Efficient Convergence Acceleration for a Parallel CFD Code

R.D. Williams[a], J. Häuser[b], and R. Winkelmann[b]

[a]California Institute of Technology, Pasadena, California
[b]Center of Logistics and Expert Systems, Salzgitter, Germany

1. INTRODUCTION

In this paper, we consider strategies for efficiently solving the system of nonlinear equations that arises from making an implicit time-step of a flow solver. By efficiency, we mean an analysis of costs, such as machine usage, and benefits, being approach to convergence. When the Navier-Stokes equations are discretized on to a given spatial mesh, using a given differencing scheme, the result is a set of ordinary differential equations, with time as the independent variable:

$$\frac{\partial \boldsymbol{u}_i}{\partial t} + \Phi(\boldsymbol{u}) = 0, \qquad \boldsymbol{u} \in \Re^N$$

Here, N is the number of degrees of freedom in the spatially discretized solution; $N=nk$ is a product of the number of grid-points, n, with the number of degrees of freedom per grid-point, k. The ODE's are then discretized using either an explicit or an implicit scheme, or a so-called block-implicit scheme. In the former case, we have a large number of independent equations to solve, one for each grid point, and in the latter cases, we have a large set of simultaneous nonlinear equations to solve, which may be written $\boldsymbol{F}(\boldsymbol{u}) = 0$, where

$$\boldsymbol{F}(\boldsymbol{u}) \equiv \frac{\boldsymbol{u} - \boldsymbol{u}_{old}}{\Delta t} + \Phi(\boldsymbol{u})$$

In the implicit case, such large sets of nonlinear equations are generally solved with a variant of Newton's method, where a sequence of linear systems are solved, and the method hopefully converges to the solution of the nonlinear system. The great advantage of the Newton iteration is its quadratic convergence, so that in a neighborhood of the solution, the residual after the step is a constant times the square of the residual before the step. In the following, we shall refer to the "Full Newton", or "Fully Implicit" methods to mean that we are doing the Newton step, where the linear systems are of size NxN. In spite of the cost of solving these large linear systems, this is clearly the correct method to use near convergence, because of the large benefit per step.

We shall also investigate "Block Implicit" methods, where the N independent variables are domain-decomposed into B blocks, so that the linear system of size N is replaced by B systems

Proceedings of the Parallel CFD'96 Conference,Capri, Italy, July 1996

each of size approximately N/B. Now the explicit methods can be seen to fit into a continuum from fully implicit, through block-implicit: the explicit scheme is simply an implicit one, but with each grid-point being its own 1x1 block.

2. BENEFITS AND COSTS

(a) **Benefit**: The benefit of each of the various methods is easy to define: it is simply the decrease on the logarithm of a residual function. In the following, we shall use a residual which is the L_2 norm of the ODE system,

$$\text{Residual} = \|\Phi(\boldsymbol{u})\|$$

so that steady state is achieved, by definition, when this quantity is zero. To be specific, we shall define the benefit of an iteration step as the *decrease in the logarithm of the residual*.

(b) **Cost**: Often in the CFD literature, there are graphs showing convergence as a function of the number of steps of some numerical method. While the measure of benefit is the same as above (based on residual), it is difficult in these cases to compare costs, since some numerical methods take much more computing resources per step than others. It is more appropriate, in our opinion, to compare numerical methods by a more realistic measure of the cost of the method, such as wall-clock time, accounting units, or total floating-point operations (flops). On a uniprocessor machine, these three measures are all (very roughly) equivalent, but on a parallel machine the ratio of the number of flops to the wall-clock time is not a constant but depends on the number of processors. If the parallel machine is being used by a single user, then the cost of using it is presumably related to the time it takes to complete a job, but in a more realistic, multi-user environment, it also depends on the number of processors used.

For users of the parallel machines at the Caltech CACR, charges are in terms of *node-hours*, which is the product of the number of processors used with the number of hours for which those processors were reserved. If the solver, running on P processors, has efficiency e, then the cost of executing a program of F flops is F/e. When the efficiency is close to one, the execution is cost is independent of the number of processors. In earlier work [3,4], we have shown that ParNSS is a very efficient code, so that the cost (in node-hours) for running the computation is essentially independent of the number of nodes.

As discussed above, using implicit methods means that a large proportion of the computation time is devoted to solving linear systems of equations. The number of independent grid variables, N, is also the order of the matrix. With a direct method like LU-decomposition, the flop-count, and hence the cost, is of order N^3.

Another estimate is provided by the convergence rate of Krylov-space methods, such as the GMRES [5,6] that we use to solve the linear systems. This is controlled by the condition number, which is generally inversely proportional to the square of the mesh spacing, h. If we assume that Nh^3 remains constant (conservation of geometric volume), then the condition number for an iterative method rises as $N^{2/3}$, so that the flop count for the solution rises at least as fast as $N^{5/3}$, which is much better than the direct methods, but at the price of less robustness. We point out that the analysis above is really restricted to symmetric matrices characteristic of elliptic PDE's, not the highly unsymmetric matrices that arise from hyperbolic systems. Of course we should also note that for an explicit step, the flop count is proportional to N.

Now that we have defined benefit and cost, we need to measure these for the methods that we have available: from the expensive-but-effective fully implicit method, through the block-implicit methods, to the cheaper-but-less-effective explicit method. shall plot benefit against cost for each at various stages of the convergence. We should emphasize that we are trying to maximize efficiency, which is the ratio of benefit to cost.

3. MULTIBLOCK METHODS

The basic approach is that the complex geometry of the solution domain is partitioned into a number B of smaller domains, each topologically equivalent to a simple domain such as a hexahedron: we shall refer to these as blocks. This partition comes from the grid-generation process, based on the geometry and topology of the solution domain, optimization of grid quality, and *a priori* knowledge of the investigator about the flow field. We would like to point out that topologically, we can always use a single block as a grid for essentially any arbitrarily complex geometry, but in most cases of interest, a numerical solver will not be able to use this highly distorted monoblock for simulation of a fluid flow.

For efficient numerical solution of general PDE's, there are certain conditions that a grid must satisfy [1], such as resolution of physical features of the domain boundary, good control of grid skewness, volume ratio of neighboring cells, grid smoothness and lack of singularities in the grid. For CFD, an additional requirement for the grid is that the grid-line configuration is reasonably aligned with the streamlines of the flow, and for high-Reynolds number flow, there must also be special gridding near physical boundaries so that the boundary layer is correctly modeled. Given a solution domain with complex geometry, together with these constraints, there is a minimum number of blocks that can satisfy these requirements. Thus we assume in the following that the basic topology of B blocks has already been generated with careful regard to grid quality, and that the total number of grid-points is n.

4. ParNSS — PARALLEL NAVIER-STOKES SOLVER

In the following, we use the ParNSS code [1-4] as a workbench on which to test ideas of cost and benefit. ParNSS is a multiblock Navier-Stokes solver originally developed to simulate hypersonic flow, now extended to transonic and subsonic flow. The message-passing is written in PVM, ensuring portability: ParNSS has run on SGI systems at CLE in Salzgitter, Germany, Intel Paragons at Caltech in Pasadena, California, and on the IBM SP2 at ESTEC in Noordwijk, Netherlands. ParNSS is a finite-volume, hexahedral multiblock solver that utilizes various techniques to accelerate convergence to a steady state, and we briefly summarize these techniques below.

For the purposes of this paper, we shall consider the computational objective to be steady state flow rather than a time-accurate unsteady computation. In particular, this paper is concerned with approximating the full linear system inherent in a full implicit step with many small linear systems, through domain decomposition.

Time-Stepping: the CFL Number

The steady state is approached by a process that is similar to time-stepping. For each block, we set up the equations for an implicit time-step and solve the resulting nonlinear equations, except that *each block can choose its own time-step*. For each block b, we compute the natural time-step Δt_b, which is the minimum over grid cells of the size of the cell divided by the fluid velocity in the cell. This timestep is that corresponding to a CFL number of 1, and is the largest that can be used if the explicit scheme is to be stable. The actual time-step that is used on each block is then a product of Δt_b with a parameter that we call the "CFL Number". We expect the implicit step to be stable even when the CFL Number is greater than 1, and we expect the explicit step to be unstable if the CFL Number is much larger than 1.

Notice that each block has its own timestep, determined from local conditions, so that we are not approaching the steady state through physical time-stepping, but simply using time as a parameter that can be manipulated and distorted to lead us to the steady state.

If the CFL Number is large, we make good progress toward our goal, the steady state, but the scheme that we using may not converge, or it may converge to unphysical values; but if the CFL Number is too small, we are being very safe, but making little progress. In ParNSS, we have chosen to start the CFL at some safe value, perhaps 0.01 where all the schemes converge, then increase it conservatively, at 5% per step, but when there is an indication of poor convergence the CFL Number is halved. In this way the CFL Number is pushed up to the greatest safe value, thereby maximizing the progress toward the steady state.

5. EXPLICIT, IMPLICIT AND BLOCK-IMPLICIT

Explicit steps

When the code begins, the flow is set to the free-stream values everywhere in the domain. We start with an explicit scheme to create the general configuration of the flow, including a rough approximation of the position of the shocks. The CFL number is initially very small, and it is increased cautiously and conservatively. Explicit schemes are highly efficient on parallel machines because of the large ratio of computation to calculation, and in simple cases this explicit iteration is sufficient for full convergence. However, in interesting cases a point is reached when the residual decreases very little even with large numbers of iterations: the computation is said to be stalled.

Block-implicit steps

An implicit scheme is used to continue iterations. The major advantage is stability even with CFL numbers considerably greater than one, and the consequent rapid convergence. The disadvantage is that a fully implicit step involves the solution of a linear system involving the mesh over the whole solution domain, and is therefore much more computationally intensive than explicit steps. In fact we do not use fully implicit steps at this stage, but rather a block-implicit scheme with a locally- determined time step on each block, meaning that the time steps are in general different on each block. We are approximately solving the system of linear equations by neglecting matrix elements which connect cells in different blocks. Since it is the blocks which are distributed among processors, this approximation means that each solve can occur within a processor with no communication to other processors. Furthermore it should be noted that the value of a step is measured according to how much it can reduce the residual, not according to how well it solves the fully-implicit linear system; we are not doing a time-accurate solve here, but only trying to drive the residual to zero by any means possible.

During the block-implicit phase, the time-step is chosen on a block-by-block basis by a minimum over the block of a local smoothness measure. Here is another deviation form exact time- stepping, which would of course require the same time-step on all blocks.

As noted above, ParNSS is a very efficient code, and this is because blocks are not split across multiple processors. The reason for this is as follows. A block-implicit step involves solving a linear system whose size is the number of degrees of freedom in the block, which requires considerable computing resources; only after the linear solve is there communication with other blocks that may be on other processors.

Newton's Method — Fully Implicit

Finally, when it is clear that we are close to the steady state, we use the fully implicit step with a global time step, with all the communication overhead that this implies. We allow the timestep to become very large or infinite. The justification for this is by observing that an implicit step of a differential equation system with infinite time step is equivalent to using Newton's method for the steady state of that system. Newton's method is quadratically convergent when the initial approximation to the solution is close enough to the exact solution, meaning that the magnitude of the residual is squared with each step, so that it tends to zero very quickly.

Unfortunately we are not able to achieve full implicitness with our multiblock grids, since the linear solve always works on a single block. While blocks can be split to move away from full implicitness, we cannot reduce the number of blocks below the number B that is specified by the grid-generation process.

The Solution Scheme

The three steps to a solve that are expressed above may be recast as a “schedule of implicitness”: beginning with a fully explicit scheme, we move to a slightly implicit scheme with very small blocks (perhaps 3x3x3) being treated independently, then larger and larger sub-blocks until eventually we are doing the fully implicit scheme.

The schedule of implicitness is reminiscent of the temperature schedule in optimization by simulated annealing (SA). In that case, a stochastic acceptance-rejection procedure is used to minimize a function. The procedure is parameterized by a variable analogous to temperature, which decreases slowly to zero. The difficult part of a practical SA algorithm is the decision on how fast the temperature decreases, whether it is decided in advance or dynamically. In the same way, we must decide how many steps should be done for each sub-blocking or super-blocking, or produce a decision mechanism for moving on to the next larger subset of the grid-points for the implicit steps.

Numerical Experiments

The first experiment in convergence strategies is an Euler flow in two dimensions: the NACA0012 airfoil at Mach 1.7. The original grid is of 48000 points, with two blocks, and we have also split this grid into 8, 32 and 128 blocks. We have run this on an Intel Paragon distributed-memory machine using four different schemes: 8-Block Implicit, 32-Block Implicit, 128-Block Implicit, and Explicit. The residual is plotted on a log scale, which is our measure of benefit, and the horizontal axis is node-hours which is cost.

We approached convergence in three stages, allowing the methods to compete at each stage for the least cost to achieve a given benefit. The grid data from the best scheme was then used as input for the next stage.

Stage	Ending Residual	CFLstart (implicit)	CFLstart (explicit)	CFLmax (explicit)
1	0.1	0.01	0.01	0.5
2	0.001	1.0	0.5	0.5
3	0.00001	(continue)	0.5	0.5

As shown in the table, we used each scheme to reduce the residual from its initial value to 0.1, then to 0.001, again picked the most efficient scheme, then in the third stage reduced the residual to 0.00001. By this stage, the subsequent reduction to machine accuracy is just two or three more steps.

The CFL Number for each scheme was 0.01 to begin: this is because the flow is not even tangent to the boundaries, and we must be very conservative. The CFL now rises at 5% per step; for the explicit scheme it is bounded at 0.5 to prevent instability, but for the implicit schemes there is no maximum. Stage 2 has the starting value set at 1.0, and in Stage 3, the CFL continues from the value (197) that it had at the end of Stage 2.

The results are shown in Figure 1. We found, as hypothesized above, that the most efficient route to convergence for this problem begins with the Explicit scheme, then uses the 128-Block Implicit scheme, and finishes with the 8-Block Implicit scheme. Unfortunately we were not able to run the 2-block mesh because of memory constraints on the Paragon node — each node has 64 Mbytes.

The results for the explicit scheme in stage 2 show some spikes in the value of the residual, and as a result the CFL Number was halved to 0.25, only to rise back at 5% per step to the maximum value of 0.5. In other runs of ParNSS with different flow conditions, the implicit runs also show sudden increases in residual as a symptom of poor convergence, and we also halve the CFL Number in these cases with good effect.

Figure 2 shows the load balance of the block implicit method. A set of linear equations is solved independently on each block, using a Krylov method (GMRES), and so the number of iterations, and hence the time taken, is different on each block. In the Figure, the ratio of the areas of the shaded block to the outline block is proportional to the (wall-clock) time spent doing the GMRES solve on that block. We can see that most blocks take roughly the same time to be solved, but those in the undisturbed fluid above and to the left of the leading edge take much less time. For this particular experiment, it would seem that the load-balance is not sufficiently poor to warrant the creation of rebalancing software, and we will make further investigations to decide if this is true in general.

In the future, we will investigate the scaling of the computation and cost-benefit analysis with variable n, the number of grid-points, to complement this work on scaling with the number of blocks.