PARALLEL GLOBAL LINE JACOBI PRECONDITIONING IN NIMROD

Carl Sovinec, LANL
June 11, 1999

This note is a sequel to the report, "New Results with Line Jacobi Preconditioning," dated May 27, 1999, where I promoted the convergence properties of block-based line Jacobi preconditioning in cases such that grid blocks are periodic (circles or annuli). I concluded that the results were sufficiently encouraging to warrant trying global line Jacobi preconditioning in parallel. Here, I discuss one parallel implementation (using decomposition swapping) and scaling results. In short, expectations are borne out. Using this global preconditioning will reduce computation time and increase parallel efficiency significantly for many large-scale problems run with NIMROD. I'll start with a brief description of the algorithm that has been implemented, then discuss fixed problem size and increasing problem size scalings run on the T3E at NERSC and on Nirvana, the open-partition of ASCI Blue Mountain, at LANL.

The computations performed in global line Jacobi are essentially the same as the computations performed in block-based line Jacobi. One-dimensional matrices, resulting from throwing out all off-diagonal elements in one of the two logical rblock dimensions, are solved directly. The finite element formulation yields connections only among nearest neighbors, so these matrices are tridiagonal in form, with or without periodic elements, over vertex-local blocks of vector components. Linear systems are solved for both sets of grid lines (horizontal and vertical in logical space), and the solutions are averaged. What's different from the block-based scheme is that the lines now extend over adjacent, conforming rblocks to the largest extent possible. The communication set-up routine (parallel_line_init in file parallel.f) does not assume a regular block pattern. There may be multiple, unconnected rblock regions, and different sets of lines will extend over each.

The heart of the scheme is a pair of decomposition swaps completed before and after the tridiagonal solves. Where rblocks form a conforming linear pattern, the data is reorganized and transferred among processors as necessary to form long thin blocks. This is illustrated in Fig. 1. The swaps permit complete, serial 1-D solves in the long dimension of the new regions. The lines separating different sets of rblocks (the top and bottom borders in Fig. 1), are presently computed redundantly by the new blocks above and below the border. This avoids a communication step that is unique to these borders, but it leads to inefficiency if the surface to volume ratio of the rblocks is small. The communication and copying operations are handled by the parallel_line_comm_bl2line and parallel_line_comm_line2bl routines that have been added to parallel.f. Separate versions of each routine are also required for complex data; these routines have the _comp suffix. Note that the bl2line and line2bl operations are not exact inverses of each other, since normal blocks share border data. In forming the lines, shared data is only taken from one side of the rblock border, but it’s placed on both sides when returning to the rblock pattern.

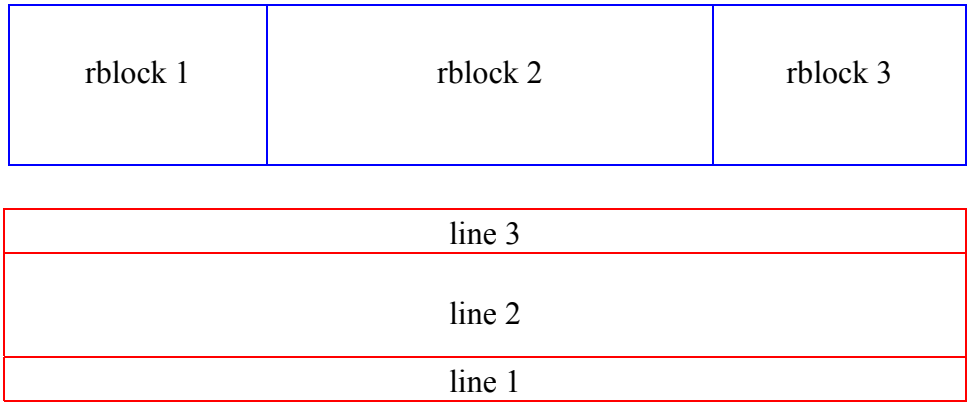

Figure 1. Decomposition swaps reorganize from the normal rblock pattern (above) to the line pattern (below) for global preconditioning, then swap back to the rblock pattern.

The swapping is done for both sets of line directions during each conjugate gradient iteration. This requires a lot of copying and communication, but it is worthwhile. To demonstrate how well the parallel global preconditioning works, let's consider the 100x100 grid DIII-D case used in the previous report. The time step is one global Alfven time. Figure 2 shows the time per time step (averaged over 5 steps) for the time step loop, the swapping time, the normal seam communication time, and the parallel efficiency relative to the one-block serial problem, as the grid is broken into multiple blocks and more processors are used. The nxbl / nybl block decomposition is balanced (1x1, 2x1, 2x2, 3x2, ...) to minimize the surface to volume ratio for seaming. [Unbalanced decomposition should be tested, too.] With 16 processors, the speed-up is 6.8; efficiency loss is monotonic. The number of cg iterations remains at 306 (+-1, *possibly* due to numerical difference in matrix elements that are summed across block borders in some cases but not in others), since the line solves are independent of block decomposition. For comparison, a 16x1 block line Jacobi run takes 847 iterations, and a 4x4 block-direct (full 2D Lapack solve within each rblock) takes 1022.[1]

The degree to which this improves fixed-size performance over our usual approach is illustrated in Figures 3 and 4. A comparison is made among the new global line Jacobi and the block-based line Jacobi, run with annular decomposition (nybl=1) and balanced decomposition (nybl~nxbl). Using periodic blocks and block-based preconditioning takes about half the number of iterations as using balanced decomposition; however, the global scheme is another factor of 2.8 better at 16 processors. Timing information is plotted in Figure 4 as the inverse of

[1] Some fraction (10-20%, perhaps) of the discrepancy results from the symmetric averaging at the degenerate point. Averaging between blocks is not needed with the global line scheme, and the 1-block discrepancy, 306 vs. 385, brought this issue to my attention. Thus, the block schemes can be improved at the degenerate point, but scaling trends will undoubtedly remain.

loop time to emphasize the large-processor performance. While the moderate processor range benefits

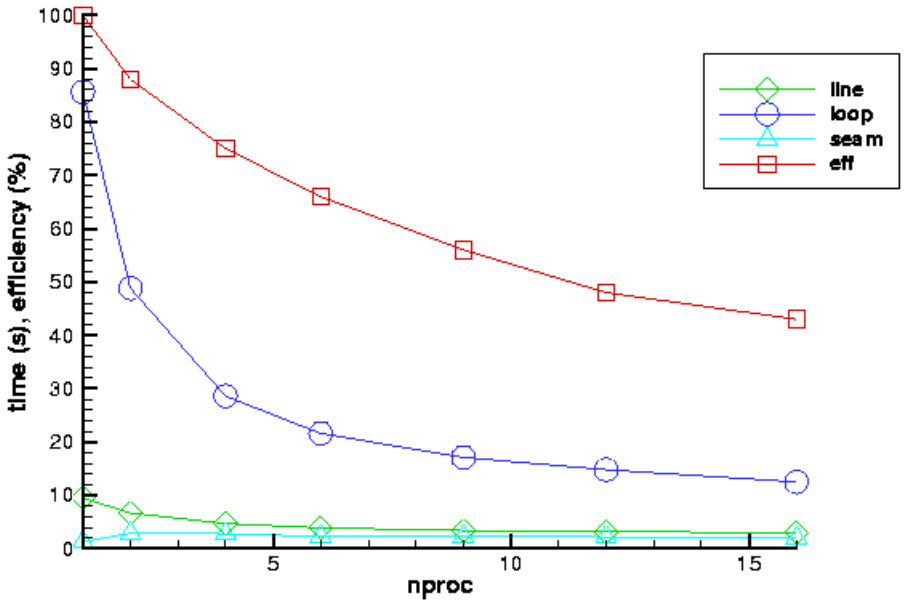

Figure 2. Fixed problem size scaling for global line Jacobi on the T3E for the 100x100 grid case. The reported times are the total time step loop ("loop"), the line swapping time ("line") including data copies and parallel communication, and the seam communication time ("seam"). Times are per time step, averaged over 5 steps. The efficiency is (1-block loop time)/(# processors * loop time). The number of blocks equals the number of processors in each case, and the nxbl / nybl decomposition is balanced.

from the switch to annular decomposition, the improvement tapers off with increasing decomposition. This is unexpected from the iteration performance, and it's not due to seaming--seam times for the two block-based preconditioning runs with 16 processors are virtually identical. It may result from memory caching as the dimensions of the balanced case become small. The new global scheme is considerably better than either block-based approach, running a factor of 2.2 faster than the annular case at 16 processors. It also beats using block-direct preconditioning, where 16 block balanced decomposition takes 1022 iterations at a rate of 3.1×10^{-2} s^{-1} and annular decomposition takes 736 iterations at a rate of 3.3×10^{-2} s^{-1}.

On Nirvana, the fixed problem scaling is more impressive. Figure 5 shows the timings and parallel efficiency for the same cases as Figure 2. Steve has pointed out that the high efficiency suggests some type of memory caching effect. Nonetheless, the 16-processor case demonstrates a 13.5 speed-up factor. On this machine, the global 16-processor time is 3.5 times faster than annular block line Jacobi, and 6.7 times faster than 4x4 block-direct.

The fixed problem scaling tests how much faster a given problem can be run by using more processors. Increasing the problem size while adding processors attacks the issue of performing more difficult simulations. Here, I use the same equilibrium as above but vary the grid size with

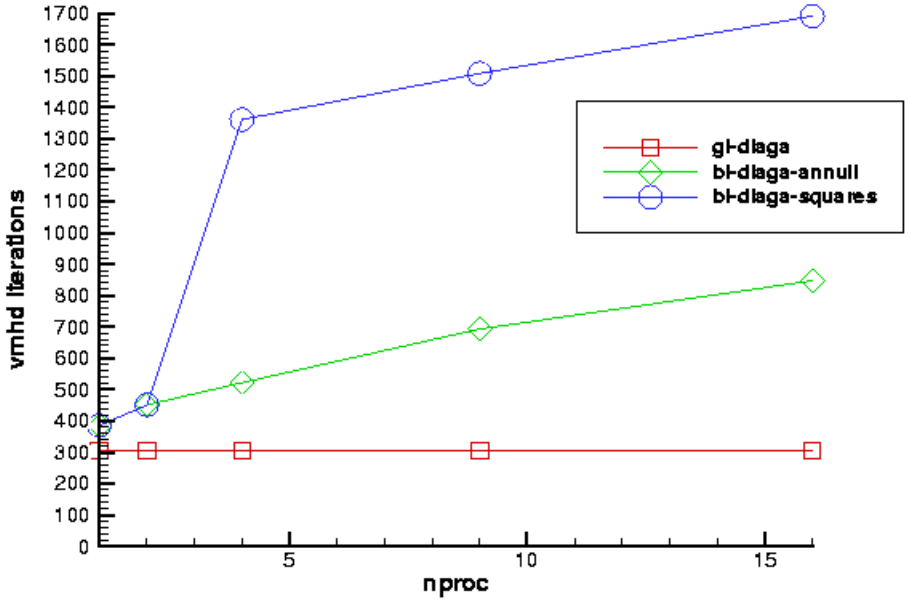

Figure 3. Comparison of preconditioner effectiveness on the 100x100 case. Block-based line Jacobi with periodic blocks (nybl=1, nxbl=nprocs), and balanced decomposition are compared with the global scheme.

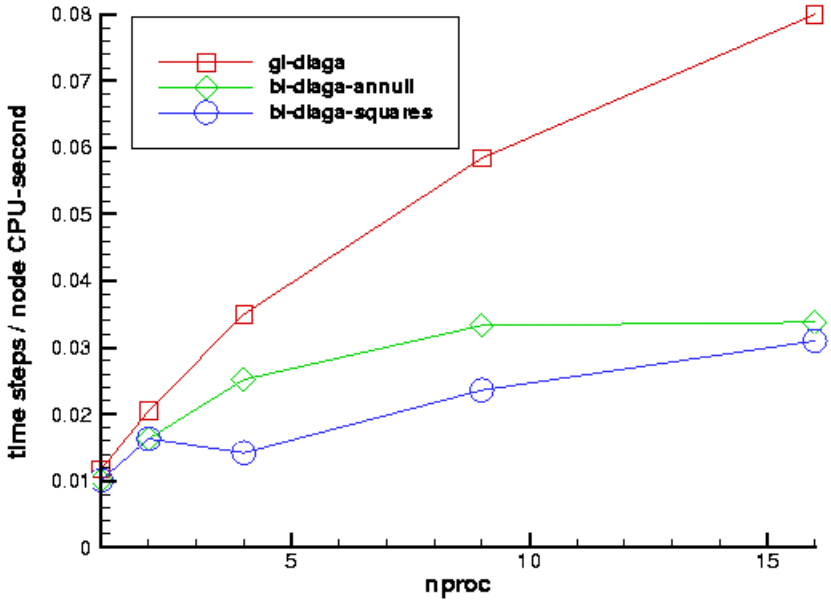

Figure 4. Comparison of time step rates (per CPU time per processor). The rate is the inverse of the loop time per node per time step.

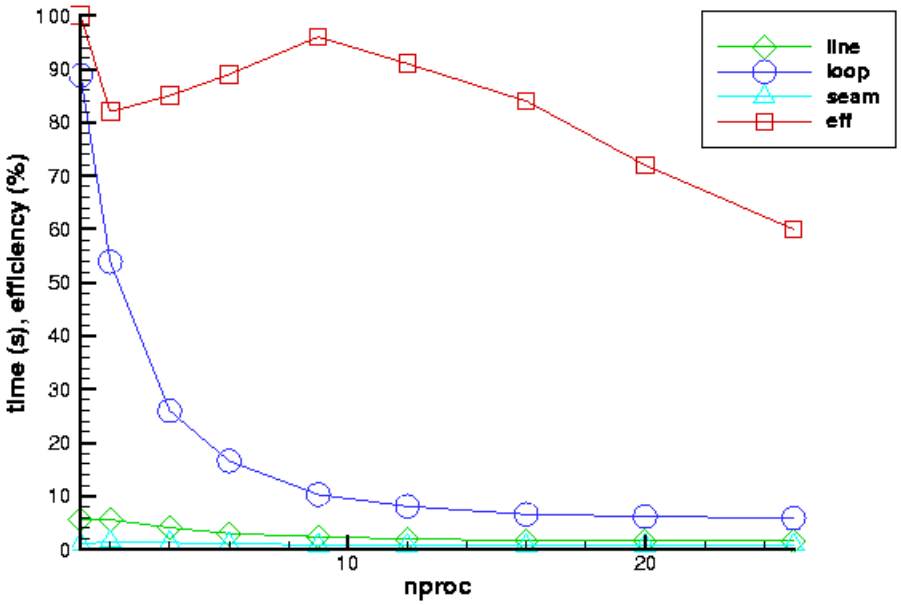

Figure 5. Fixed problem size scaling for global line Jacobi on Nirvana for the 100x100 grid case. The nxbl / nybl decomposition is balanced.

the number of processors such that each block contains 2500 cells (50x50) with one processor per block. The smallest case has one block and one processor, and the largest case has 64 blocks and processors. The results from tests run on the T3E are shown in Figure 6. The number of CG iterations is linear in the number of cells across one dimension (and the root of the total number of cells). The loop time increases linearly, too, to a large extent; magnetic field iterations are increasing faster than velocity iterations, and this impacts the large problem times. Like the block line Jacobi results with periodic blocks, global line Jacobi does not produce a perfect scaling. That would have iteration count independent of the grid size. But, the improvement is significant. In Figures 7 and 8, I compare iterations and loop times for different schemes up to a 200x200 grid. At the 200x200 grid, global preconditioning is 2.1 times faster than annular block preconditioning (saving a factor of 2.6 in iterations), and it's 3.3 times faster than the balanced block preconditioning (saving a factor of 7.6 in iterations). Using block-direct with the 200x200 grid takes 2475 iterations and 399 s per step.

The varied problem size computations on Nirvana (Figure 9) are less conclusive in terms of scaling performance. Up to the 200x200 grid, loop time scales as expected, but at 250x250, it jumps by a factor of three. My batch script to run this used the dedicated queue and precluded off-box decomposition, but some hardware issue must be coming into play.

The last figure (10) is a time step scaling with a 150x150 grid, 9-block and processor decomposition, and global line Jacobi preconditioning. Here I've varied dt from 1/100 of an Alfven time to 1000 Alfven times (wave CFL is 2×10^7). The velocity iterations saturate at about

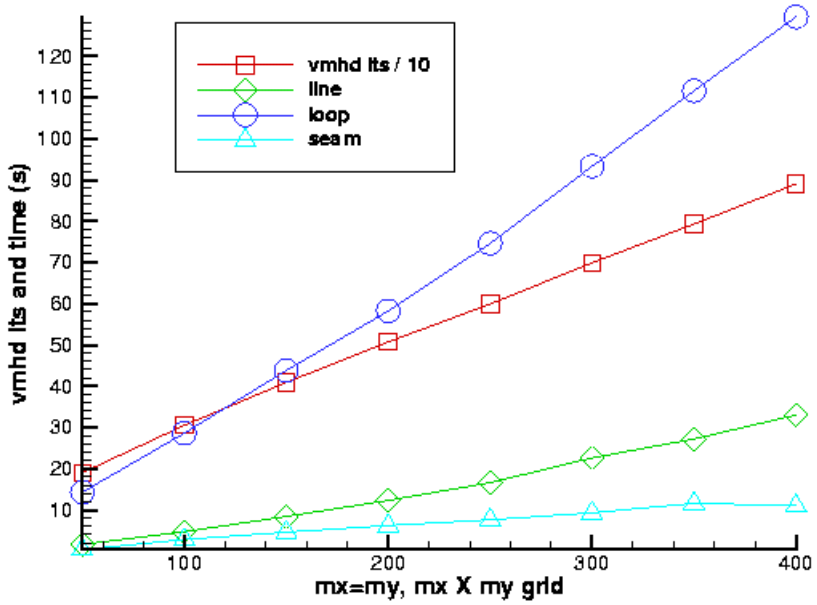

Figure 6. Increasing problem size scaling for global line Jacobi on the T3E. The nxbl / nybl decomposition is balanced.

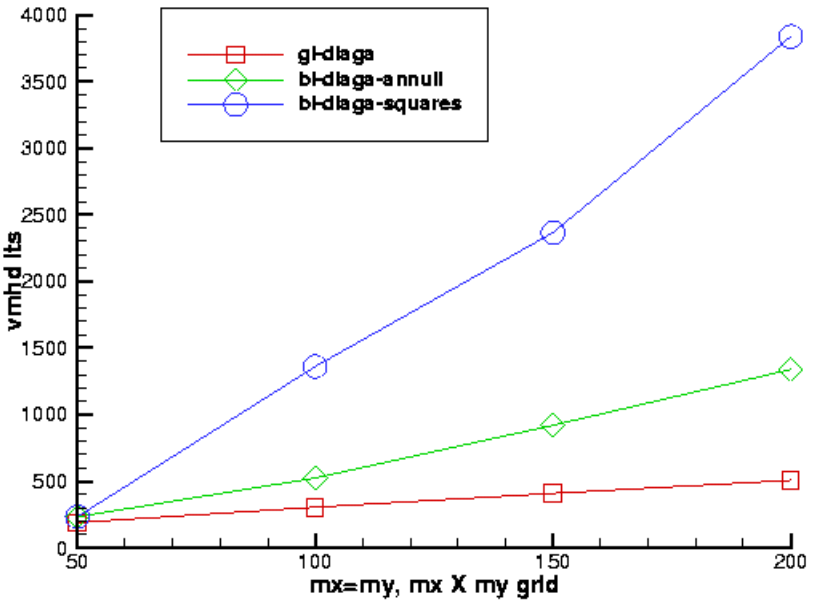

Figure 7. Comparison of preconditioner effectiveness with increasing problem size.

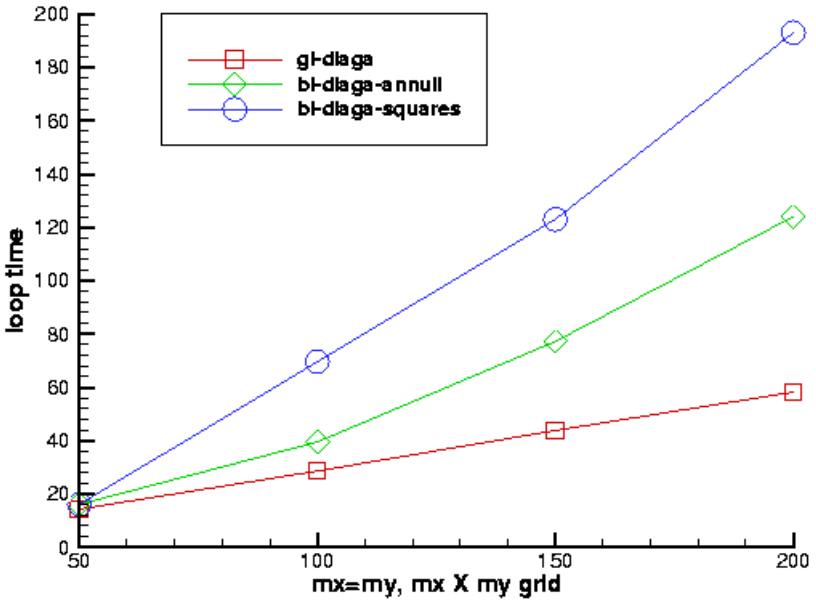

Figure 8. Comparison of time per time step on the T3E with increasing problem size.

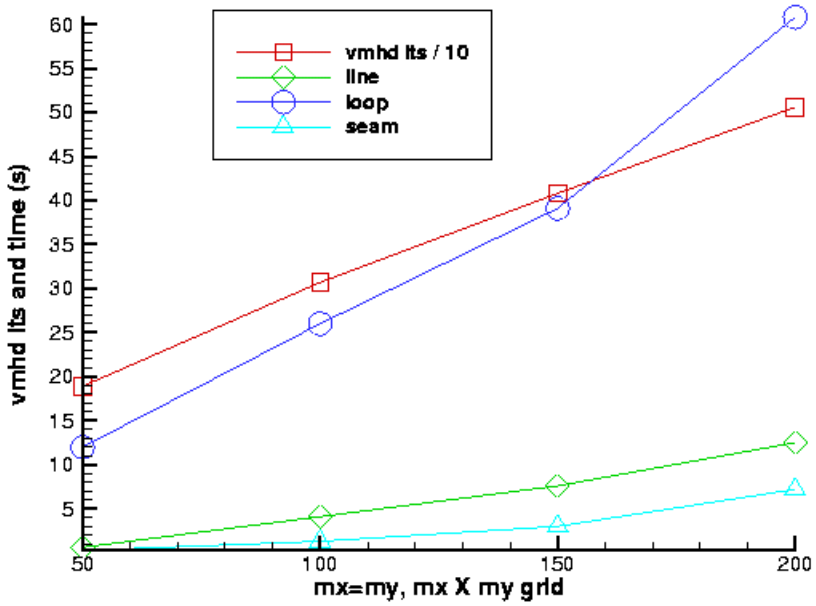

Figure 9. Increasing problem size scaling for global line Jacobi on Nirvana. The nxbl / nybl decomposition is balanced.

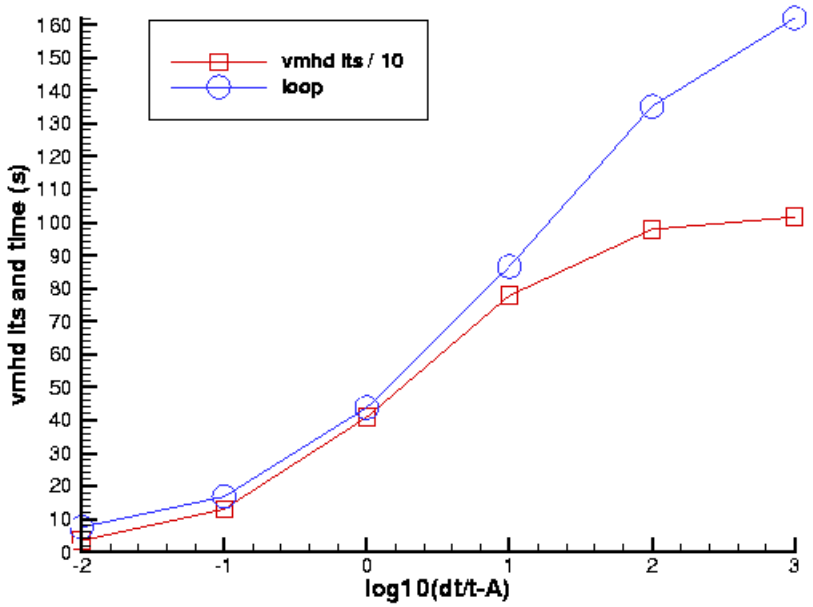

Figure 10. Velocity iterations and CPU time per time step for global line Jacobi on the T3E with varied time step.

1000. The magnetic field iterations (not shown) seem to saturate at a larger time step than the velocity iterations, so CPU time keeps increasing, but also shows signs of saturating. As with block line Jacobi and annular decomposition, the simulation will always cover a certain amount of simulation time fastest with the largest time step. The time step can be set for accuracy requirements or flow CFL limitations. [We may need to start thinking about implicit advection at some point in time.]

A few items need to be completed. The block-border averaging needs to deal with borders between rblocks and tblocks. The global scheme uses a modified procedure between rblocks, and this has temporarily been applied to all borders. Scalar optimization may improve the copying of data for decomposition swaps (reducing "line" time), and the line solves may benefit from more optimization, too. [I wouldn't expect much on the latter. While testing the line Gauss-Seidel, I tried using Lapack routines and results were nearly identical.] Finally, I've asked Steve to check over the communication calls to see if they can be improved.

While the test results presented here have all been completed on linear problems, multiple Fourier components introduce no additional difficulties. The block-to-line data swapping is done within each Fourier layer, independently. Total efficiency should roughly be the product of the linear test efficiency and the layer efficiency. The latter remains high (~80%) when the number of layers is the same as the number of Fourier coefficients; this has been tested up to 86 layers and Fourier coefficients. Thus, it's not hard to imagine using a thousand or more processors on a large nonlinear problem with the code we now have.

While this new parallel global preconditioning option is a significant improvement, it is not the last word with regard to linear algebra in NIMROD. As I've already noted, the scaling isn't perfect, and we may find something better. In addition, while it's possible that something like it can be used in the tblocks, that possibility is probably remote. Finally, we may need to solve nonsymmetric matrices at some point in time. For these reasons, I still support implementing AZTEC and investigating other improvements for the Fortran 90 solver in NIMROD. In the meantime, the latest option will help many problems run significantly faster.

As always, I advise testing different preconditioning options before running a large simulation. You should do pretty well with the new global line option in all cases, but the condition number of our matrices varies tremendously with the problem and with resolution. A few minutes spent testing different options can easily save hours of run time.