NEW RESULTS WITH LINE JACOBI PRECONDITIONING

Carl Sovinec, LANL

May 27, 1999

This note reports on recent findings regarding the Fortran 90 preconditioned conjugate gradient solver that is part of NIMROD. This study started with Mike Heroux's suggestion to try line Gauss-Seidel as a preconditioner in the structured blocks of quadrilaterals. While the new line Gauss-Seidel is often better than our ILU, a bug fix in the 1D solves used with line Jacobi (solver='bl_diag*') leads to significantly improved performance in periodic grid blocks. In fact, the new results are encouraging enough to inspire thoughts on parallel *global* line Jacobi. I'll touch on that again in the conclusion. First, I'll provide a description, then the new timings.

As a refresher, a Jacobi solver is the simplest matrix splitting technique for an iterative solve of a matrix equation. If we split the matrix **A** into **L**+**D**+**U**, where **D** is the diagonal and **L** and **U** are the strictly lower and upper triangles, respectively, Jacobi iterations find the solution of

$$\mathbf{A} \cdot \mathbf{x} = \mathbf{b}$$

by iterating

$$\mathbf{D} \cdot \mathbf{x}_{i+1} = \mathbf{b} - (\mathbf{L} + \mathbf{U}) \cdot \mathbf{x}_i$$

or

$$\mathbf{D} \cdot \Delta\mathbf{x}_{i+1} = \mathbf{r}_i \equiv \mathbf{b} - \mathbf{A} \cdot \mathbf{x}_i \ .$$

The previous equation will look familiar if you have read Dalton's recent document on our ill-conditioned matrices. The scheme converges provided the magnitude of the largest eigenvalue of $\mathbf{D}^{-1}(\mathbf{L}+\mathbf{U})$ is less than unity. This step of solving an approximation for **A** (**D** in this case) with the residual being the rhs is also the preconditioning step for CG, provided the approximation is symmetric. Thus, a Jacobi step can be used as a preconditioner.

If we generalize what we call the diagonal of **A**, we can improve performance. NIMROD's 'diagonal' option takes an element of **x** to be a physical vector, so the nqtyXnqty (often 3x3) submatrices or "blocks" (as used in linear algebra jargon, not grid blocks) along the diagonal of **A** are inverted. If we further generalize and consider an entire grid line within a structured grid block as an element of a 1D matrix, we have line Jacobi. As another refinement, we can perform the Jacobi step with both the x and y grid lines (usually radial and azimuthal) separately, using the same rhs for both sets of lines, then add the two results together. This is an additive technique, and it's what the 'bl_diaga' option does. We can also do Gauss-Seidel sweeps with lines, provided one sweeps up and back each preconditioning step to preserve symmetry.

On to results--what's displayed in Figs. 1-3 is from one of the early DIII-D cases provided by Ming. The first set of data uses a single grid block with a 100x100 grid. The time step is an Alfven time, which corresponds to a wave-CFL number of 8800 for this grid. Results are averaged over 5 time steps (except ILU, where only 2 steps are taken), nybl=1, so grid blocks are annuli or circles, and the number of T3E processors used is the same as the number of grid blocks.

From Fig. 1 we can observe that both line-based methods are much more effective than ILU, especially for coarse block decomposition. This probably results from the ILU generating much smaller off-diagonal factors to maintain stability, and these factors are applied block-wise, since it's a 2D incomplete factorization. The off-diagonal factors in the line methods are applied line-by-line. Another issue is the importance of doing full solves in the azimuthal direction. With the line Jacobi and nxbl=1 and nybl=2, the number VMHD its jumps to 1339, whereas the 2x1 decomposition is 451. This probably affects ILU as well. The azimuthal direction in our ILU is ordered as the band offset, which is treated less accurately than the inner, radial direction in the incomplete factorization, I think. Thus, our ILU may do better with reordering, but this modification is not as trivial as it sounds, due to special coding for periodic connections and the degenerate point.

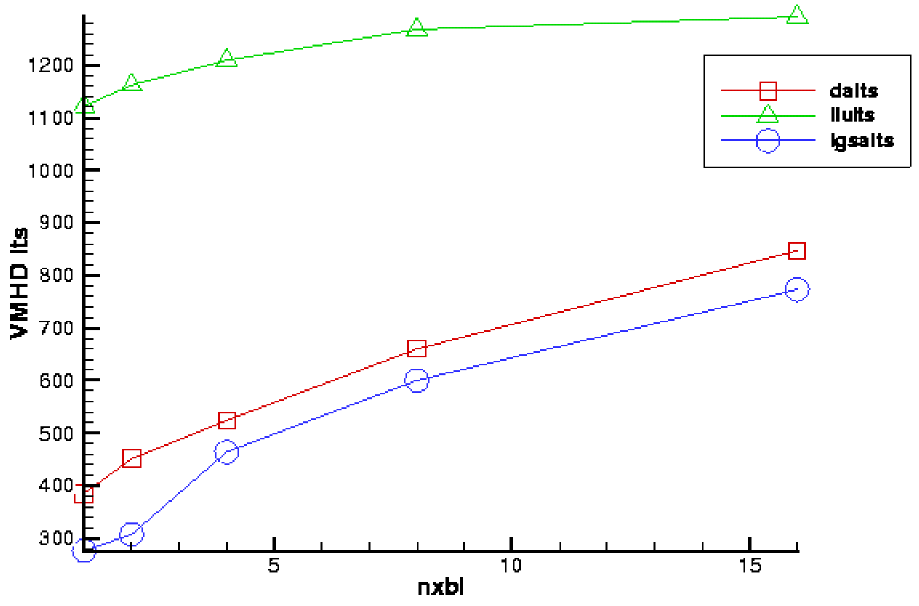

Figure 1. Conjugate gradient iterations per time step with line Jacobi (da), ILU, and line Gauss-Seidel (lgsa) for the 100x100 grid problem with nybl=1. Both line methods plotted use the average of solves and/or passes in the two different grid line directions.

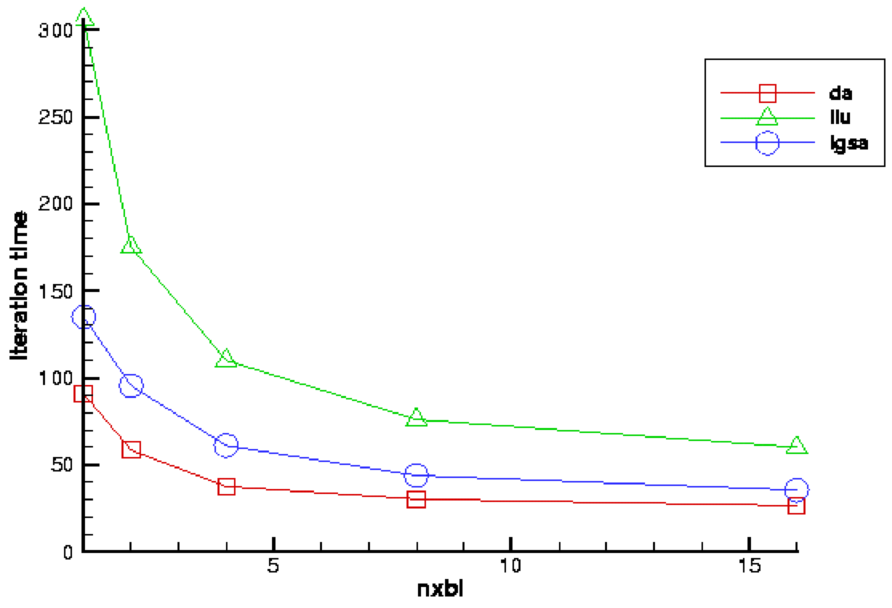

Figure 2. Iteration time per time step on the T3E for the same cases displayed in Fig. 1.

The iteration or solver time per step is shown in Fig. 2 for the same cases. The corrected line Jacobi easily wins over ILU, and it also does better than the line Gauss-Seidel. The line GS always has fewer iterations, but it requires two line solves plus two line matrix multiplies for every line solve in line Jacobi. Since most of the time is spent in the preconditioner (unlike methods for nonsymmetric matrices), the iteration savings is not enough to reduce total time. Checking a 2x2 block decomposition, ILU does a little better, 101s vs. 110s for 4x1, while both line methods are hurt losing the full periodic solves: 67.6 vs. 37.5 for Jacobi and 90.8 vs. 60.6 for GS. The parallel efficiency doesn't hold up with any of the block-based solves. This led Alan to his foray into two-level preconditioning. Let's leave this topic for later.

For the next segment, let's consider how well the line Jacobi behaves with fixed block decomposition while varying the difficulty of the linear problem. First, let's vary the time step. Figure 3 shows the number of CG iterations and iteration time per step while varying from 1/10 to 100 Alfven times. While both increase with $\log_{10}(dt)$, the increases are obviously no worse than logarithmic, and the iterations indicate the saturation Alan showed as the problem becomes elliptic in the limit of large dt. [The largest dt represents a wave CFL of 880,000: "Send us your tired and slow modes."] Clearly, if accuracy permits for slow phenomena (and it should), the time step can be set for the physics (or flow CFL) without having CPU time shooting up due to the preconditioner faltering. The fastest computation is at the largest time step permissible by the numerical algorithm and accuracy considerations.

Physically interesting simulations also need a lot of resolution. In Fig. 4, the resolution is varied from 50x50 to 200x200, using four blocks, since a 1-block 200x200 case doesn't fit in the memory of a single T3E processor. The iterations do increase with resolution, so this isn't

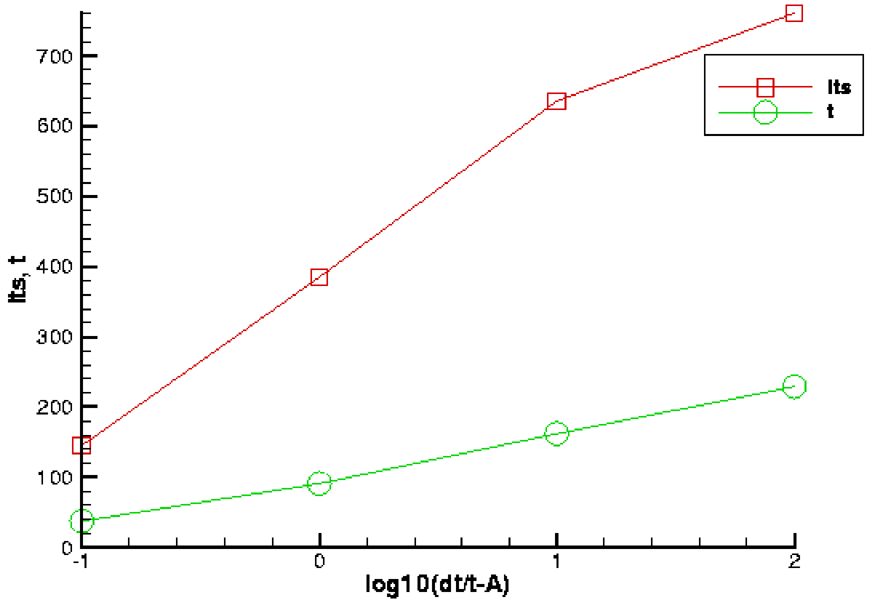

Figure 3. Iterations and iteration time per time step with line Jacobi preconditioning as the time step is varied over three orders of magnitude for the 100x100 grid, 1block test.

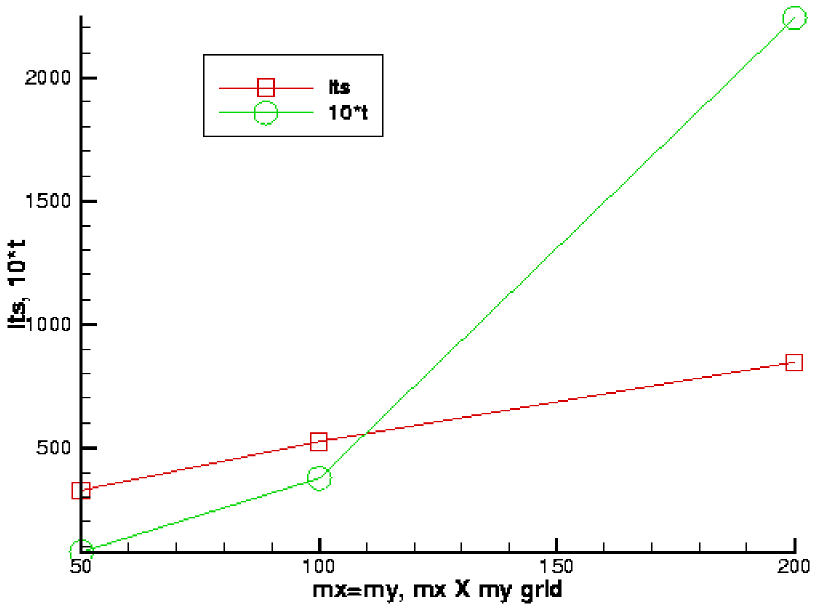

Figure 4. Iterations and iteration time per time step with line Jacobi preconditioning as the grid is varied from 50x50 to 200x200 with a nxbl=4, nybl=1 block decomposition and the time step at one Alfven time. Note that iteration time is multiplied by 10 to use a single vertical scale.

scaling perfectly; however, this is the best I have seen, yet. ILU, for example jumps from 480 iterations at 50x50 to 1209 at 100x100. The iteration time with line Jacobi, therefore, does a little worse than quadratic in mx=my or linear in the number of cells.

The parallel efficiency of the block-based preconditioners suffers with decomposition. Figure 5 shows efficiency, t(one proc.)/t(n procs.)*n, vs. number of processors (or blocks) for the original 100x100 grid cases of the first two figures. Part of this is the increasing number of CG iterations as the radial lines are shortened. Using only azimuthal lines, which would be the limit for averaged line Jacobi with a block being each azimuthal grid line, the iteration count is 1856. Part of the efficiency loss is also communication, as the surface to volume ratio for the blocks increases. With 8 processors and averaged line Jacobi, seaming time is 21% of the time step, and at 16 processors, it's 31%.

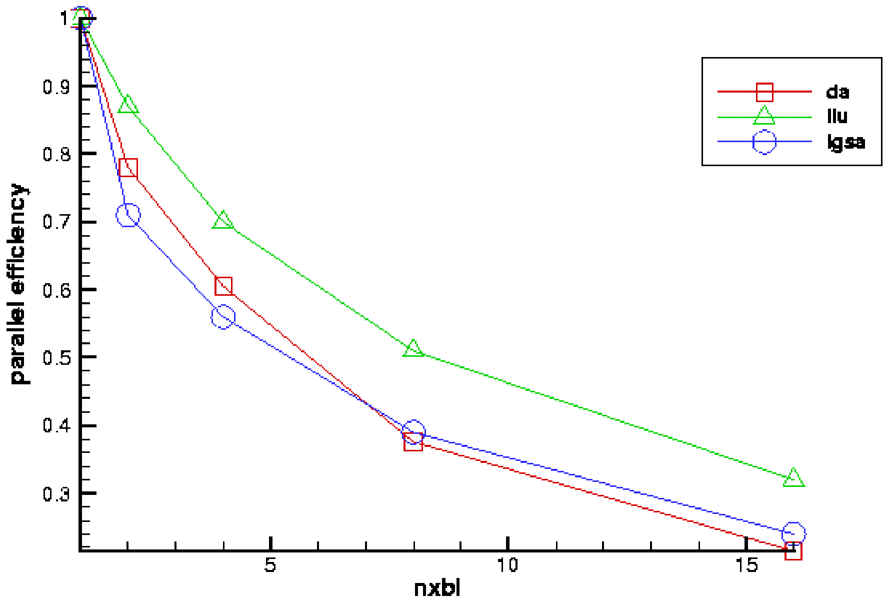

Figure 5. Parallel efficiency for the different preconditioners applied to the 100x100 problem of Figs. 1 and 2.

While many of our present-day computations can benefit from switching to line Jacobi with the bug fix, and keeping blocks periodic, I'm excited about the prospect of performing the line Jacobi preconditioning globally. This would retain the effectiveness of one-block preconditioning with the grid broken into blocks. The important feature of the line Jacobi is that each line solve is independent of the others, and where there's independence, there's potential parallelization. [With the layer decomposition, the independence is between Fourier components for the linear solves and most of the finite element computations. When FFTs and nonlinear products are computed, the independence is between points in the poloidal plane.]

Steve and I discussed how the global line preconditioning could be done in parallel, and there seems to be two possibilities. We could perform decomposition swaps, like what's done with the layers. If a set of lines extends across multiple blocks, the set can be broken into the same number of subsets as there are processors. Data is swapped from grid-block decomposition to line-subset decomposition, line solves are performed serially, then the data is swapped back. This may be communication intensive, especially since it would require four, maybe three, (for line solves in both directions) swaps per cg iteration. Relating to the layers, the communication is a significant part of the whole FFT/pseudospectral operation, but it scales. If we can do this in the linear solves, it would be a major advance for NIMROD. The second possibility that Steve mentioned is doing parallel line solves. Here entire line data is not swapped. Some processors work on their sections of some of the lines, while other processors are working on their sections of different lines, then only data from the border is passed, and the different line solves are extended further. This should require much less communication, but it *may* be more difficult to implement. Both approaches are worth more thought and testing.

In summary, a bug fix in NIMROD's line Jacobi preconditioner option leads to improved linear solver performance on realistic problems when grid decomposition is in the radial direction only. With fixed decomposition, the scaling of this *preconditioner*, i.e. how well the CG iterations remain fixed as problem size varies, seems to be much closer to ideal than anything else we have tried. With this serial performance and the independence of the different line solves, there is hope that global preconditioning can be effective in parallel, too.