IWOCL, May 2026

Using Intel Shared System USM to ease porting applications to run on GPUs with SYCL – a GROMACS case study.

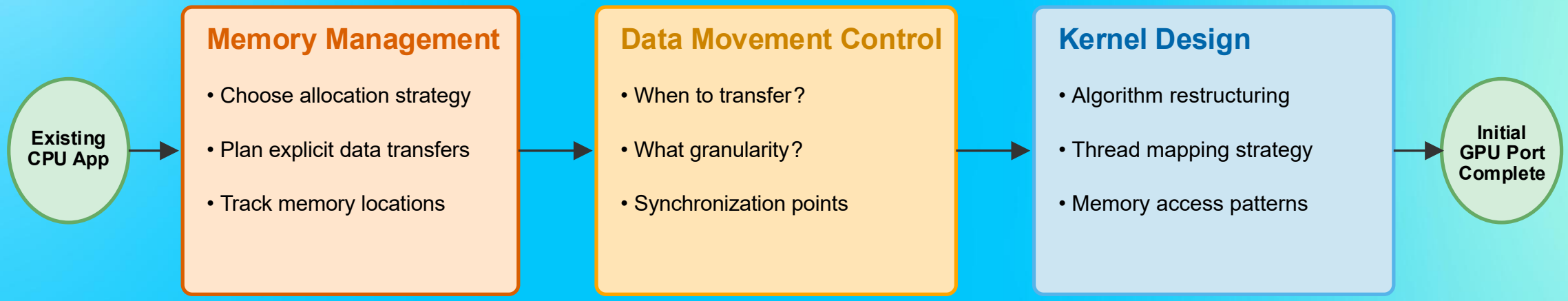
Mark Abraham & François Dugast

Software Engineers @ Intel

Agenda

- Why shared system USM?
- N-body introduction
- Performance results

Challenges when starting a GPU port...



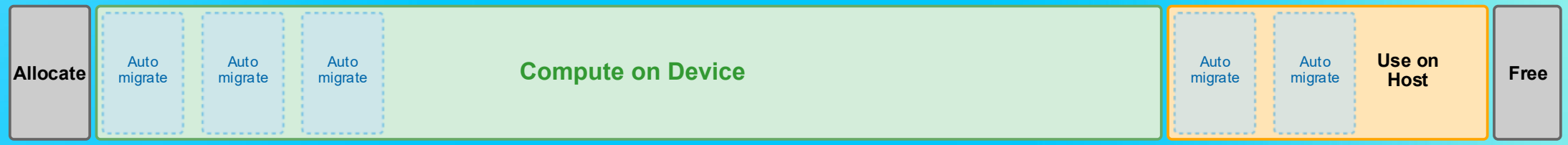
User must manage all complexity manually

Comparison of memory-management workflows

Traditional (Complex for user)

User manages all data movement

Shared System USM (Simpler for user)



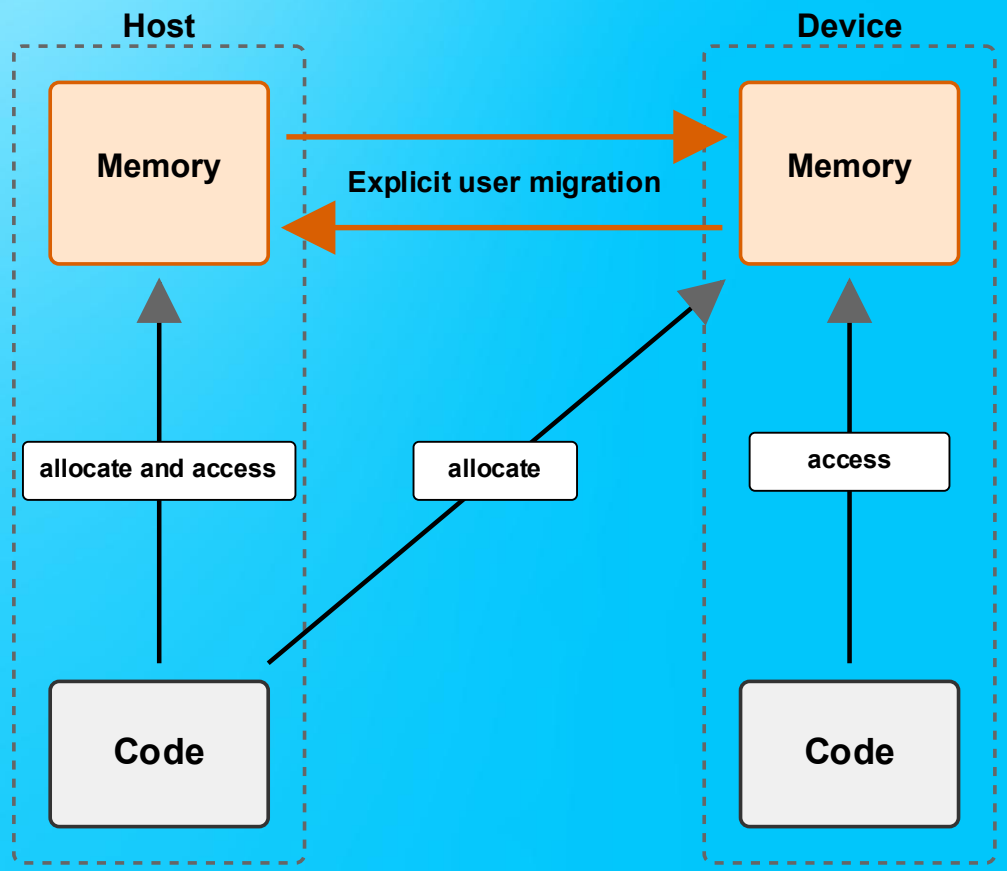
OS kernel manages data movement automatically

Time

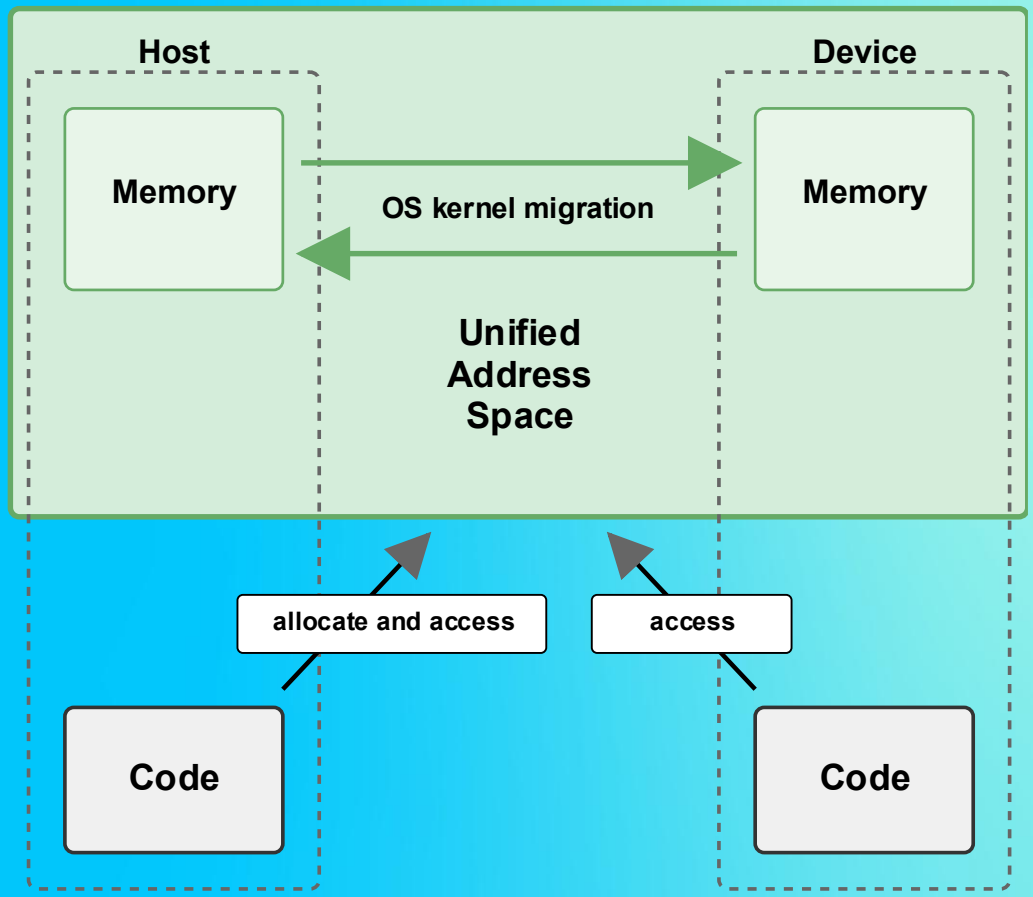
Legend: Allocation/Free | Explicit copy | Compute on device | Automatic migration | Host task

Some implementation details

Traditional (Complex for user)

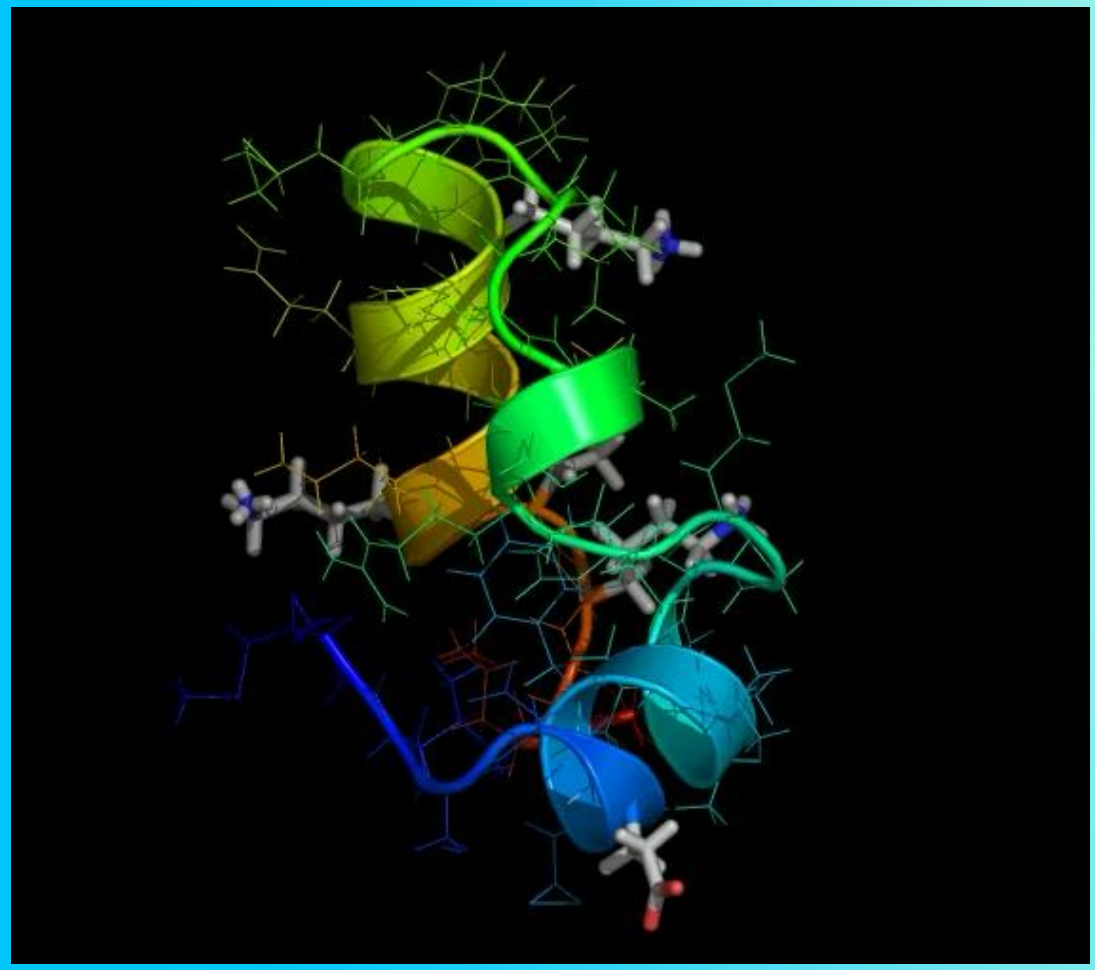


Shared System USM (Simpler for user)

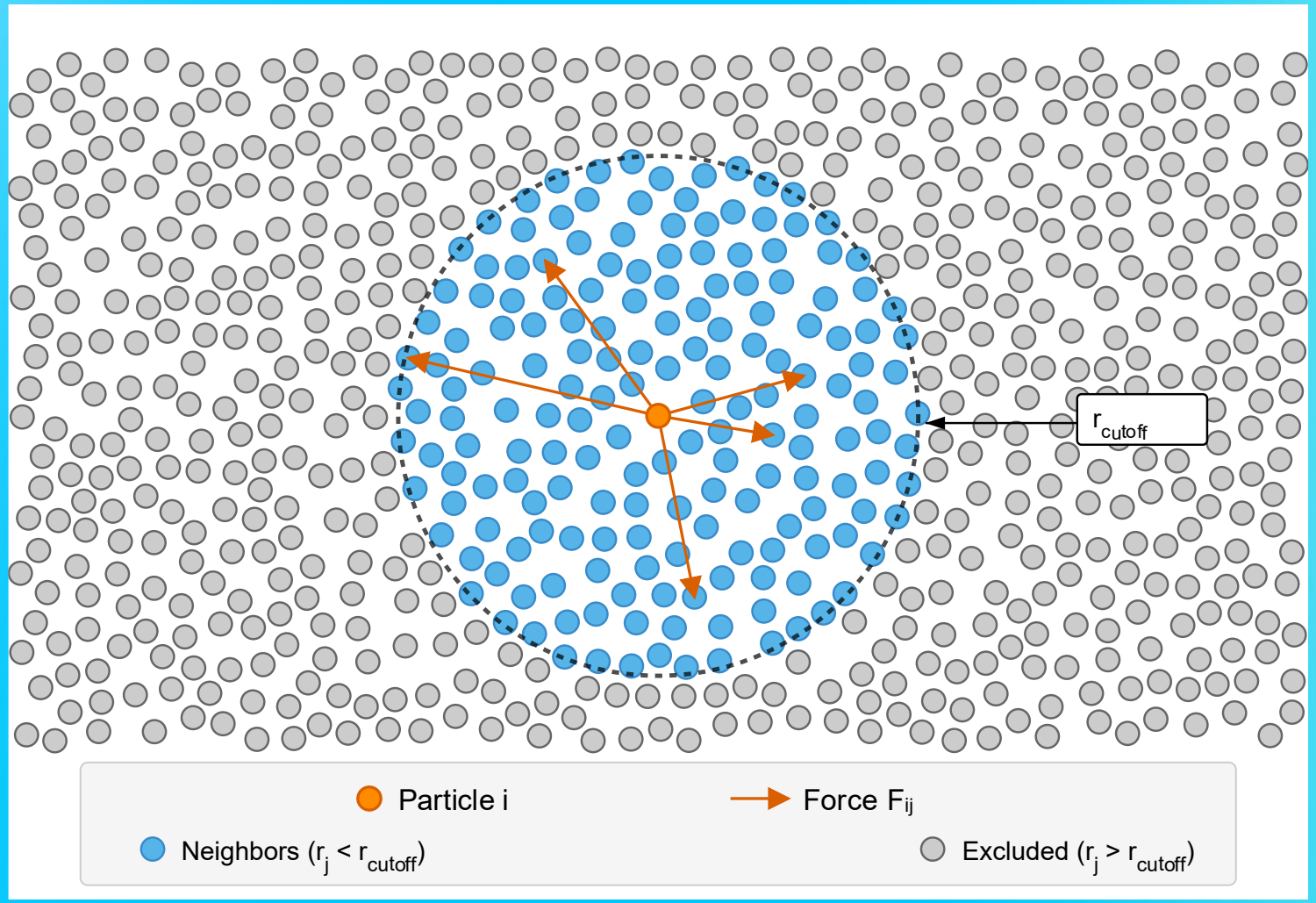


Using shared system USM in GROMACS

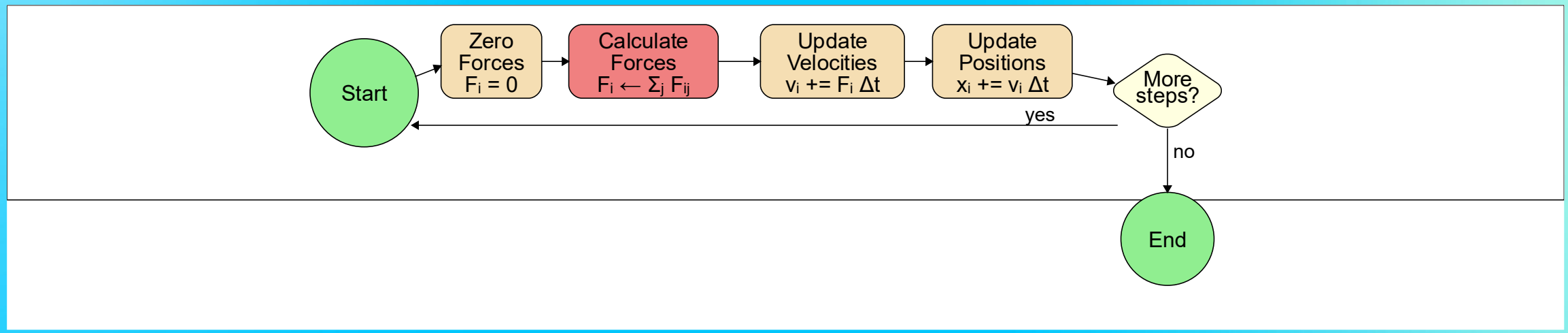
- GROMACS is
  - well known
  - high-performance
  - biomolecular simulation code
  - featuring N-body kernels with irregular stencils
- Ported to and optimized for all GPU frameworks…
- … but we're going to ignore that and pretend to start to port it again, using shared system USM



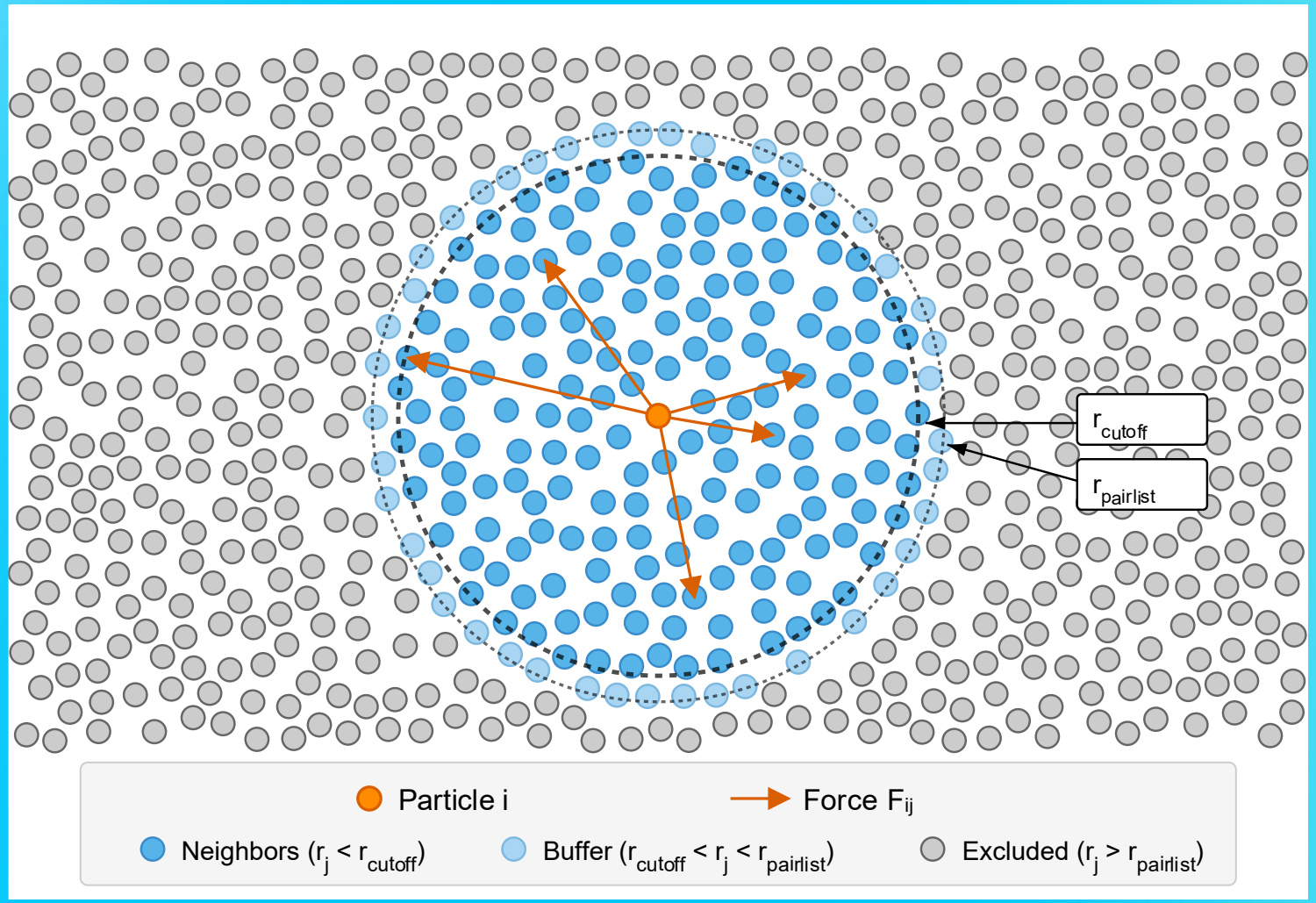
Particle interactions with cutoff radius



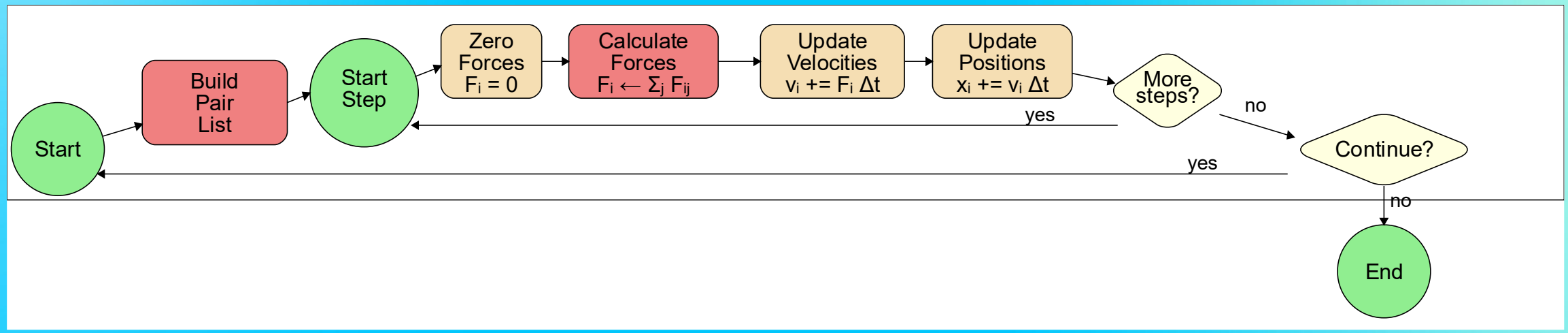
Molecular dynamics algorithm



Particle interactions with cutoff and pair-list radius

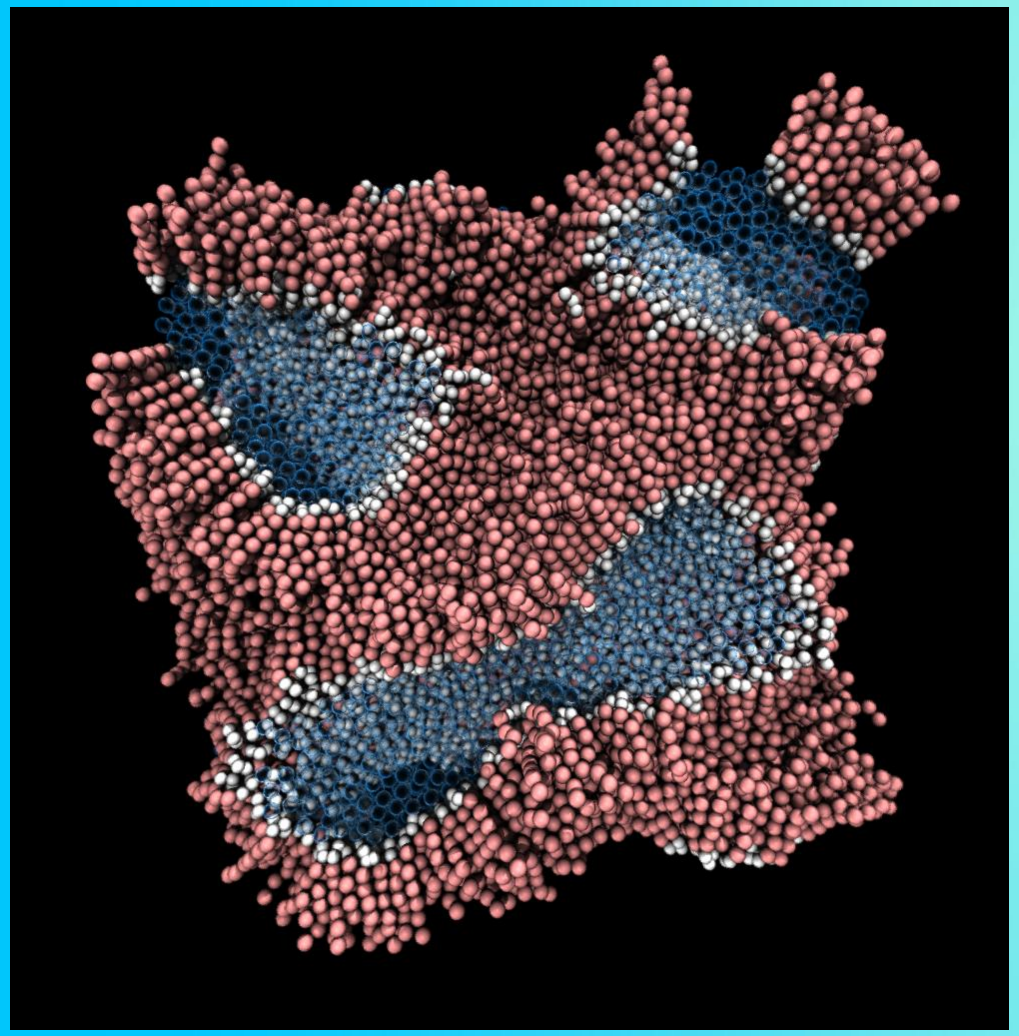


Molecular dynamics algorithm with pair list



How might we start to port this to GPUs?

- Profile to find the bottlenecks:
  - That's the N-body force kernel
- Identify the elements of potential parallelism:
  - Each i atom works on multiple j atoms in a double loop of irregular length
- Design the first GPU version of that compute kernel:
  - Make a single work-item handle the pair list for all j atoms of an i atom
- Use shared system USM:
  - Leave allocations alone
  - Let the OS kernel do the transfers



The shared system USM N-body kernel

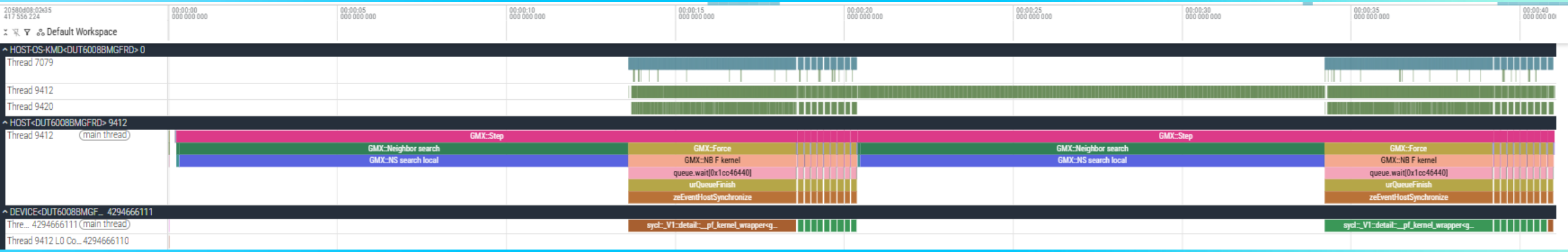
- Almost identical to GROMACS reference CPU kernel
- Claude code wrote this correctly the first time!
  - Each work item handles a single i atom
    - Loops over all the j atoms
    - Compute force between the two atoms
    - Accumulate force contribution to each atom
- Slideware kernel coming up
  - Details of the functional form omitted
  - Only x dimension shown
  - Various verbose details elided

```
queue.parallel_for<Nbnxn1x1_SystemUSM>(sycl::range<1>(numIAtoms), [=](sycl::id<1> idx) {
    const int iEntry = idx[0];
    // Load i-atom parameters from pairlist
    const int ci          = ilist[iEntry].ci;
    const int cjIndBegin = ilist[iEntry].cj_ind_start;
    const int cjIndEnd   = ilist[iEntry].cj_ind_end;
    // ... more preparations related to the i particle

    // Loop over j-atoms in pairlist
    for (int cjInd = cjIndBegin; cjInd < cjIndEnd; cjInd++)
    {
        const int cj = jlist[cjInd].cj;
        // Load j-atom data
        const int  js = cj * xstride;
        const real xj = x_ptr[js + XX];

        // ... enforce cutoff and compute forces

        // Increment i-atom and j-atom forces
        ifx += fx;
        sycl::atomic_ref<float> atomic_fjx(f_ptr[cj * fstride + XX]);
        atomic_fjx.fetch_add(-fx); // Newton's third law
    }
    // Store the i-atom forces (atomic updates for thread safety)
    sycl::atomic_ref<float> atomic_fix(f_ptr[ai * fstride + XX]);
    atomic_fix.fetch_add(fix);
}).wait();
```

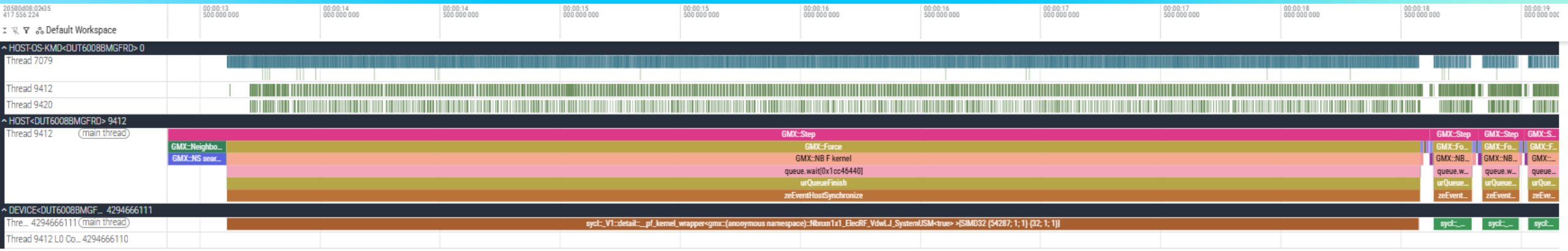
unitrace

- Open-source tracing tool from Intel: https://github.com/intel/pti-gpu
- Integrates tracing from
  - GPU
  - KMD (bpftrace)
  - LevelZero
  - SYCL (XPTI)
  - GROMACS (ITT)
- Presented via http://ui.perfetto.dev – tracing visualizer

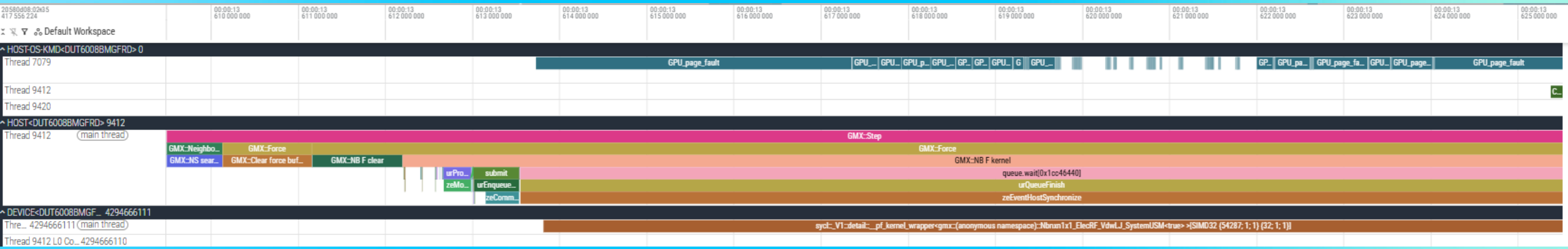
Trace of 19 steps on 1.5M atoms on Intel Arc B570



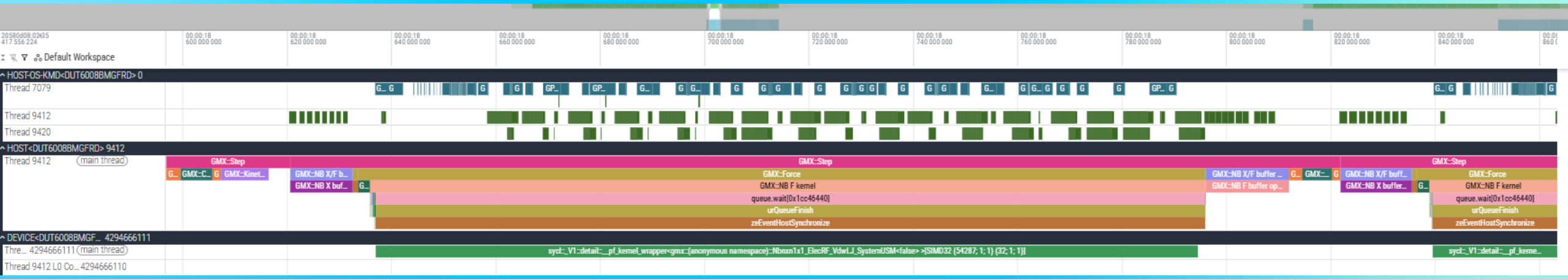
Zoom on the first step after pair search



Zoom on the first part of the first step after pair search

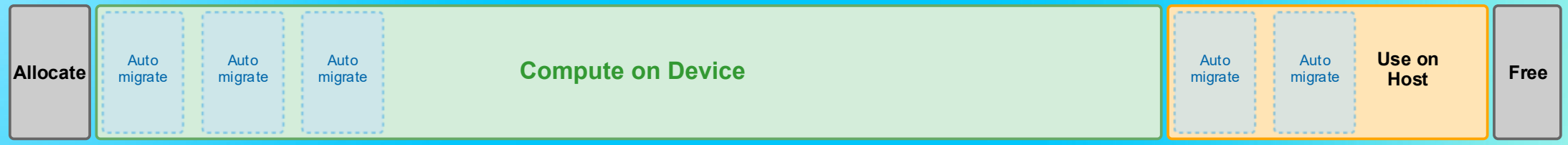


Zoom on the second step after pair search



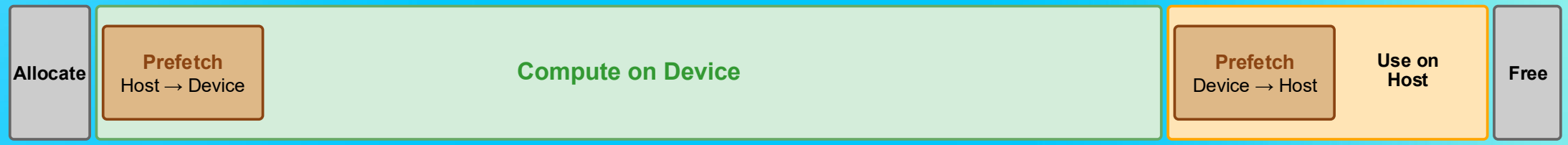
Comparison of memory-management workflows

Shared System USM (Simpler for user)

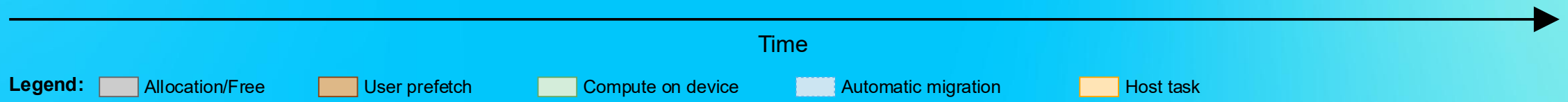


OS kernel manages data movement automatically

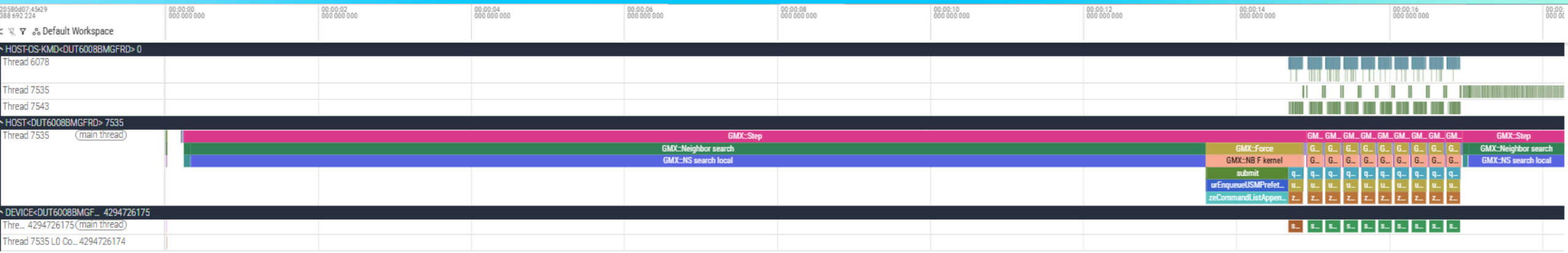
Shared System USM + Prefetches (Less simple for user)



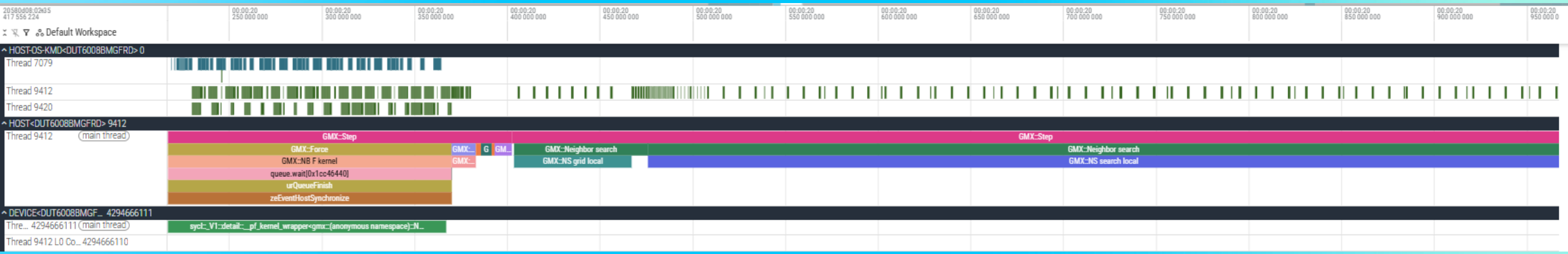
User explicitly prefetches data before use



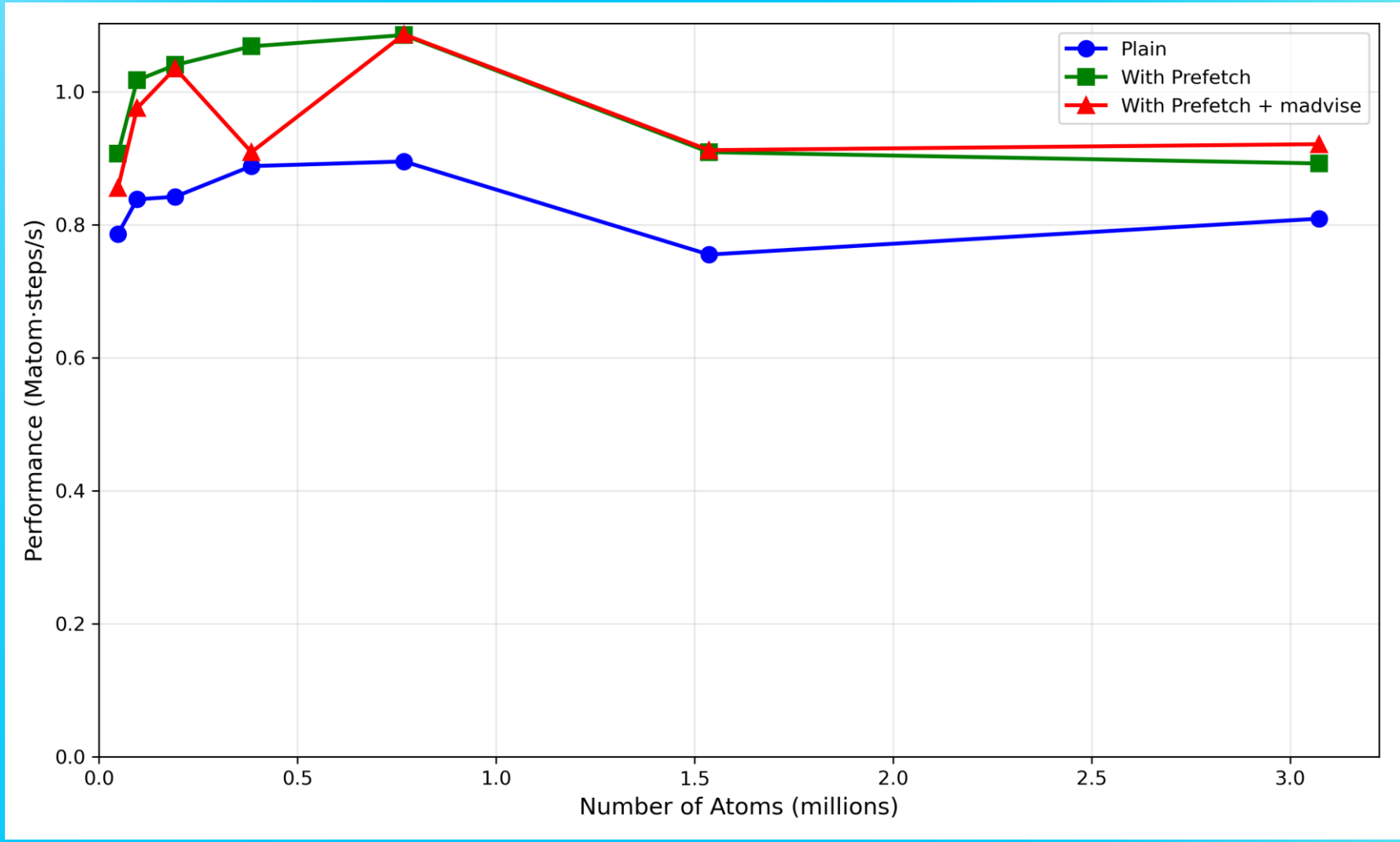
Trace of 9 steps with prefetch of 3GB pair list



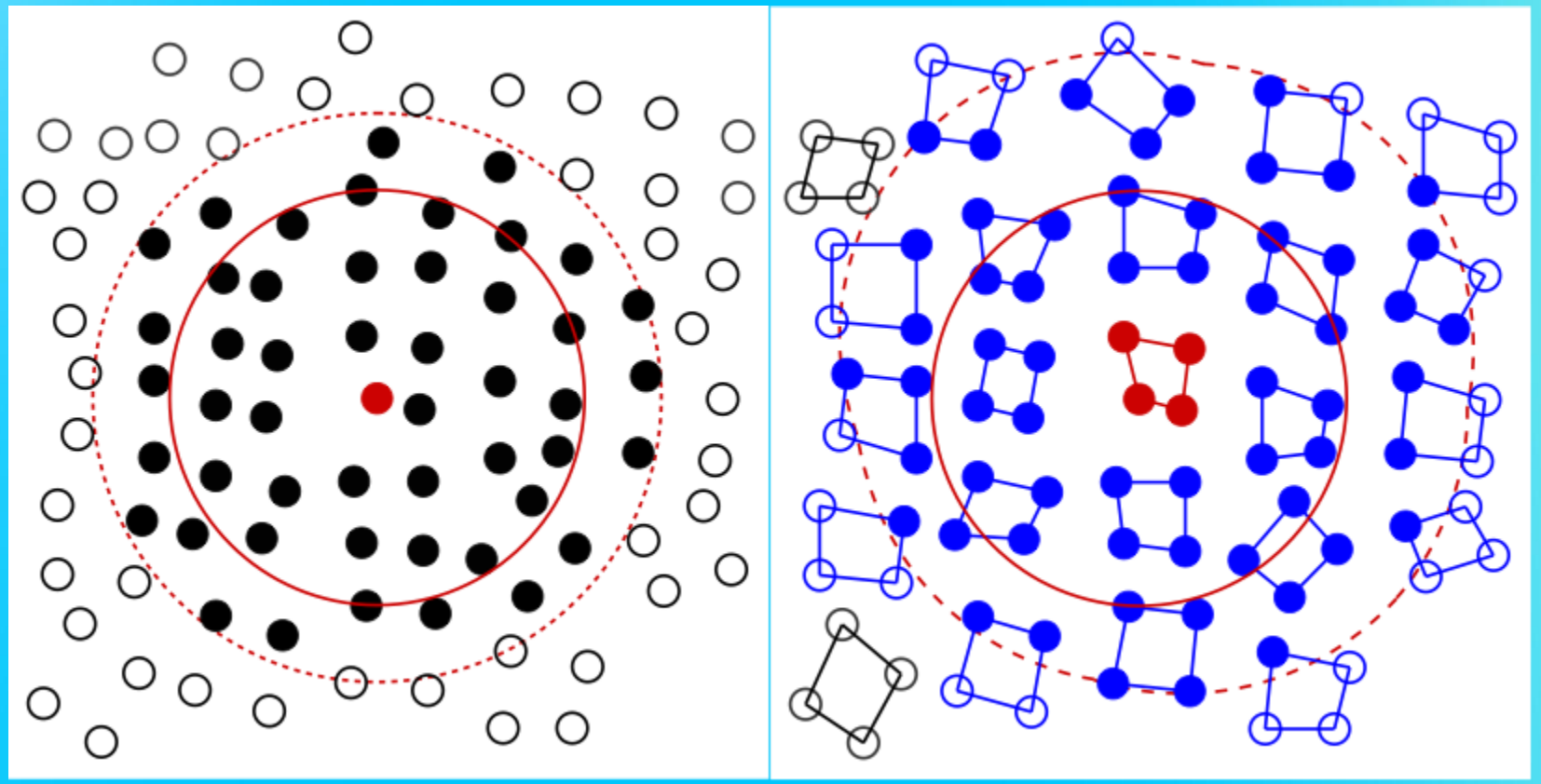
Zoom on the start of the second pair search



Weak scaling performance

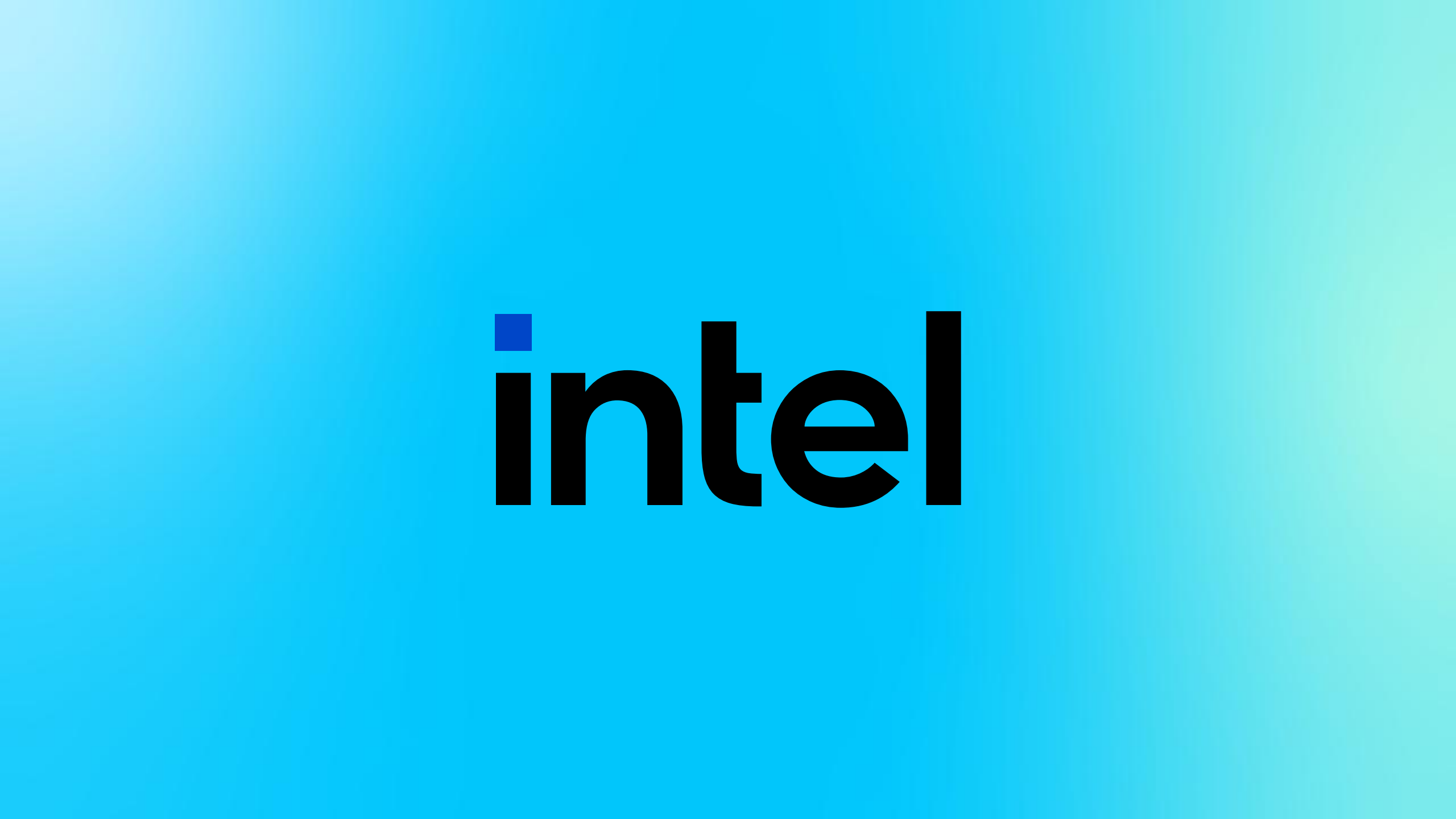


Further optimizations – clusters of particles



Conclusion

- An initial port was quite easy to do
- Most of the work was deciding how to parallelise the kernel
- Tracing exposed performance issues
- Prefetch was useful for improving initial performance
- … but still a very long way from optimized performance


intel