Using OpenMP threads in NIMROD

Jacob King

Ben Jamroz, Tom Jenkins, Scott Kruger

Tech-X Corp.

Why add threaded parallelism to NIMROD?

- Threads use shared memory – may help alleviate memory constraints.
- If each MPI process controls N threads, the number of cores participating in MPI calls are reduced by a factor of N.
  - Unless there are threaded MPI calls.
- Potentially better scaling for large jobs with reduced MPI calls.

Strategy: Use OpenMP threads to parallelize rblock loops.

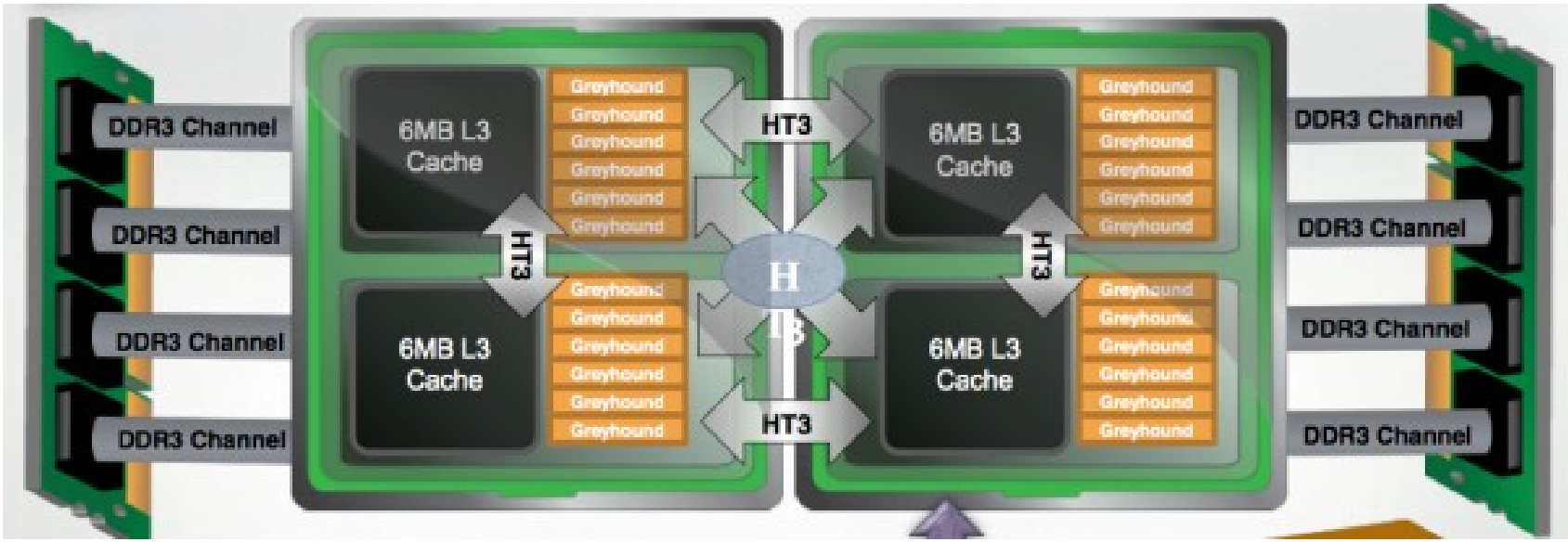
- For example, in finite_element.F90, get_rhs

```
!---------------------------------------------------------------------
!     evaluate the right-hand side for the appropriate integrand.
!---------------------------------------------------------------------
      CALL timer(timestart)
      !$omp parallel do default(shared) private(ibl) &
      !$omp& schedule(static,1)
      DO ibl=1,nrbl
        CALL rblock_get_comp_rhs_q(rb(ibl),rhsdum(ibl),integrand)
      ENDDO
      !$omp end parallel do
      DO ibl=nrbl+1,nbl
        CALL tblock_get_comp_rhs_q(tb(ibl),rhsdum(ibl),integrand)
      ENDDO
      CALL timer(timeend)
      time_rhs = time_rhs + timeend - timestart
```

- Timers have been moved outside of threaded regions (except the FFT timer which is placed in a master region).
- Thread safety must be checked - “grep -i SAVE *{.f90,.F90}”.

User must be more aware of underlying hardware with threaded parallelism.

- Nonuniform memory access adds complications.
- For example, on Hopper no more than 6 threads are recommended.



Which routines have been threaded?

- matrix_create
- integration in get_rhs
- static condensation routines
- other more minor loops (mostly in field_comps, iter_*)

- What is not threaded?
  - SuperLU calls appear to be the most significant.
  - Some threading possible with threaded BLAS library.
  - qp_update, many field_comps routines, anything in closures/

- As not all routines are threaded, consider head to head comparisons with MPI-only parallelism as a proof of concept.

Two options are available for mpi_alltoall calls in FFTs.

1) Thread-safe mpi_alltoall calls with a unique mpi communicator based on the block id.

2) All threads pack data into a single array, and only the master thread calls mpi_alltoall.

The better option will depend on the bandwidth and latency of the interconnects.

Cray Gemini interconnects have very low latency.

Option 1) MPI calls in FFTs use a thread-safe MPI communicator handle.

- Optional ibl argument is passed which identifies correct MPI_comm from comm_mode_blk(ibl).
- Integrand routines determine ibl from ibl=global2local(rb%id).
- Threads reduce MPI calls from seaming operations, but not in FFT calls in RHS integrands.

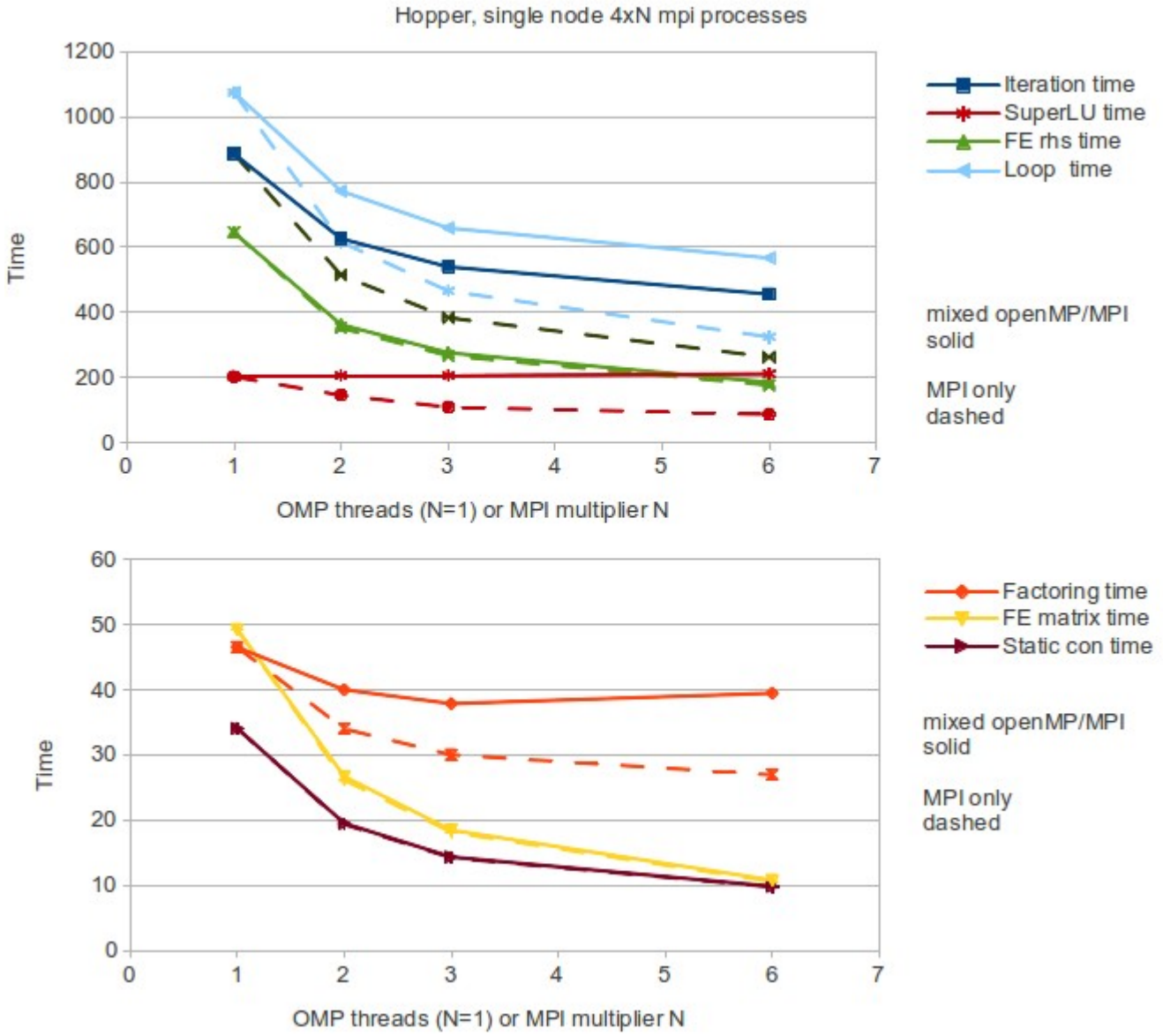
Option 2) Master thread makes all mpi_alltoall calls in FFTs.

- MPI calls are reduced by nthreads.
- Could lead to gains on high latency machines, and/or scaling gains.
- Gains potentially offset by time taken by threads to load and unload a shared array.

Test case: Tearing mode case provided by Tom Jenkins.

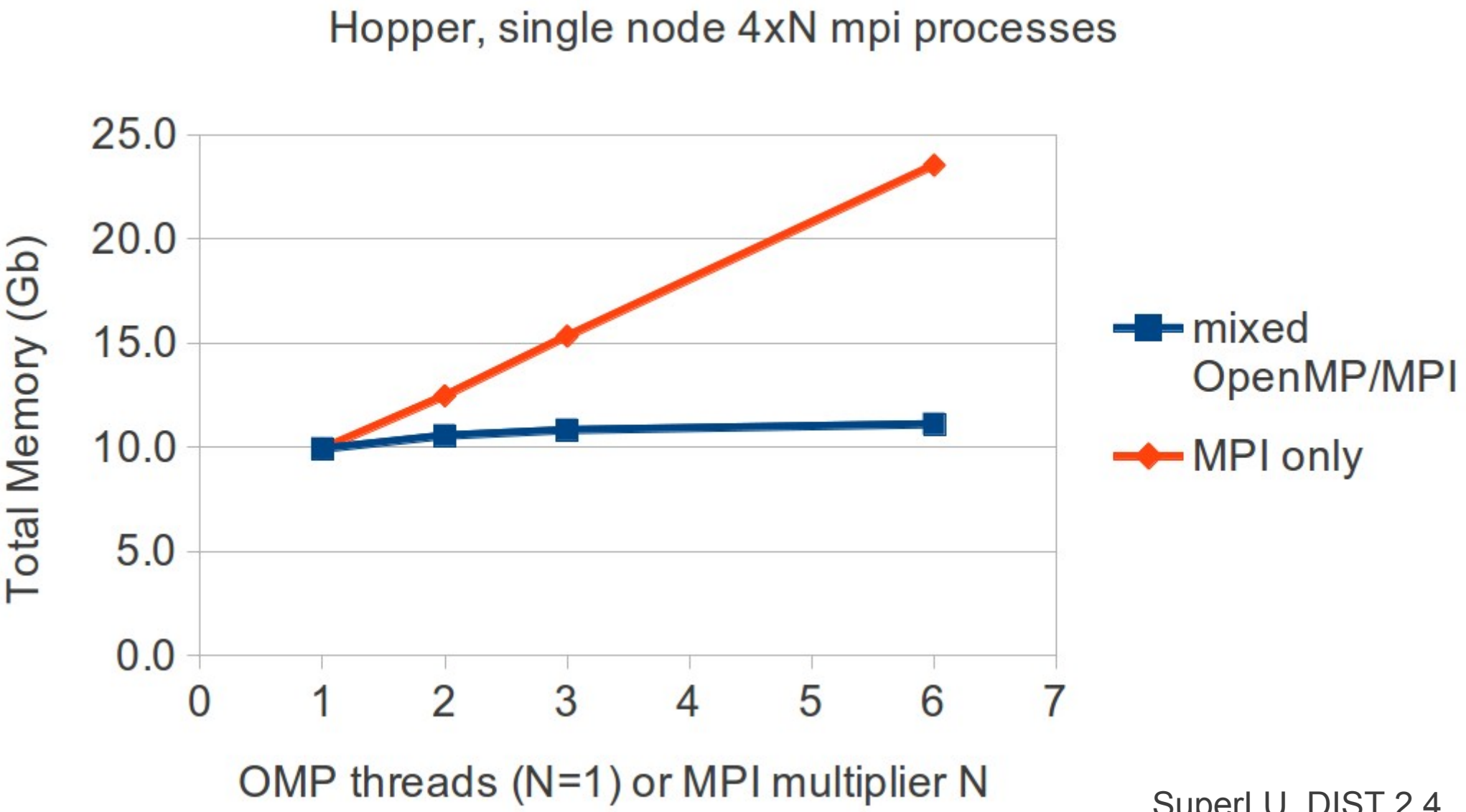
- Nonlinear case, but in linear stage.
- Uses anisotropic thermal conduction.
- Preconditioning matricies are recomputed during approximately ½ the time steps.
- B/n solves take 2-4 iterations, T takes 5-7 its., and V takes up to 2-30 its.
- 48x64 pd4 mesh split into 12x32 blocks with 22 modes decomposed into 22 layers.
- With strong scaling this case is fully decomposed at 8448 processors (one 4x2 FE block per proc.).

Intranode strong scaling: OpenMP threads vs. MPI



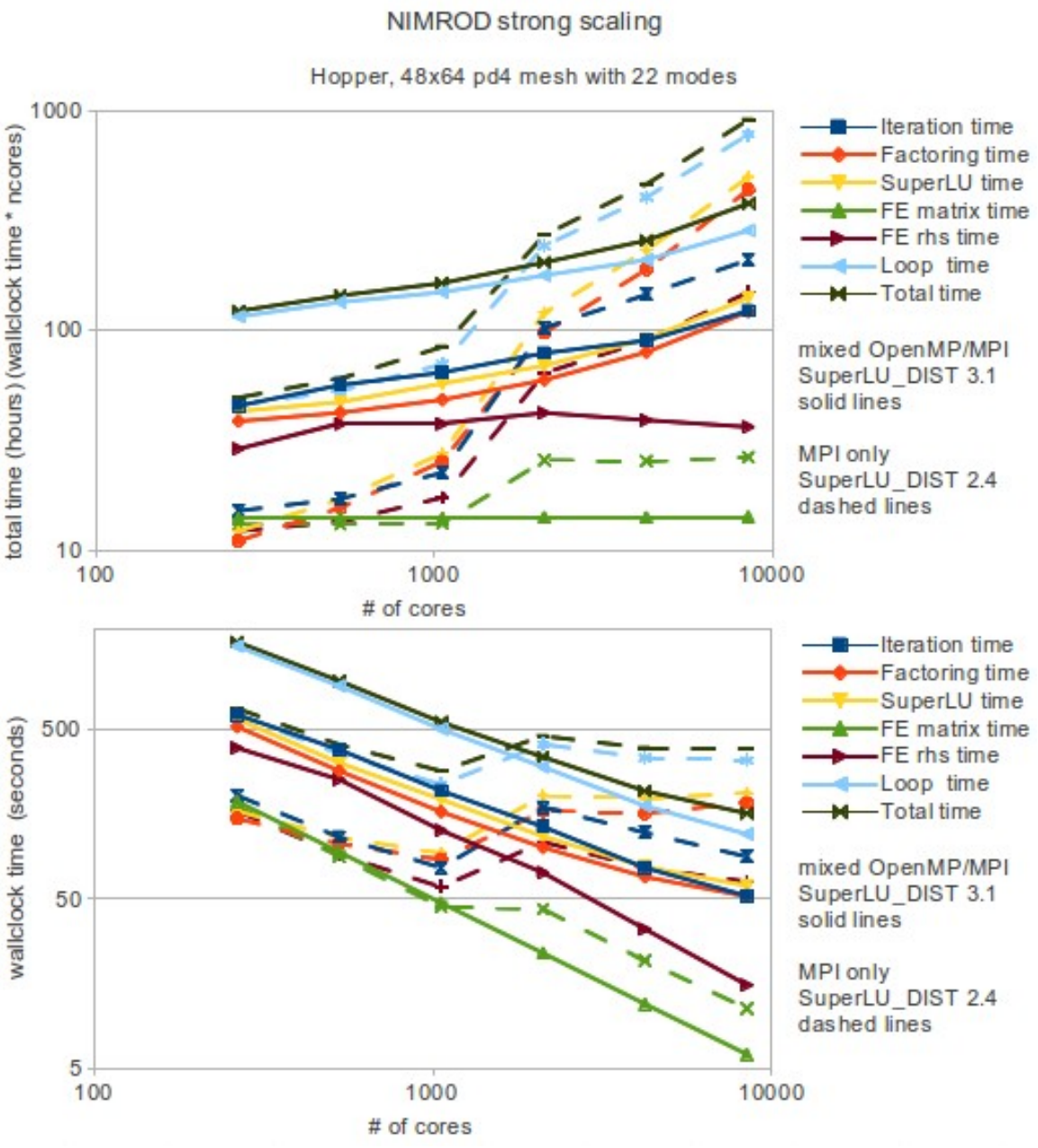
- Scaling is not perfect - likely from cache effects.

Intranode strong scaling: OpenMP threads provide memory gains.



SuperLU_DIST 2.4

Internode strong scaling: OpenMP threads vs. MPI



- In MPI-only cases with 2112 cpus and greater, the solver fails to converge and the result differs.
- A fairer comparison would reduce the cores in the SuperLU processor grid.

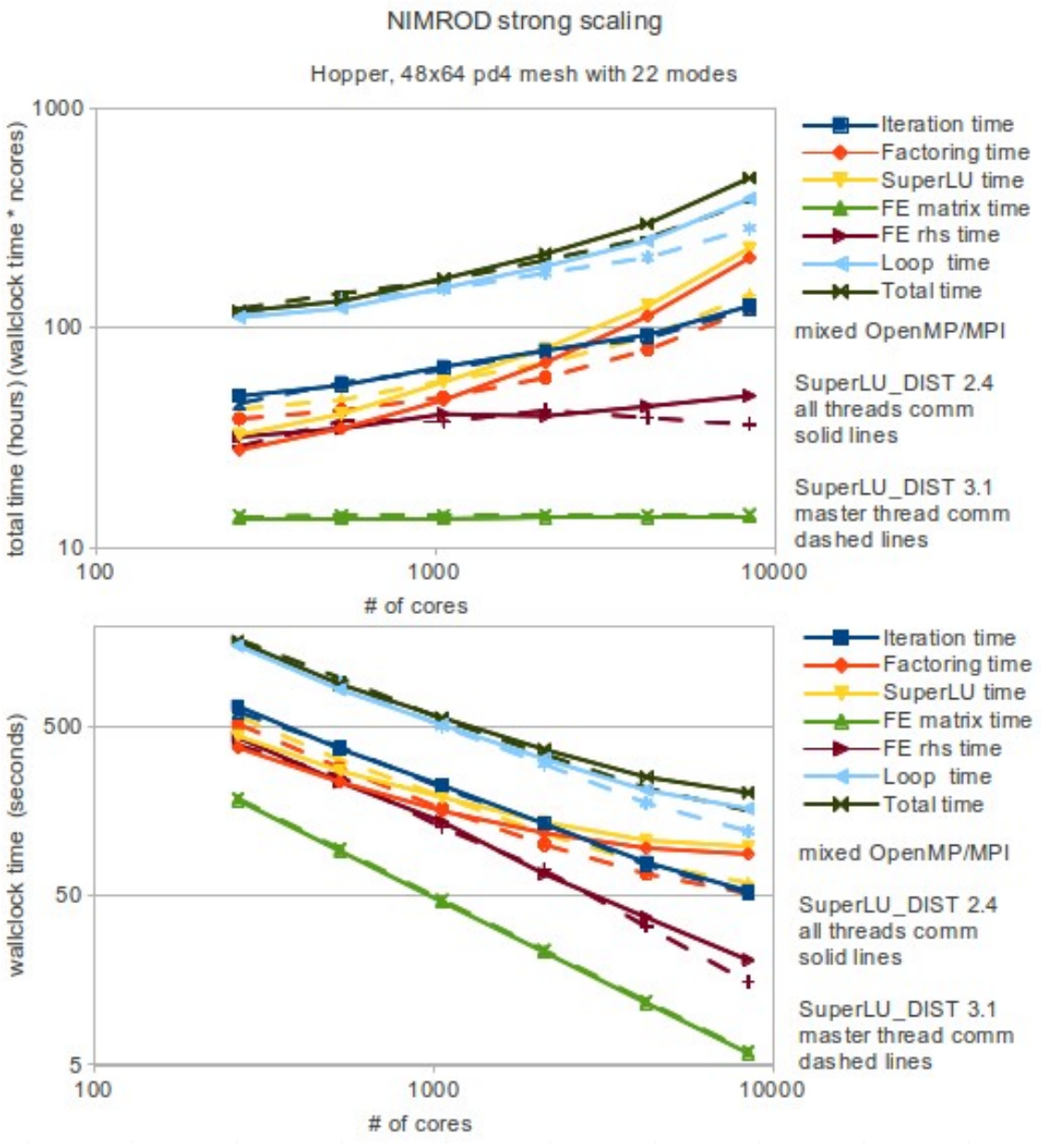
Why are the OpenMP cases slower at a small number of nodes?

- For this case:
  - ~ 25% - SuperLU_DIST is not threaded.
  - ~ 25% - Threaded/Master communication is slower than expected in FFTs.
  - ~ 50% - Unthreaded regions that are parallelized with MPI communication (May require some restructuring of the block loops to efficiently parallelize some of the remaining regions).

We are using SuperLU_DIST 3.1 for the OpenMP cases.

- Cases with 24 MPI processes per node do not converge (GMRES solver is poorly preconditioned) with SuperLU_DIST 3.1.
- SuperLU_DIST 3.1 scales better than SuperLU_DIST 2.4, but is slower on smaller sized jobs (Next slide).

Internode strong scaling: SuperLU_DIST version



- SuperLU_DIST 3.1 scales better than SuperLU_DIST 2.4, but is slower on smaller sized jobs (Next slide).

We (Ben Jamroz) have also interfaced with the Pastix solver.

- The solver has the ability to run with threads and potentially GPU accelleration.
- The solver is working on local Tech-X clusters, however builds on Hopper do not run (Internal Pastix errors).
- Comparison with SuperLU_DIST and scaling with Pastix is on going.

Summary

- Using threads adds an extra layer of parallelism to NIMROD.
- Threading reduces memory usage relative to MPI.
- Even if performance does not meet or exceed MPI parallelism, threads can help with memory limited jobs.
- Many jobs are memory limited on Hopper, and threading could accellerate these jobs – even with current limited thread implementation.