particle decomposition
  - Intra-node parallelization (e.g. OpenMP, PThreads)
    - Implementation of hybrid algorithms to calculate forces and construct neighbour lists
    - Sub-domain decomposition

Mitglied der Helmholtz-Gemeinschaft

Why is thread parallelization essential?

- Number of cores per socket in HPC systems increase
- Thread parallelism required to efficiently use systems with SMT capacity



source: http://www.top500.org/statistics/list/

Mitglied der Helmholtz-Gemeinschaft

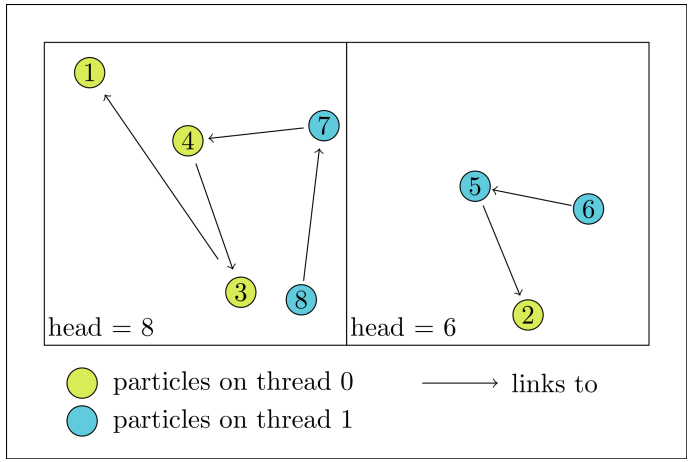
Parallelization difficulties

- Naive way to introduce thread-parallelism with OpenMP:

```
head(:) = 0; list(:) = 0;
!$OMP PARALLEL DO PRIVATE(i,cell_coords)
do i = 1,n_part
 cell_coords = get_idx(part_pos(:,i),cell_length)
 list(i) = head(cell_coords)
 head(cell_coords) = i
end do
!$OMP END PARALLEL
```

- Results in erroneous execution:

  memory access conflicts due to simultaneous updates of cells



Mitglied der Helmholtz-Gemeinschaft

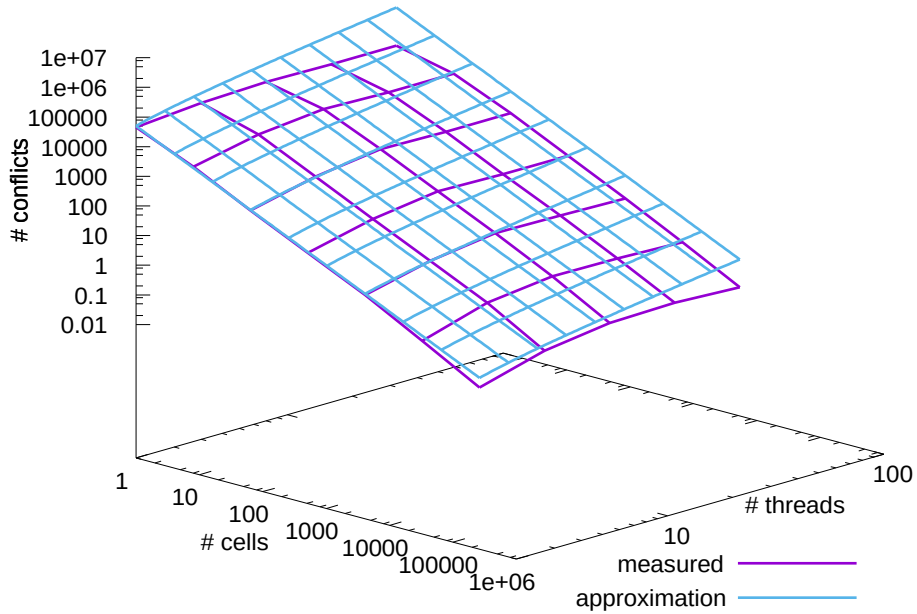
Parallelization difficulties

- Higher probability of memory conflicts, for:

  → increasing particle density (particles per cell)
  → increasing thread number

- Estimated conflicts:

$$n_{conflicts} = \frac{n_{part} \cdot (n_{threads} - 1)}{2 \cdot n_{cells}}$$



Solution 1: Copies

- Provide a copy of the array *head* for each thread
- Each thread first sorts locally its assigned particles

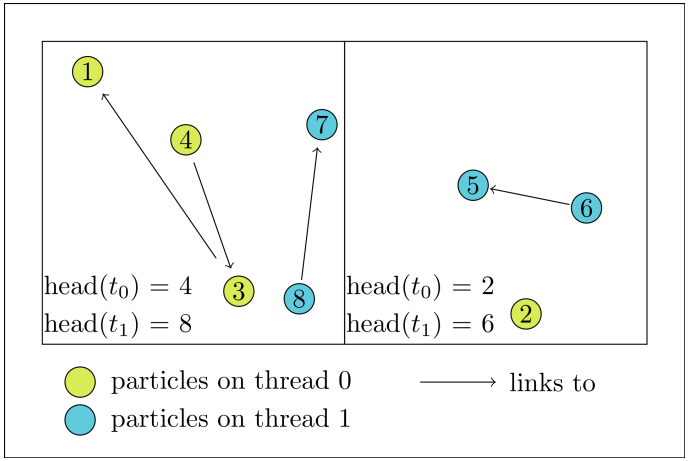
```
head(:,:) = 0; list(:) = 0;
!$OMP PARALLEL PRIVATE(i,cell_coords,tid)
tid = OMP_GET_THREAD_NUM()
do i = start_idx(tid),end_idx(tid)
 cell_coords = get_idx(part_pos(:,i),cell_length)
 list(i) = head(cell_coords,tid)
 head(cell_coords,tid) = i
end do
!$OMP END PARALLEL
```

Mitglied der Helmholtz-Gemeinschaft

Solution 1: Copies

- Combine the partial lists into completed cell structure

- Advantages:
  - → No data dependence between threads
  - → No synchronization between threads needed



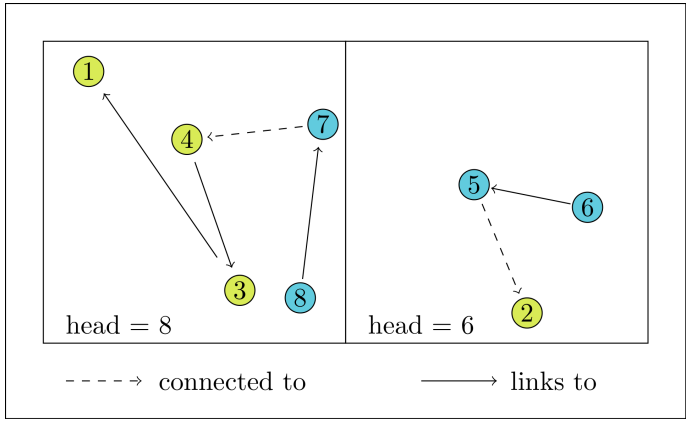
- Disadvantages:
  - → Memory demand linearly increases with thread count
  - → Two steps required to reach completed list of cells

Mitglied der Helmholtz-Gemeinschaft

Solution 1: Copies

- Combine the partial lists into completed cell structure

- Advantages:
  - → No data dependence between threads
  - → No synchronization between threads needed

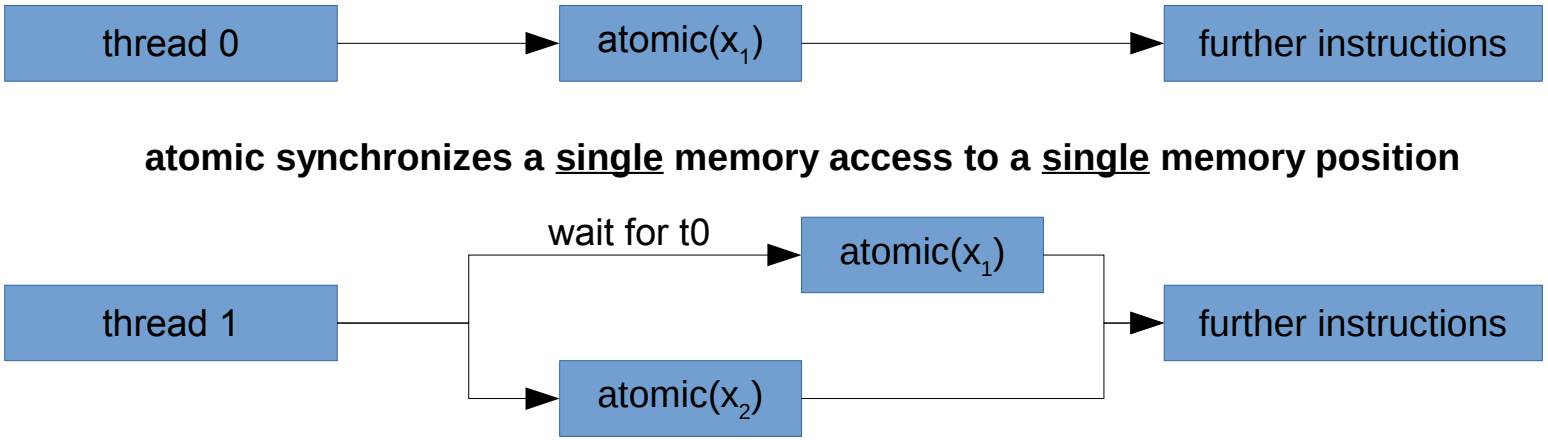


- Disadvantages:
  - → Memory demand linearly increases with thread count
  - → Two steps required to reach completed list of cells

Mitglied der Helmholtz-Gemeinschaft

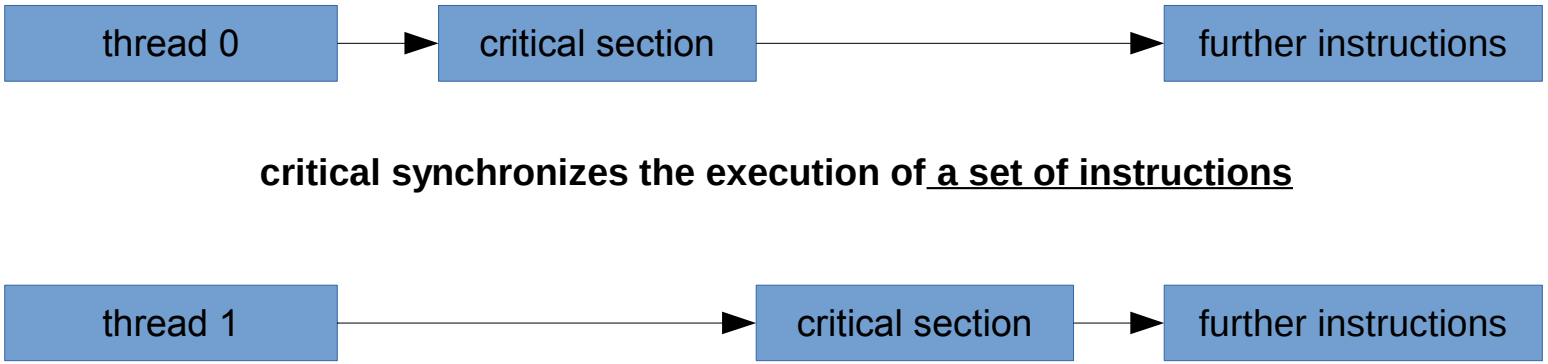
Solution 2: Synchronizing threads

- Synchronize the threads to avoid access of the same cell by different threads at the same time
- OpenMP provides the following synchronization tools:



Solution 2: Synchronizing threads

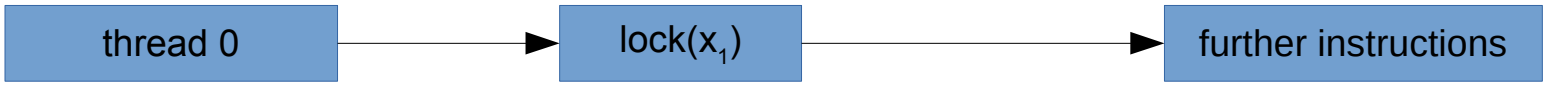
- Synchronize the threads to avoid access of the same cell by different threads at the same time
- OpenMP provides the following synchronization tools:



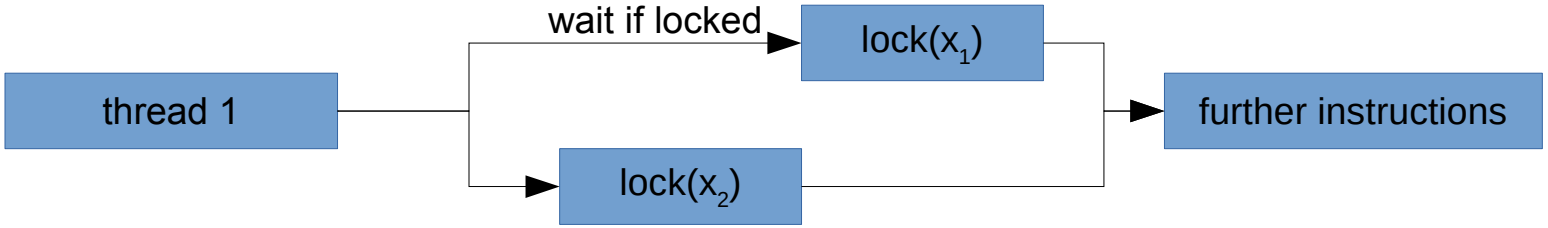
Mitglied der Helmholtz-Gemeinschaft

Solution 2: Synchronizing threads

- Synchronize the threads to avoid access of the same cell by different threads at the same time
- OpenMP provides the following synchronization tools:



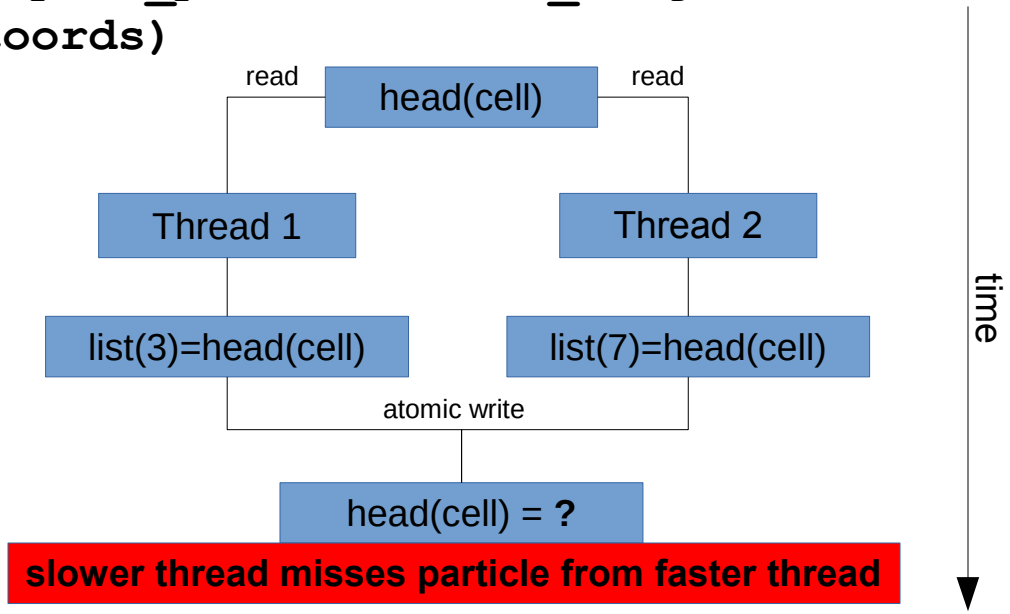
locks provide a more flexible way to synchronize access to a set of instructions



Mitglied der Helmholtz-Gemeinschaft

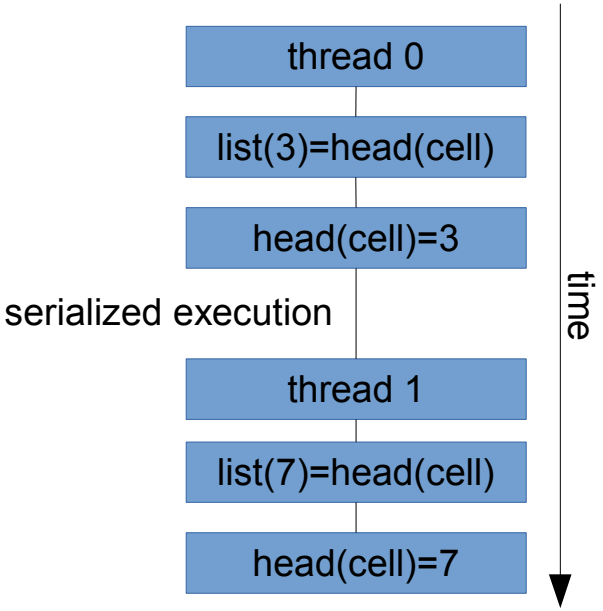
Solution (?) 2: Synchronizing with atomic

```
head(:) = 0; list(:) = 0;
!$OMP PARALLEL PRIVATE(i,cell_coords,tid)
tid = OMP_GET_THREAD_NUM()
do i = start_idx(tid),end_idx(tid)
  cell_coords = get_idx(part_pos(:,i),cell_length)
  list(i) = head(cell_coords)
  !$OMP ATOMIC
  head(cell_coords) = i
  !$OMP END ATOMIC
end do
!$OMP END PARALLEL
```



Solution (?) 2: Synchronizing with critical

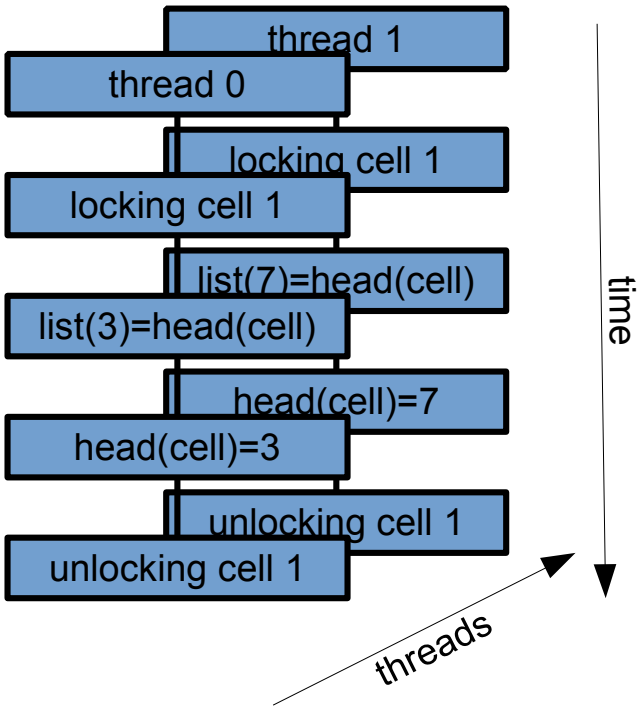
```
head(:) = 0; list(:) = 0;
!$OMP PARALLEL PRIVATE(i,cell_coords,tid)
tid = OMP_GET_THREAD_NUM()
do i = start_idx(tid),end_idx(tid)
  cell_coords = get_idx(part_pos(:,i),cell_length)
  !$OMP CRITICAL
  list(i) = head(cell_coords)
  head(cell_coords) = i
  !$OMP END CRITICAL
end do
!$OMP END PARALLEL
```



Mitglied der Helmholtz-Gemeinschaft

Solution 2: Synchronizing with locks

```
do i = 1,n_cells
 call OMP_INIT_LOCK(locks(i))
end do
head(:) = 0; list(:) = 0;
!$OMP PARALLEL PRIVATE(i,cell_coords,tid)
tid = OMP_GET_THREAD_NUM()
do i = start_idx(tid),end_idx(tid)
  cell_coords = get_idx(part_pos(:,i), &
  cell_length)
  call OMP_SET_LOCK(locks(cell_coords))
  list(i) = head(cell_coords)
  head(cell_coords) = i
  call OMP_UNSET_LOCK(locks(cell_coords))
end do
!$OMP END PARALLEL
```



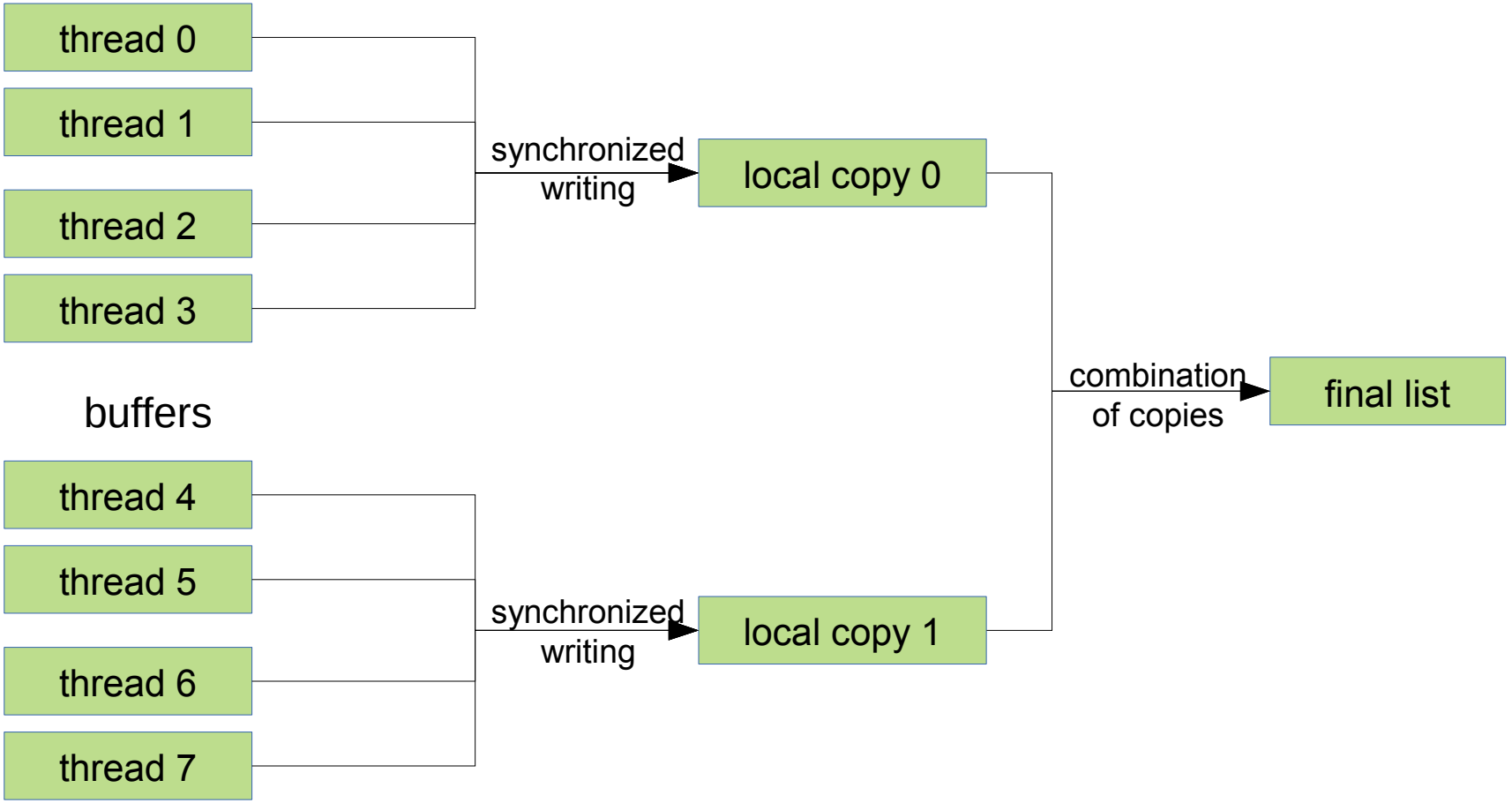
Mitglied der Helmholtz-Gemeinschaft

Solution 2: Synchronizing with locks

- Advantages:
  - → Good scalability
  - → Better performance than critical sections
- Disadvantages:
  - → Additional memory requirement (small compared to copy variant)
  - → Overhead due to lock administration (factor 3-8)
  - → Runtime gain only for large number of threads

Mitglied der Helmholtz-Gemeinschaft

Solution 3: Combining Copies and Syncs

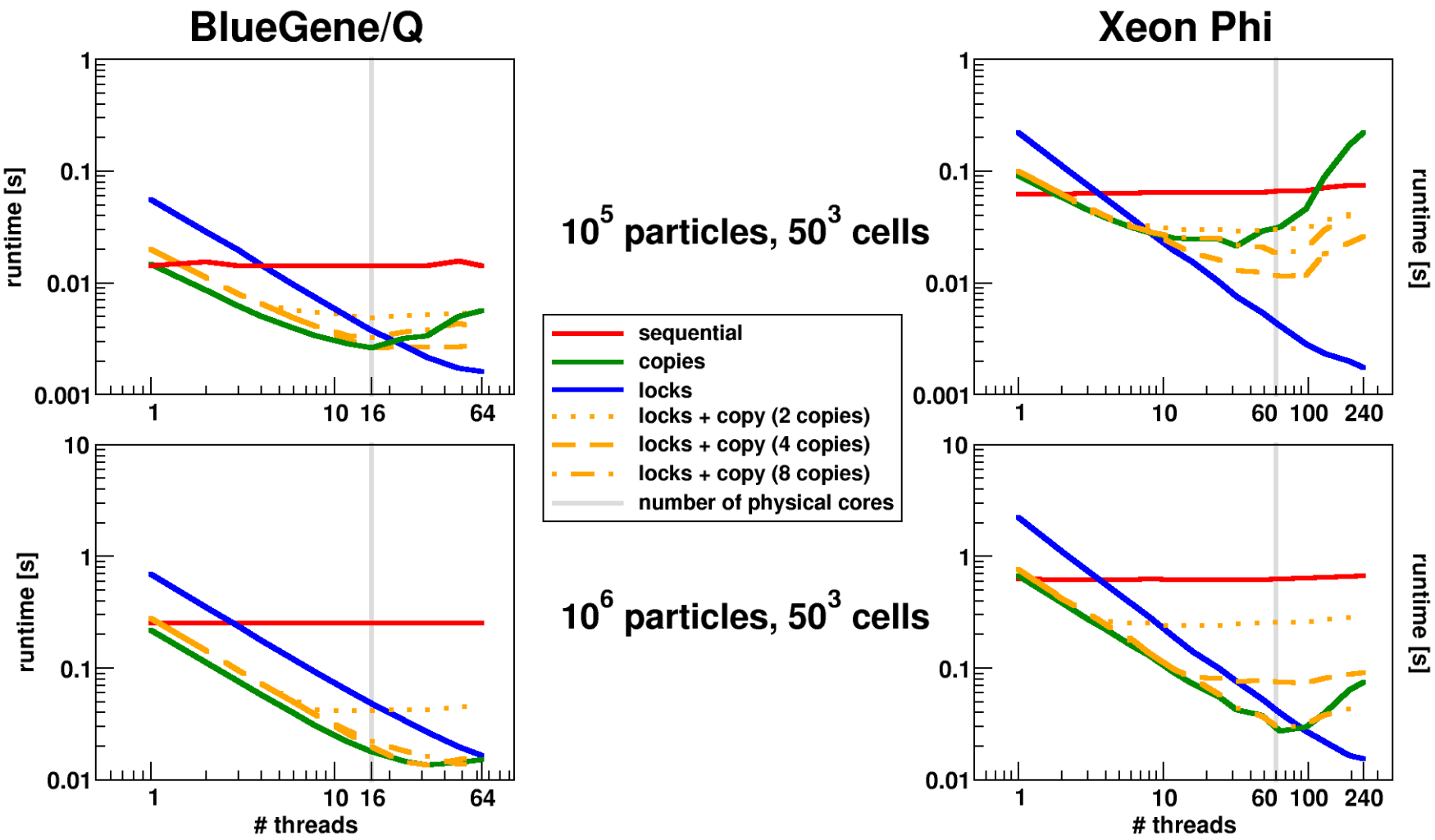


Variant 3: Combining Copies and Syncs

- Number of local copies influences scaling behavior (difficult to predict performance)
- Advantages:
  → One lock for each copy required
  (not one for each cell)
  → Hiding of synchronization times
  (threads continue to fill buffers during synchronization)
- Disadvantages:
  → Implementation complexity enhanced
  → Still higher memory requirement than synchronization-only variants due to copies

Mitglied der Helmholtz-Gemeinschaft

Performance



Mitglied der Helmholtz-Gemeinschaft

Conclusions

- Lock solution:
  - useful for large number of threads (>32)
  - overhead reduces parallel efficiency
- Copy solution:
  - parallel efficiency degrades with decreasing particle count per cell (overhead of empty cell administration)
  - preferable for smaller number of threads (memory demand)
- Atomic and critical not applicable
- Similar scaling behavior on BlueGene/Q and XeonPhi

Mitglied der Helmholtz-Gemeinschaft

Outlook

- Further analysis:
  → combination with Verlet-lists
  → force calculations
  → comparison of different compilers (GNU, XL, Intel, ...)
- Are OpenMP tasks (with data dependences) a competitive implementation (OpenMP 4.0)?
- Comparison of pure OpenMP parallelization to hybrid MPI/OpenMP implementations (BlueGene/Q)

Mitglied der Helmholtz-Gemeinschaft

Thank you for your attention