Performance and Scaling Information for the NIMROD Code

Carl Sovinec, University of Wisconsin-Madison

September 13, 2013

This report describes parallel scaling information for the NIMROD plasma simulation code (https://nimrodteam.org), which is used to model macroscopic dynamics of magnetized plasma in laboratory and natural systems. After an algorithmic summary, we review results on hybrid threaded/MPI parallelization from the Cray XE-6 Hopper system at NERSC. We then describe modifications that reduce memory requirements for distributed-memory algebraic solves.

The NIMROD code solves nonlinear initial-value partial differential equations (PDEs) that describe the evolution of plasma number density, momentum density, temperature, and magnetic field over three spatial dimensions. The fields are represented on a plane of 2D spectral finite elements with finite Fourier series for the third coordinate. Due to the mathematical stiffness of the problems of interest, each time-step requires the solution of large ill-conditioned matrices. The Fourier representation of the periodic coordinate leads to dense submatrices from the convolutions, and NIMROD uses matrix-free generalized minimal residual (GMRES) iteration with fast Fourier transforms (FFTs) and block-based preconditioning, where each block represents a distinct Fourier component. With the Fourier representation, diagonal blocks hold the largest matrix elements and are inverted with the parallel sparse direct solver, SuperLU_DIST.[1] This preconditioning is augmented by limited block-Gauss-Seidel-like steps for off-diagonal (in Fourier component) blocks with asynchronous communication overlapped with on-processor computation.

Last year, Jacob King of Tech-X Corporation implemented hybrid threading/MPI parallelization in NIMROD using the OpenMP library for shared-memory operations. He incorporated threading over several different parts of the computation: the finite-element construction of matrix elements and algebraic vectors, static condensation steps (limited cyclic reduction of matrix elements), and looping for computation of other coefficients. This is not comprehensive for NIMROD; threading is not applied in the preconditioning steps, for example, but it is sufficient to demonstrate some of the benefits of hybrid OpenMP/MPI parallelism. The directives for OpenMP also provide a step toward future improvement with processing accelerators via the OpenACC library.

Strong scaling from 264 to 8448 cores on Hopper is shown in Figure 1 for an RF-wave/MHD coupling computation with hybrid parallelization. These cases have a 48×64 quartic-polynomial finite-element mesh with 22 Fourier components, and for the largest run each processor owns a 4×2 section of the finite-element mesh with a single Fourier component. Time spent in NIMROD's time-advance-loop is shown along with subset times of SuperLU, finite-element vector, and finite-element matrix computations. The finite-element matrix time, which contains no communication, has nearly perfect scaling. The FE vector time contains the all-to-all communication necessary for FFTs. However, the number of processors participating in each all-to-all call is not increasing in this scaling, rather the amount of transmitted data per call is decreasing as the finite-element mesh is decomposed. The LU factorization and solve for the preconditioner, performed with the external SuperLU_DIST (3.1) library, does not scale as well as the other large components of the time-step loop.

[1] X. S. Li and J. W. Demmel, ACM Trans. Math. Software **29**, 110 (2003).

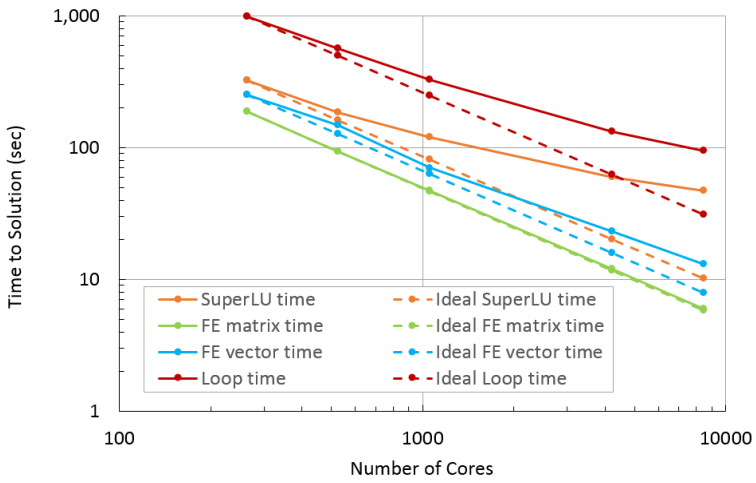

Figure 1. Strong-scaling results with hybrid parallelism, 8 MPI cores with three OpenMP threads per node, on the Hopper XE6 in late spring 2012 (jking@txcorp.com). The job represents 50 time-steps for a production NIMROD case: a 48×64 quartic-polynomial-basis finite-element mesh with 22 Fourier components. Loop time reports the full time taken by the NIMROD time advance, while other timers represent non-overlapping measures of aspects of the time advance. The SuperLU_DIST 3.1 library is linked.

Figure 2 plots weak-scaling data for the same computation with 8448 to 65,664 cores. The number of Fourier components is incremented in each case: 22, 43, 76 and 171 for respectively larger numbers of cores. As with the strong scaling in Figure 1, the time-advance loop, FE vector, FE matrix and SuperLU times are plotted. The FE matrix and SuperLU kernels both show good scaling, as the former does not perform communication, and processor grid size for the latter is unchanged by this scaling. The number of processors participating in the all-to-all calls for FFTs is increased proportionally to the job size, as reflected in the increased FE vector time.

The memory requirements vary depending on the parallel decomposition in the threads and MPI processes. The memory usage on a single Hopper XE6 node (prior to the development described in the next paragraph) is shown in Figure 3 for varying numbers of threads and MPI cores. Typical production runs have 1-3 Gb per MPI core, and efficient memory use is achievable with threads. Thus, the implementation of threading facilitates large, *i.e.* high resolution, computation in addition to providing more flexible scaling.

During this past year, Jacob King refined the distributed algorithm for exporting matrix data to external solvers. The memory-scaling of persistent object data stored during run time has been improved in the new algorithm ('dsta') relative to previous implementations ('dstm' and 'dist'), as shown by the weak-scaling data in Figure 4. Use of mpi_allgather in dstm to determine communication partners for the distributed matrix operations is replaced with point-to-point communication in dsta, and the matrix sparsity pattern stored in the NIMROD factor structure is fully distributed. With the new implementation, large previously inaccessible, finite-element-mesh resolution is now achievable.

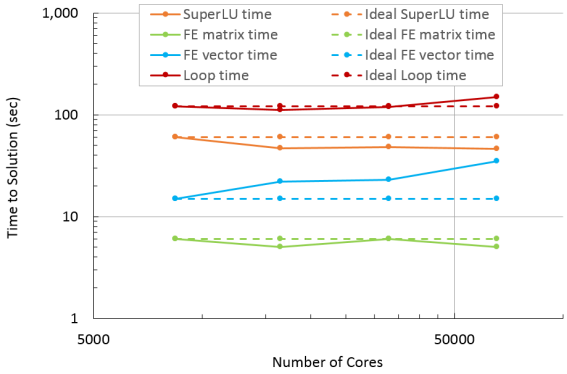

Figure 2. Weak-scaling results with hybrid parallelism, 4 MPI cores with six OpenMP threads per node, on the Hopper XE6 in late spring 2012 (jking@txcorp.com). The job is identical to that presented in Fig. 1, except the number of Fourier components are 22, 43, 76 and 171 as the number of processors is increased, respectively.

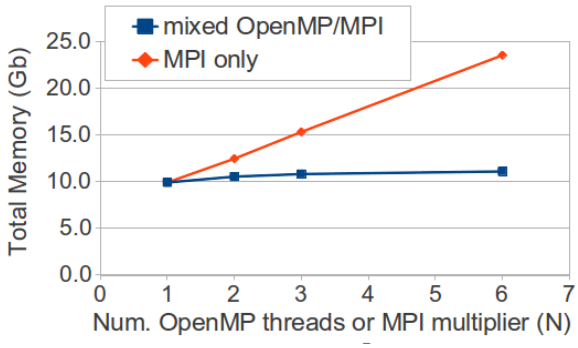

Figure 3. The effect of hybrid and MPI-only parallelism on memory usage on a single Hopper XE6 node for a job with a 48×64 quartic polynomial finite-element mesh with 3 Fourier components (jking@txcorp.com). The number of cores in this computation is 4×N, where N is either the number of threads or a multiplicative factor that determines the total number of MPI processes. Profiling is done with the CrayPat performance analysis tool.

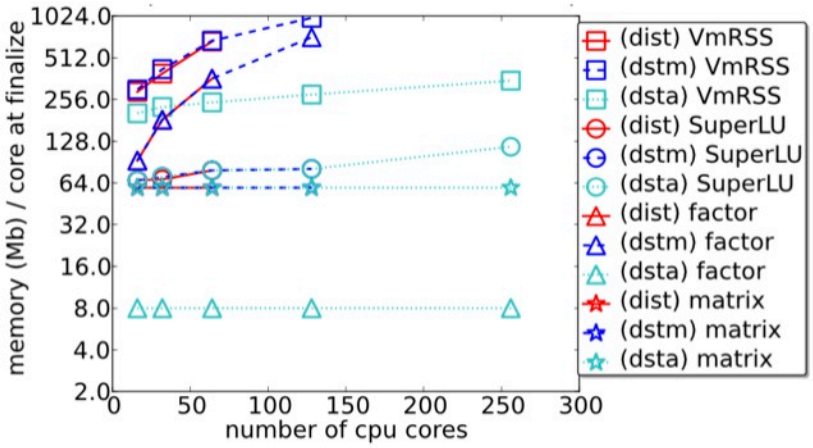

Figure 4. Weak-scaling results for a case with 72 spectral finite elements per core, from the Cray XK7 Titan at ORNL, spring 2013 (jking@txcorp.com). The labels dist, dstm and dsta refer to different algorithms within NIMROD that export matrix data to external solvers. The results show memory required for setting-up the 2D matrix that is used to precondition 3D algebraic solves. As indicated in Fig. 2, a full 3D computation may use more than 100 Fourier components and a correspondingly increased factor of CPU cores.