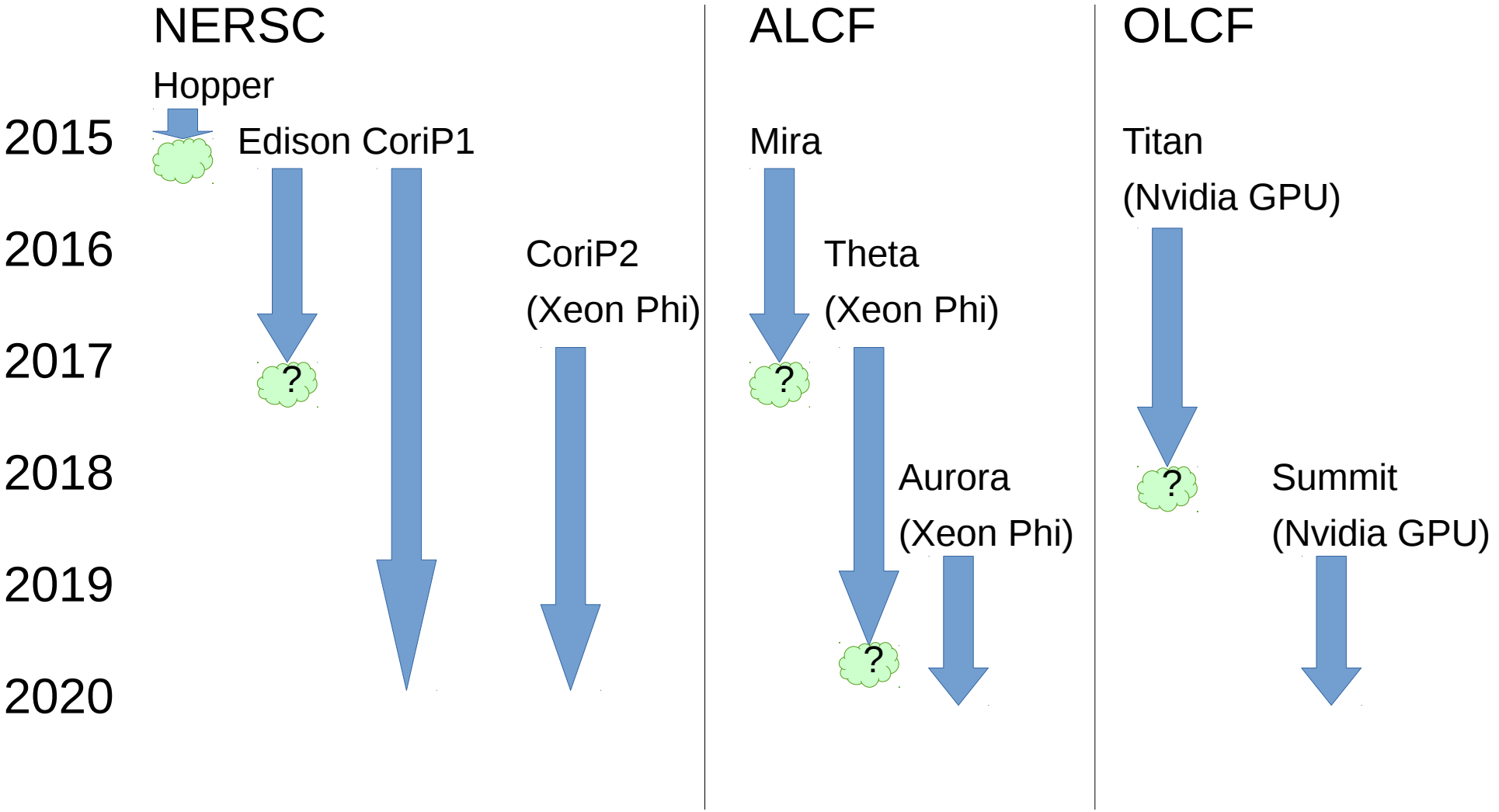
Notes on running NIMROD at ALCF (Mira BGQ)

Jacob King

The computing facility landscape is changing – using threads with Mira will help prepare us for Phi machines.



Mira is more like a Xeon Phi machine than Edison

Comparison of Mira to Xeon Phi (single node)

- Mira:
  - 16 cores
  - 4x oversubscribe
  - => 64 processes/node max
  - 16 Gb memory
- Xeon Phi (Knight's landing)
  - 72 cores
  - 4x oversubscribe
  - => 288 processes/node max
  - Up 384 Gb memory

There are two relevant machines: Mira and Cetus

- See http://www.alcf.anl.gov/user-guides/mira-cetus-vesta
- We do not have access to Vesta.
- Cetus → small jobs for testing (4096 nodes max)
  - Good to debug performance here.
- Mira → production runs (e.g. 1024+ node jobs)

Setup on Mira

- See http://www.alcf.anl.gov/user-guides/overview-how-compile-and-link
- I set the following in .soft
  - +mpiwrapper-xl
  - +cmake
  - @default
- BGQ cmake is required (do not try to build it)
- I plan to switch to +mpiwrapper-xl.ndebug for production runs.

Building nimdevel on Mira

- I've tried to make this as painless as possible.
  - There's still a bit of pain.
- Compilation is long – expect it to take a day.
  - I will install a trunk version in the project directory, please use it if you don't need local modifications.
- Use IBM's XL compiler suite.
- Build line:
  - ./mknimall.sh -N -F -V -y -Y -j 3 -m bgq.xlc -b ${HOME}/build-xlc -i ${HOME}/software
  - Builds with FFTW, MUMPS 5.0, SuperLU_DIST 4.0 and OpenMP threading support.
  - Important kludge: after nimdevel-par build launches:
    - Edit: ${HOME}/build-xlc/nimdevel/par/nimcore/CmakeFiles/nimcore.dir/flags.make
    - Remove: -qsmp=omp -qsmp=stackcheck -O3 -qtune=qp -qarch=qp -qnooptdebug -qreport -qessl
    - Replace: -O0
    - Reduces optimization on dump_par.f90 to avoid an internal compiler error.
  - Sorry about the kludge, I haven't been able to get this to work with Cmake.

Direct solver notes

- SuperLU_DIST 4.0
  - OpenMP threads used during factorization
  - No threading during back solves
  - OpenMP threading used during BLAS calls
- Mumps 5.0.0
  - Some OpenMP threading
  - OpenMP threading used during BLAS calls
- Low priority TODO: build and test performance with SuperLU_DIST 3.3 (requires complete rebuild).

Current compiler flags are somewhat aggressive

- Optimization: -O3
  - -O4 caused trouble
- -qsmp=omp -qsmp=stackcheck
  - OpenMP threading and issue a warning if a stacksize limit is breached (limit set by stackcheck in XLSMPOPTS environment variable).
- -qtune=qp -qarch=qp
  - Generate BGQ optimizations
- -qreport
  - Report optimization in *.lst files in the build directory.
  - Good to check to ensure SIMD vector instruction usage.
- -qessl
  - Replace code with essl (essentially BLAS) routines if possible.

Running jobs

- Copy submit.sh and runjob.sh from the project directory (/projects/Plasma_Dynamics/)
- Edit runjob.sh variables as desired
- Execute submit.sh
- See http://www.alcf.anl.gov/user-guides/queueing-running-jobs

runjob.sh is the job script.

Modify these

This example runs 8 MPI processes with 8 OpenMP threads each (64 total processes per node)

```
#!/bin/bash

NNODES=1
CORES_PER_NODE=64
NTHREADS=8
STACKSIZE=128
BG_THREADLAYOUT=1
NIMEXEC=/gpfs/mira-home/jking/software/nimdevel/bin/nimrod

#------------------------------------------------------------------------------
# No need to edit below this line
#------------------------------------------------------------------------------

export OMP_NUM_THREADS=${NTHREADS}
export OMP_STACKSIZE="${STACKSIZE}M"
SSBYTES=`expr ${STACKSIZE} \* 1024`
SSBYTES=`expr ${SSBYTES} \* 1024`
SCBYTES=`expr ${STACKSIZE} / 2 \* 1024 \* 1024`
echo "setting stack size to ${STACKSIZE}MB (${SSBYTES} bytes)"
export XLSMPOPTS="stack=${SSBYTES}:stackcheck=${SCBYTES}:parthds=${NTHREADS}"
echo "XLSMPOPTS = ${XLSMPOPTS}"

RANKS_PER_NODE=`expr ${CORES_PER_NODE} / ${NTHREADS}`
NRANKS=`expr ${RANKS_PER_NODE} \* ${COBALT_JOBSIZE}`
echo "running on ${COBALT_JOBSIZE} nodes with ${RANKS_PER_NODE} MPI cores/node
and ${NTHREADS} threads"

echo "runjob --np $NRANKS -p $RANKS_PER_NODE --exp-env OMP_NUM_THREADS OMP_STAC
KSIZE XLSMPOPTS --block $COBALT_PARTNAME --verbose=INFO : $NIMEXEC"

runjob --np $NRANKS -p $RANKS_PER_NODE --exp-env OMP_NUM_THREADS OMP_STACKSIZE
XLSMPOPTS --block $COBALT_PARTNAME --verbose=INFO : $NIMEXEC
```

submit.sh is a script to submit.

```
#!/bin/bash
# Execute in directory with RUNSCRIPT to run a job

RUNSCRIPT=runjob.sh
NODES=`grep NNODES ${RUNSCRIPT} | sed 's/NNODES=//'`
qsub -A Plasma_Dynamics -t 20 -n ${NODES} --mode script ${RUNSCRIPT}
```

- You do not have to modify this script.
- Grep command ensures consistency with runjob.sh.

Machine Partitions

- From https://www.alcf.anl.gov/user-guides/machine-partitions
- In the prod-capability queue, partition sizes of 8192, 12288, 16384, 24576, 32768, and 49152 nodes are available. All partitions have a full torus network except for the 32768 partition that in the default queue is mesh in one dimension. The full torus network for the 32768 partition is available by the special queue: prod-32768-torus. The max runtime of jobs submitted to the prod-capability and the prod-32768-torus queues is 24 hours.
- The prod-short and prod-long queues support partition sizes of 512, 1024, 2048 and 4096 nodes. All partitions have a full torus network except for the 1024 partition that in the default queue is mesh in one dimension. The full torus network for the 1024 partition is available by the special queue: prod-1024-torus. The max runtime of jobs submitted to the prod-short is 6 hours, compared to 12 hours for jobs submitted to the prod-long and prod-1024-torus queues.

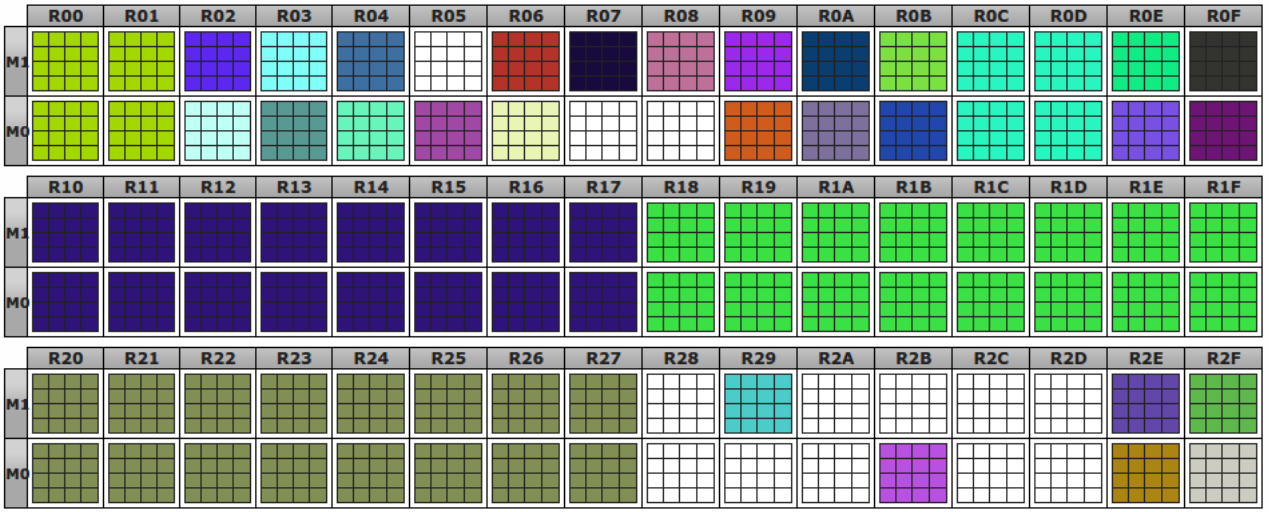
Use the web interface to view the queue

- http://status.alcf.anl.gov/mira/activity
- http://status.alcf.anl.gov/cetus/activity
- It's a nice interface to check without having to log in.

Argonne NATIONAL LABORATORY | Leadership Computing Facility

Mira Activity

Home | Mira | Activity



Each (small) block is 32 nodes. The minimum partition size is 128 nodes (one row; 4 blocks), see partitions slide. A rack is 1024 nodes.

Running Jobs | Queued Jobs | Reservations

Total Running Jobs: 32

Job Id	Project	Run Time	Walltime	Location	Queue	Nodes	Mode
505212	LiquidPhenom	16:51:07	1d 00:00:00	MIR-04000-37FF1-8192	prod-capability	8192	script
506269	SkySurvey	06:41:20	12:00:00	MIR-00000-333F1-2048	prod-long	2048	c4
505482	StructuralBiology	06:37:17	12:00:00	MIR-00C00-33F31-512	prod-long	512	script
507775	REBatteries	04:02:55	05:58:00	MIR-00400-33731-512	prod-short	256	c4
507777	REBatteries	03:51:02	05:58:00	MIR-00800-33B31-512	prod-short	256	c4
507778	REBatteries	03:28:40	05:58:00	MIR-40800-73B31-512	prod-short	256	c4
507779	REBatteries	03:07:58	05:58:00	MIR-00840-33B71-512	prod-short	256	c4
507841	REBatteries	02:34:31	05:58:00	MIR-00CC0-33FF1-512	prod-short	256	c4
507842	REBatteries	02:30:27	05:58:00	MIR-40840-73B71-512	prod-short	256	c4

How this impacts NIMROD:

- Partition sizes correspond to using cpu counts that are a power of 2:
  - e.g. prod-long queues support partition sizes of 512 (8192), 1024 (16384), 2048 (32768) and 4096 (65536) nodes (cores).
- Use FFTW transforms to set nmodes to a power of 2 (or close).
  - Set nphi = 1, 4, 9, 21, 45, 90, 189, 378 etc..
  - Reminder: small prime factors in nphi make the fft algorithm more efficient.
- Set nxbl and nybl to powers of 2.
  - For example: nphi=90; nmodes=31; mx=64; nxbl=16; my=512; nybl=128 (4x4 element blocks) is my initial target EHO job.
  - This initial job would run on 1024 nodes. Ultimately, I may increase resolution to nphi=189; nmodes=64 and mx=128; nxbl=32 to run on 4096 nodes.
  - Reminder: the Mira allocation is for large production jobs. Look at the queue and note similar usage by other projects. NERSC allocations are more appropriate for smaller jobs.

Mira is an SMP machine

- From https://www.alcf.anl.gov/user-guides/how-manage-threading
- Like Blue Gene/P, Blue Gene/Q is a true SMP (symmetric multiprocessor) , as opposed to a NUMA machine . NUMA is common in multisocket or multidie Intel or AMD (Advanced Micro Devices, Inc.) systems. The practical consequence of this is that thread scaling is possible across the entire node; on a NUMA system, it is usually observed that thread scaling beyond a NUMA domain is challenging. As a result of the lack of NUMA, careful memory allocation and placement through the use of Linux's first-touch policy is not necessary. All hardware threads see all the main memory equally and thus there is no performance hit associated with having different threads initialize and compute with the same memory.

OpenMP threads improve memory flexibility on Mira

- NIMROD has balanced memory usage among cores and can run either MPI-only or mixed MPI-OpenMP parallism equally efficient, but SuperLU_DIST (or MUMPS) does not and may require threading to scale.
- From http://www.redbooks.ibm.com/redbooks/pdfs/sg247948.pdf
- When submitting a job to the Blue Gene/Q system with the runjob command or another job scheduler command, it is important to decide how many processes or tasks to run on each node. This decision significantly impacts performance and the memory that is allocated to each process, for example:
  - Running 16 processes per node divides the total number of cores and the total physical memory into 16ths. Thus, one core and roughly 1 GB is available to each task.
  - Running one process with 16 threads might yield equivalent performance but causes no subdivision of the memory layout.

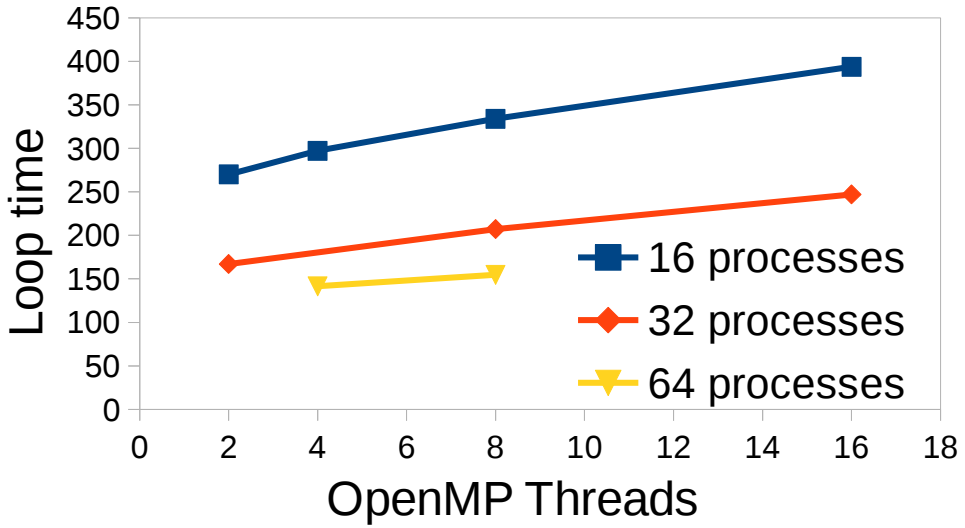
Peak theoretical performance uses 64 processes per node.

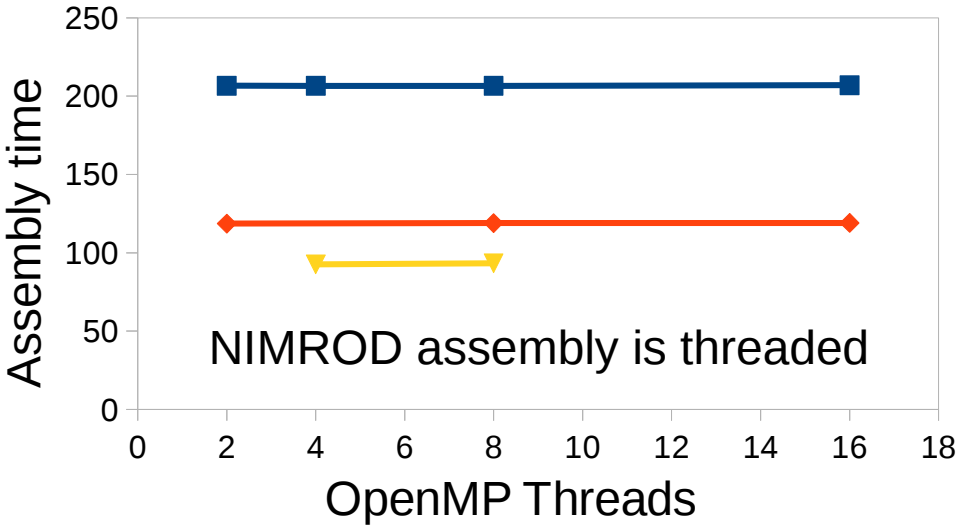
- Reminder: there are 16 physical cores with 4 hyper-threads per core.
- From http://www.redbooks.ibm.com/redbooks/pdfs/sg247948.pdf
- The best configuration of processes per node depends on the type of application, the memory requirement, and the parallel paradigm that is implemented. There might be several options for applications that use a hybrid paradigm with both MPI and OpenMP or pthreads, depending on the memory footprint. Hybrid applications that support a high degree of threading might work well with a single process per node, but scenarios with 2, 4, 8, or 16 processes per node are more common. For single-threaded applications, the memory requirement per process is the main consideration. If possible, use all 16 of the cores with 16, 32, or 64 processes per node.
- One trade off to consider is that each additional process that is running on a node has a fixed amount of overhead. Overhead consists of a replicated data segment (for example, the global storage for the process), storage for the main stack, and storage for the heap

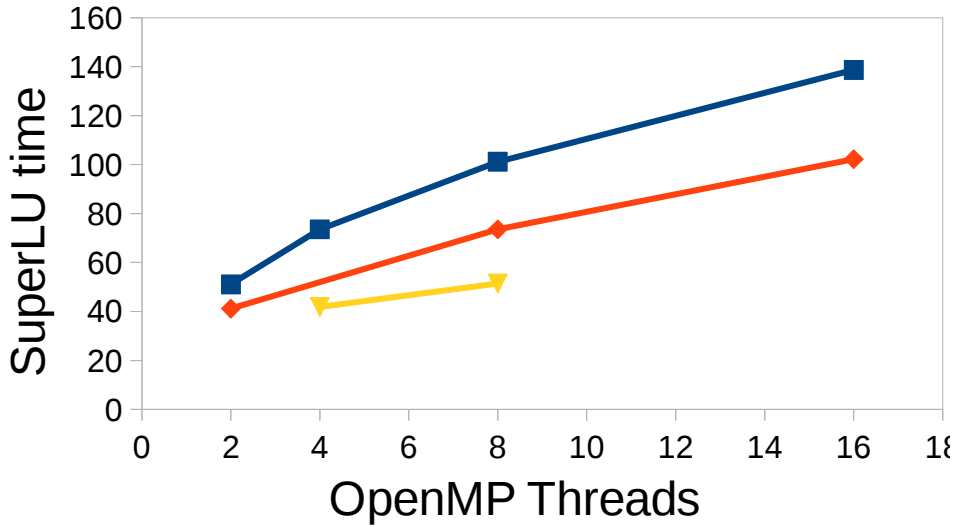
Task/Thread placement is controlled with BG_THREADLAYOUT

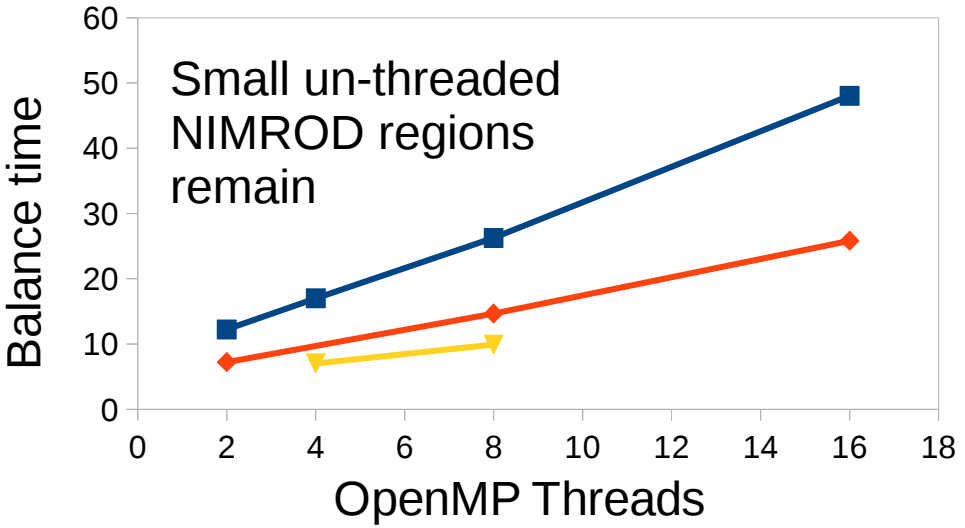
- From http://www.redbooks.ibm.com/redbooks/pdfs/sg247948.pdf
- 3.11.1 Breadth-first assignment
  - Breadth-first is the default thread layout algorithm. This algorithm corresponds to BG_THREADLAYOUT = 1. With breadth-first assignment, the hardware thread-selection algorithm progresses across the cores that are defined within the process before selecting additional threads within a given core.
- 3.11.2 Depth-first assignment
  - This algorithm corresponds to BG_THREADLAYOUT = 2. With depth-first assignment, the hardware thread-selection algorithm progresses within each core before moving to another core defined within the process.

Single node performance: SuperLU_DIST 4.0



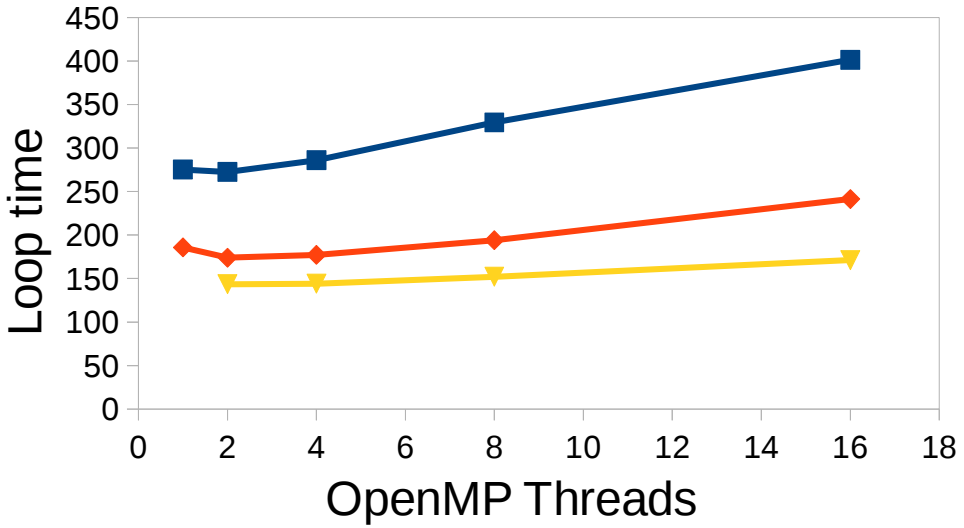


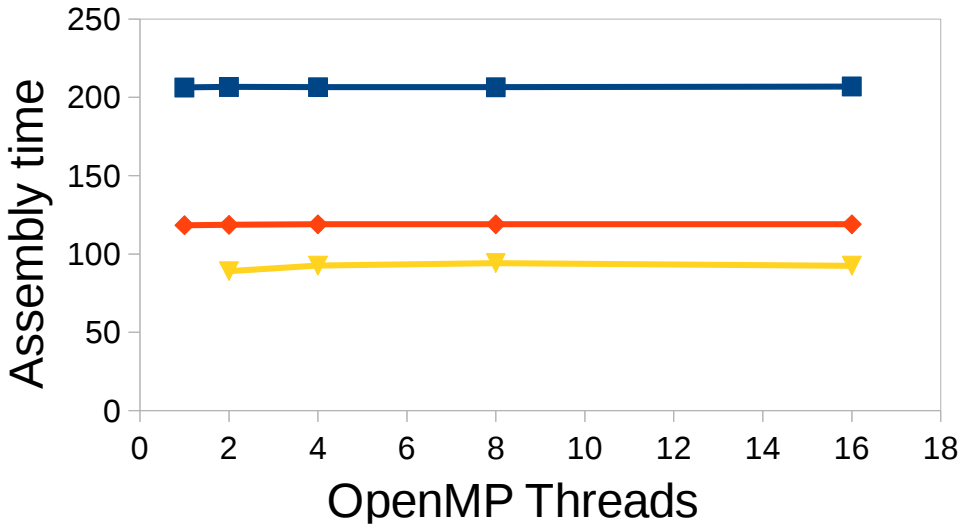


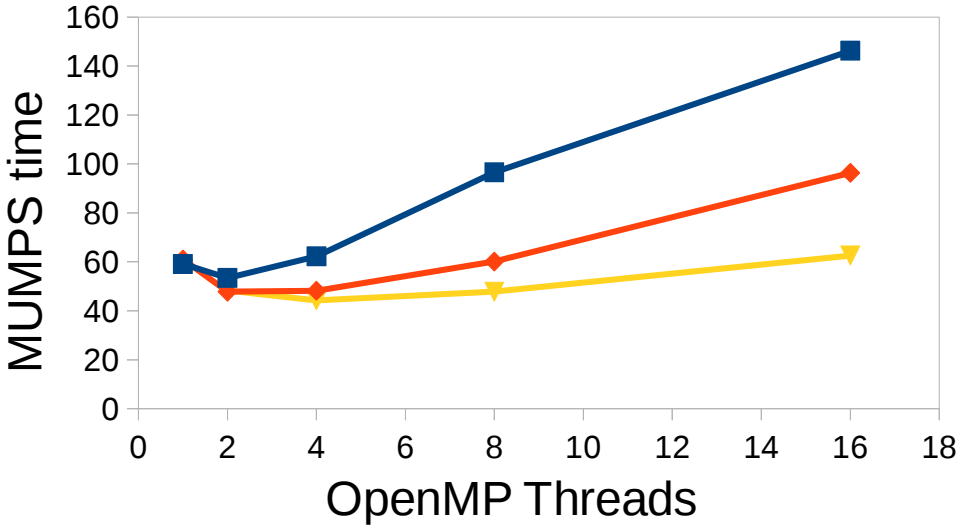


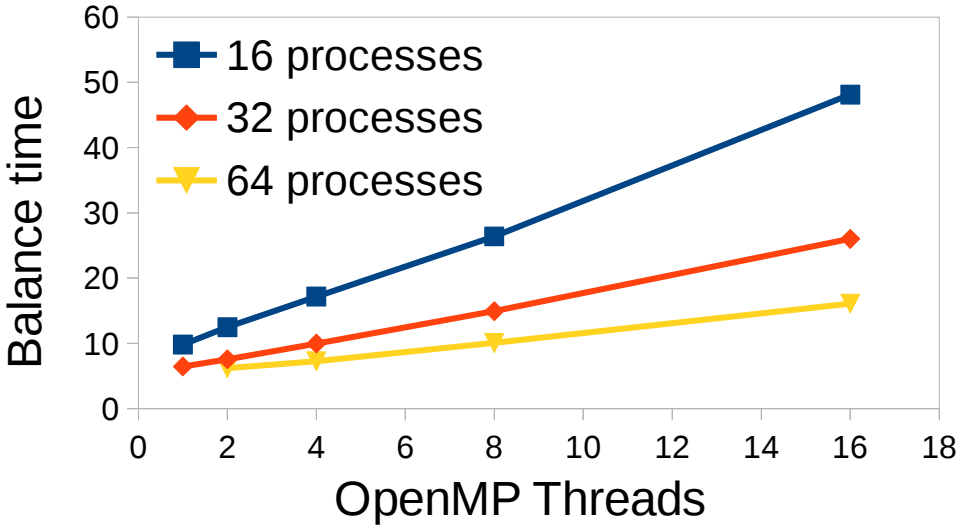
- 64 processes with 8 OpenMP threads is best mix of memory flexibility and performance
- mx=32; my=16; poly_degree=6; nmodes=1; nonlinear=T (2D EHO case)
- Assembly = FE_matrix + FE_vector + Static_con
- Balance = Loop - SuperLU - Assembly
- Cases with 1 thread → OOM error; other un-plotted cases hang in SLU factorization

Single node performance: MUMPS 5.0.0



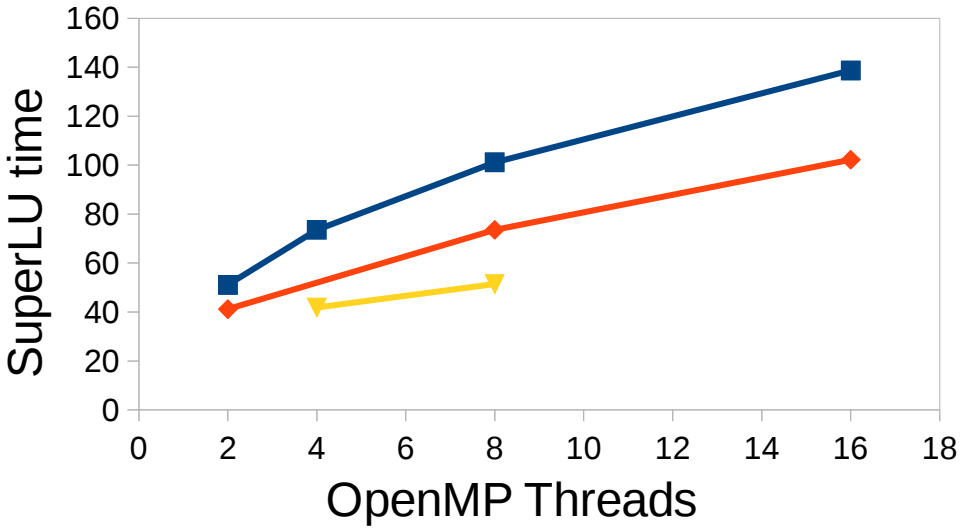


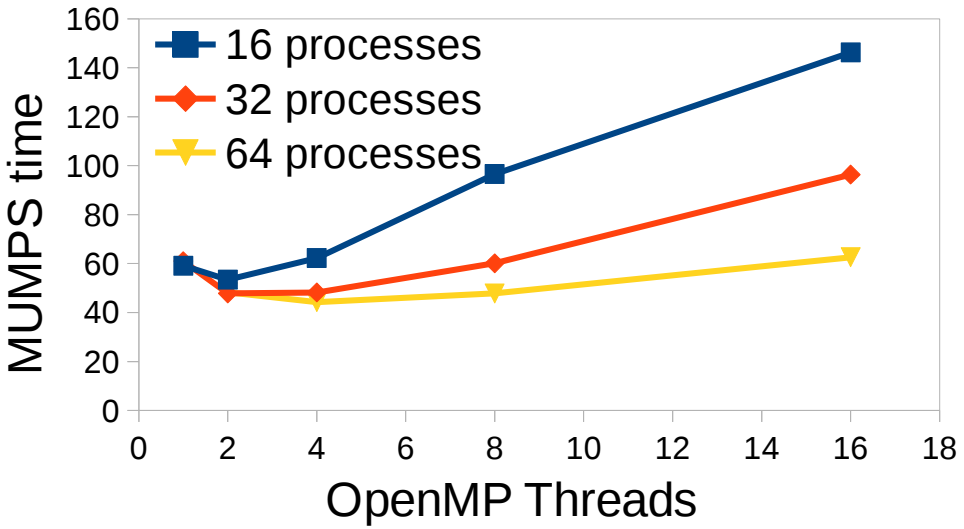




- 64 processes with 8/16 OpenMP threads is best mix of memory flexibility and performance
- mx=32; my=16; poly_degree=6; nmodes=1; nonlinear=T (2D EHO case)
- Assembly = FE_matrix + FE_vector + Static_con
- Balance = Loop - SuperLU - Assembly
- 64 processes with 1 thread → OOM error

Comparison of SuperLU_DIST 4.0 and MUMPS 5.0.0





- Overall performance is comparable
- MUMPS runs for more cases and given this greater flexibility is probably preferable for now.

Comparison of single node performance to other machines

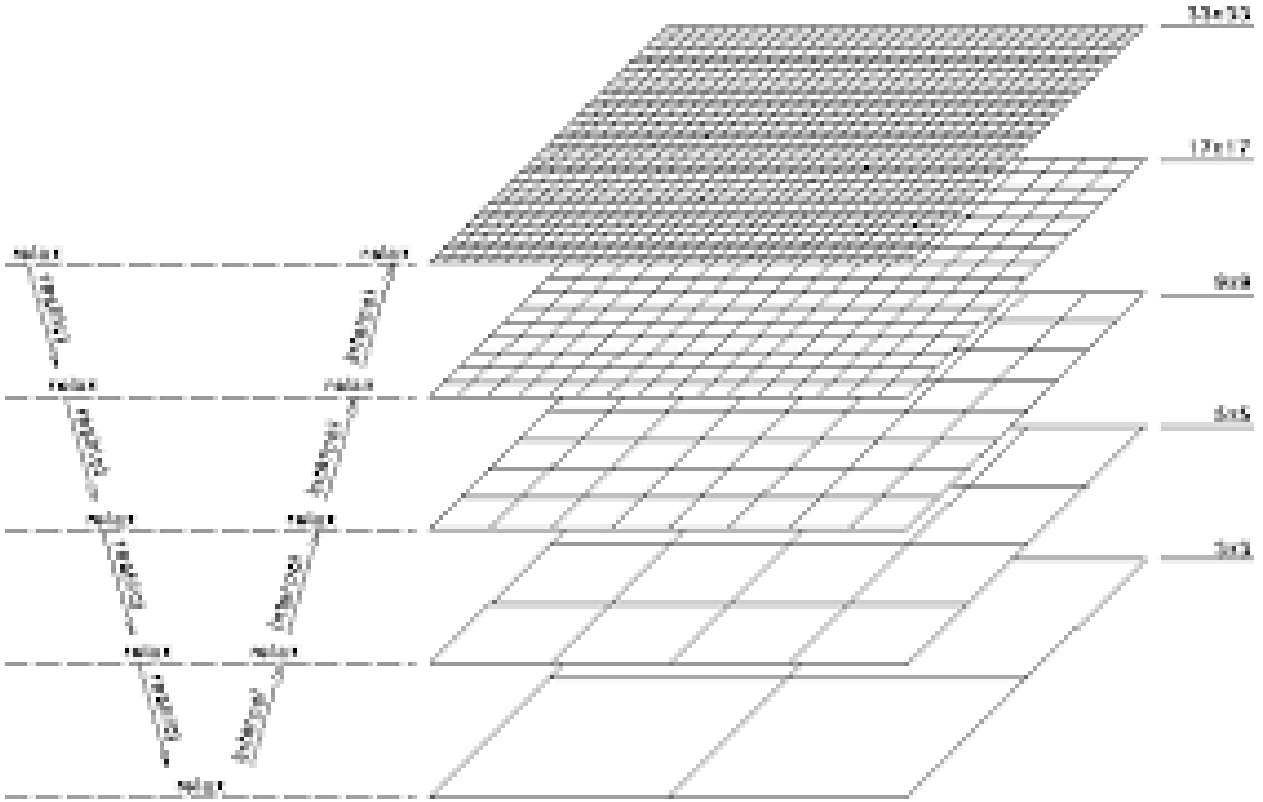
- Mira (BGQ PowerPC A2 – 16 cores ~ 55 W):
  - Loop 144s | Assembly 93s | Mumps 44s
- Intel Sandybridge i7 (4 cores ~ 45 W):
  - Loop 260s | Assembly 166s | SuperLU_DIST 39s
- Edison (Intel Ivybridge Xeon – 24 cores ~ 450 W):
  - Loop 24s | Assembly 18s | SuperLU_DIST 3s

- Edison is fast and power hungry!

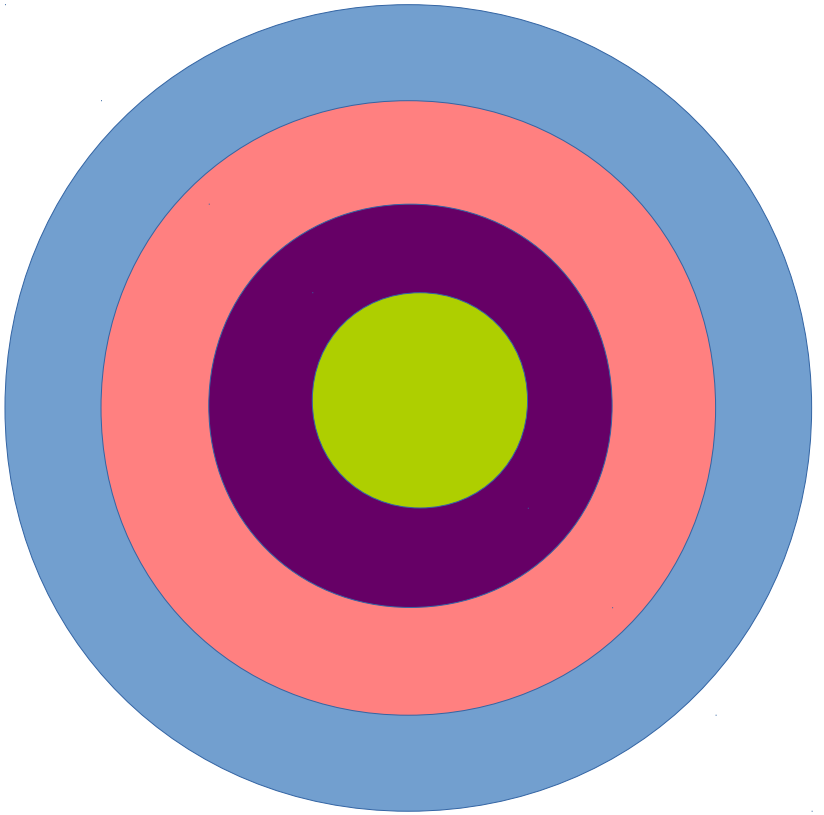
- With Edison charge factor of 2:
  - 1 NERSC MPP hour ~ 3 Mira core hours
    - => 40 Million Mira core hours ~ 13.3 NERSC MPP hours
  - But there's also memory issues!
  - Mira's low memory environment also means we need to run jobs on more nodes (at scale) so there's an additional factor.

Interface to Hypre is in progress

- Hypre is a collection of multigrid solvers
- Many many options: 70+ input options for BoomerAMG alone
- Current implementation is preliminary (2-4x slowdown vs direct solve with 2 cores)



Interface to Hypre is in progress



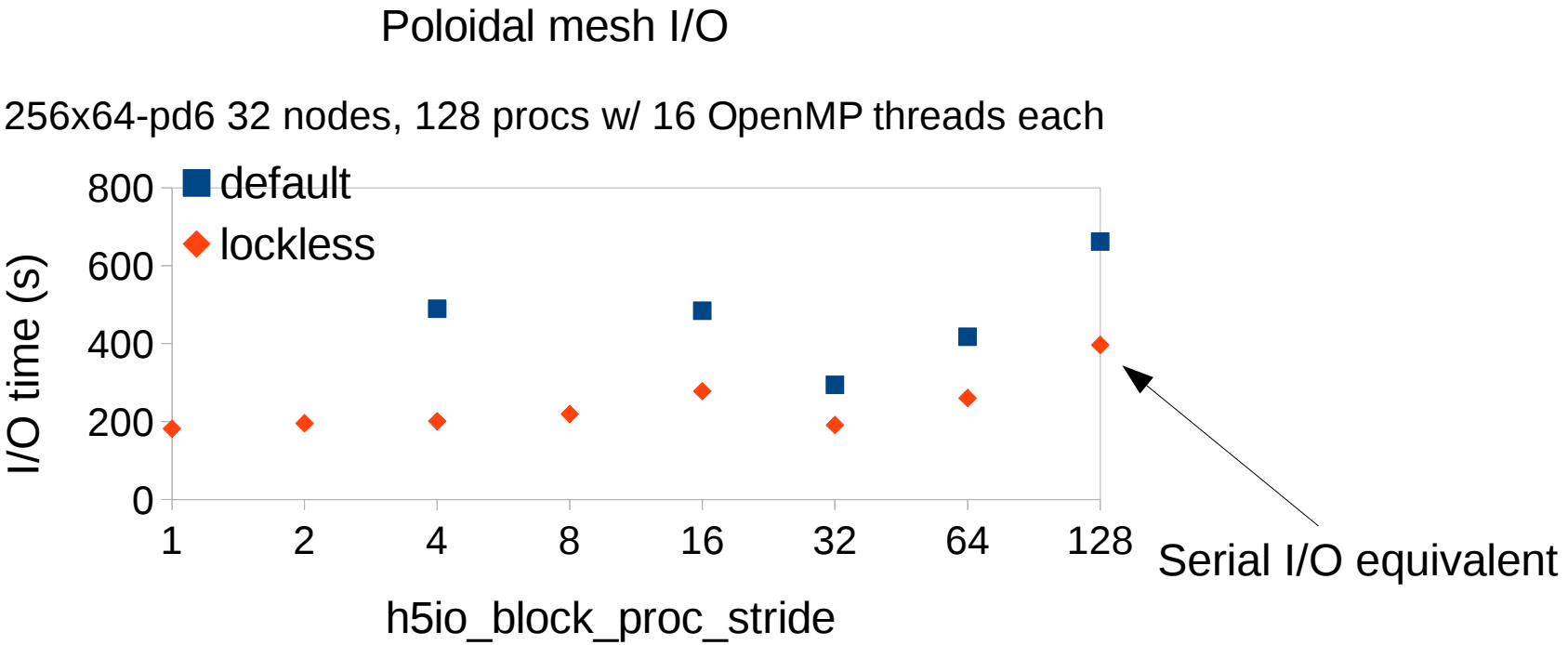
3D GMRES
- Tolerance is input tol

Preconditioner 2D GMRES (hypre)
- Setup to solve at reduced tolerance
- Efficient with 3D solves
- Smoother solves at leaves

Factorization is comparable to solve.
Implementation has nonlinear cases in mind.

Poloidal “leaves” seem to work best.

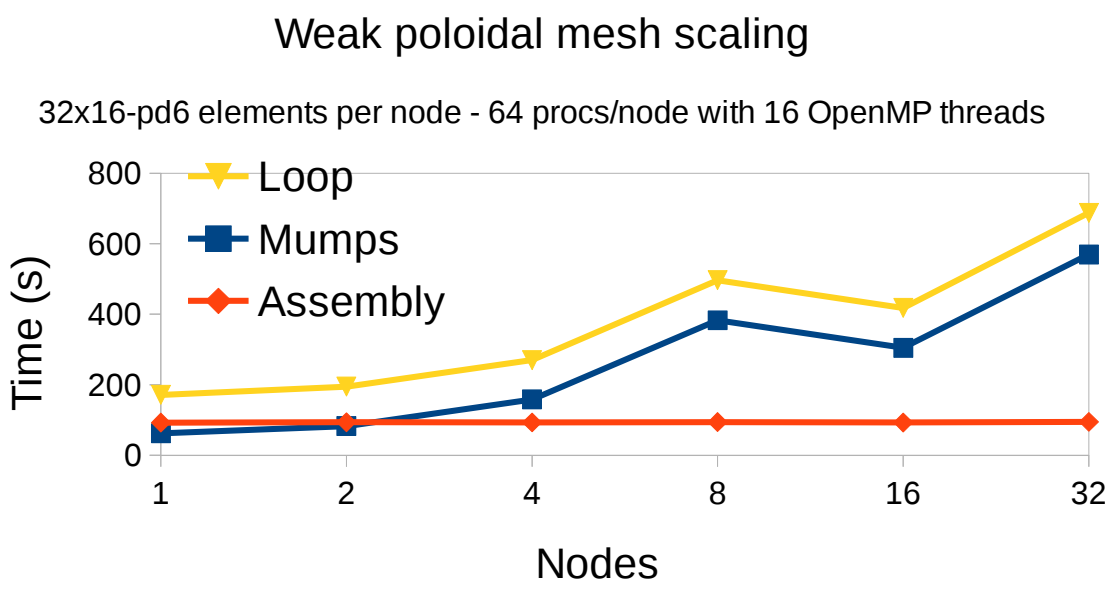
Tuning NIMROD HDF5 I/O

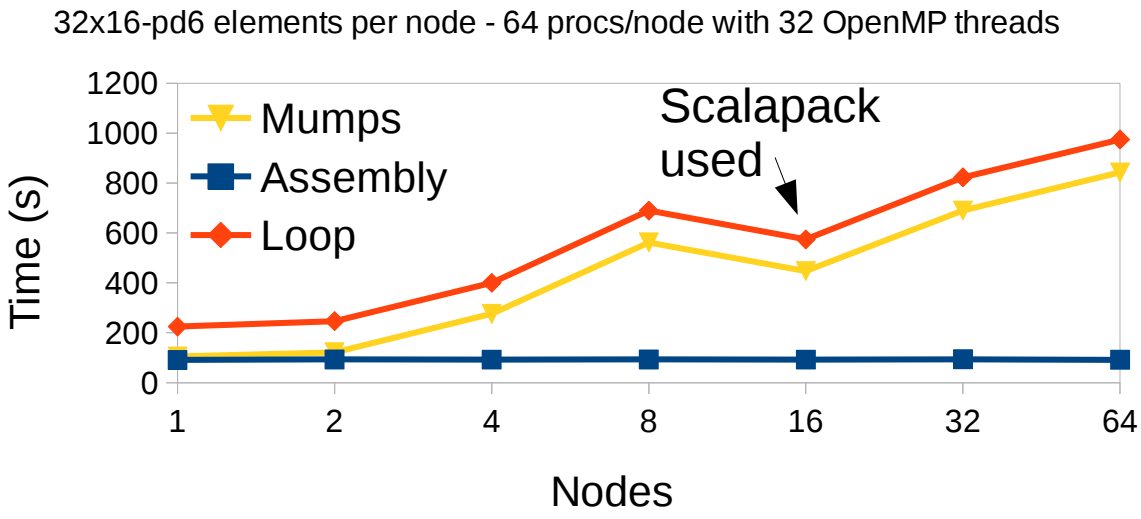


- Set h5dump=T and h5io_block_proc_stride=1
  - every MPI process participates in the write
- Lockless implies the environment variable BGLOCKLESSMPIO_F_TYPE=0x47504653
  - This is the so called 'magic number' (from the hdf5 documentation).
  - Open question: would this scaling be better if we could compile dump_par.F90 with full optimization?

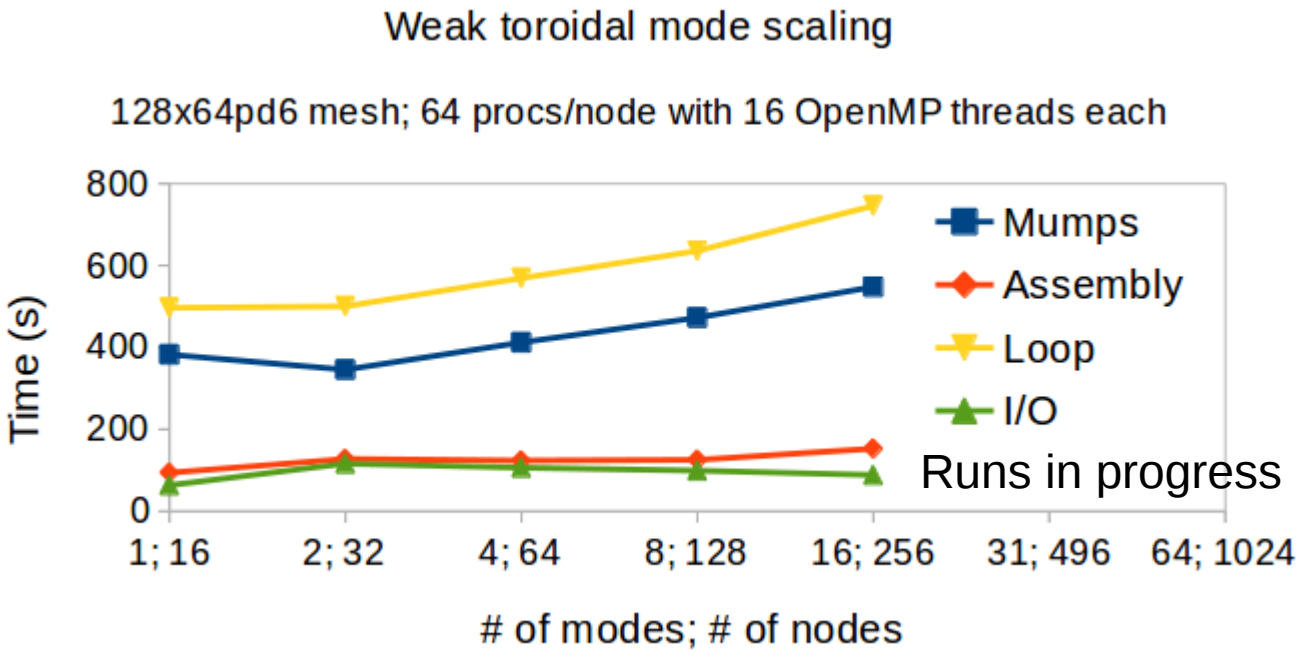
Scaling to a production run – poloidal mesh

- MUMPS scaling not perfect.
  - Approximately ~5x slowdown for a 64x larger problem.
  - Assembly time is small with good single node performance – makes hiding latency in solvers more challenging.
  - Scalapack is automatically used within MUMPS for cases with 16+ nodes.
  - Larger cases than shown result in OOM error.
- EHO production cases will use 32 to 64 nodes for each poloidal mesh.
  - Mesh sizes of 64x256pd6 to 64x512pd6, respectively.





Scaling to a production run – toroidal modes



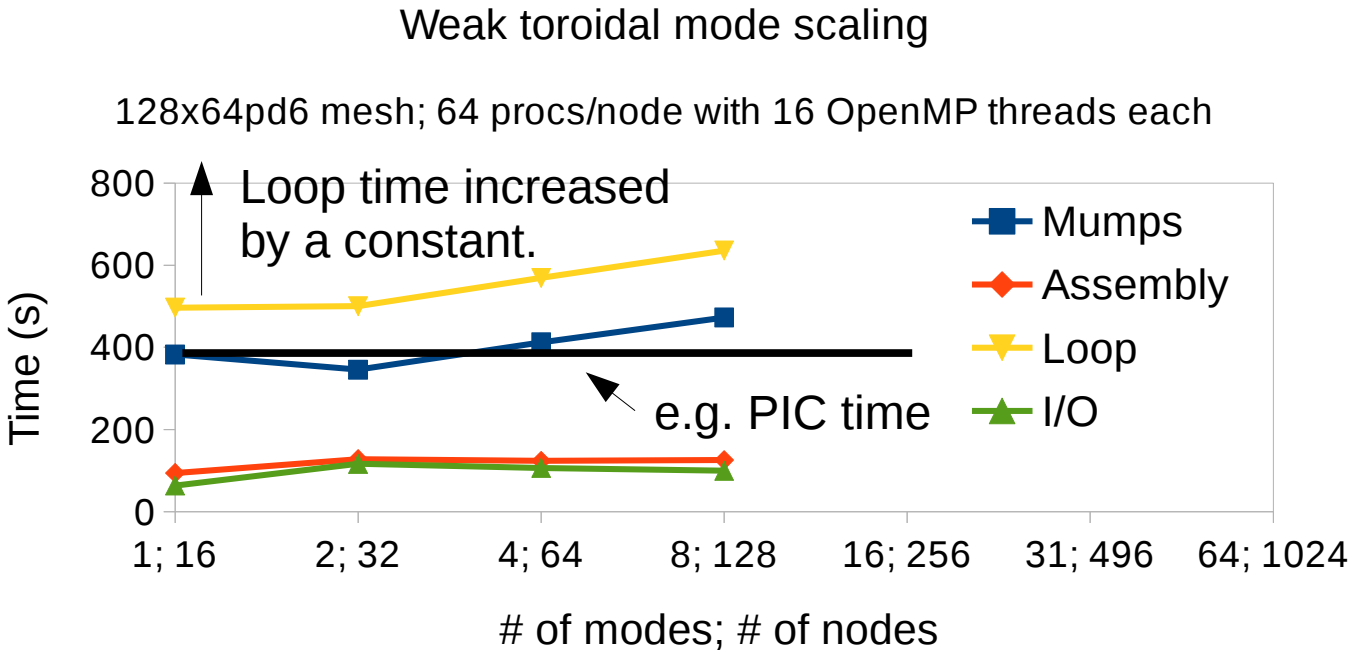
- This is not algorithmic scaling!
  - We are fundamentally solving a different problem for 1 → 2 modes. It becomes 3D.
  - The test case is initialized from a saturated EHO case. From 2 → 8 modes the EHO dynamics become barely resolved.
  - From 1 → 8 modes, the GMRES iteration counts for the various solves are changing.
  - After 8+ modes it is approximately algorithmic scaling.
- Never-the-less, excellent time-to-solution scaling is found through the 2D → 3D transition
- I/O time is a dump read and write (not included in loop time).
- Why 31 modes? See FFTW discussion.

Comparison of near-production performance to Edison XC30

- 128x64pd6x8modes (assume toroidal mode scaling is good)
- Edison (192 cores - 3) ~ 3600 W
  - Loop 488s | Assembly 361s | SuperLU_DIST 103s
- Mira (2048 cores) ~ 7040 W
  - Loop 636s | Assembly 126s | Mumps 473s
- Mira does very well with the NIMROD assembly.
  - If we can get better solver scaling, we can beat Edison.
  - Try HYPRE, Additive Schwartz with direct solves or something else.

Thoughts on PIC calculations

- Should be *very* efficient – must use threads if using map_mod.
- Global field storage size ~2 Gb for a large case:
  - mx x my x nmodes x pd^2 x vecsz(R,Z,B) x realsz (in bytes) x complex
  - 2 Gb = 128 x 128 x 64 x 5^2 x 5 x 8 x 2 / 1024^2
- Use 1-2 MPI cores per node.
- Better scaling with large number of particles.



Summary

- We are making progress for the future of many-core environments.
- Can run production, nonlinear EHO jobs.
  - Would like higher poloidal resolution.
- Hypre testing is promising, but still early.
  - Plan to finish hypre testing on Mira in the next two weeks.
  - Hypre/Many-core environments probably better for nonlinear NIMROD runs.
  - Linear runs that avoid multiple direct-solver factorizations are unlikely to be very efficient.