Final Report for first year of prime Contract NAS2-14090 "Research in Computational Aeroscience Applications Implemented on Advanced Parallel Computing Systems"

Larry Wigton, March 28, 1996

Overview

The primary purpose of the first year of the contract was to improve the numerical linear algebra routines for use in new Navier-Stokes codes, specifically Tim Barth's unstructured grid code, with spin-offs to TRANAIR. Professor Yousef Saad, one of the world's leading authorities on matrix-iterative methods and the originator of the industry standard GMRES algorithm was used as a subcontractor.

Originally it was intended that all code development work for the new numerical linear algebra algorithms would be done at Boeing under Saad's supervision. However Saad was able to hire a research assistant, Andrew Chapman, to do this work, so some of Boeing's direct participation in the contract was delayed until initial coding of fundamental algorithms was completed. This extra time was devoted by Boeing to write a much needed fast distance calculation routine for Navier-Stokes codes using the new one-equation turbulence models. Key ideas for this distance calculation were motivated by Tim Barth's code. However, work on the distance function was an unexpected bonus, not part of the original contract work statement and thus was not charged to the contract. Even so, the fast distance calculation routine is being made available to grateful Navier-Stokes code developers at NASA.

The primary focus of the first year's work, devoted to improving matrix-iterative methods, was very successful. New algorithms have been developed which activate the full potential of classical Cray-class computers as well as distributed-memory parallel computers. The technology which has been developed is having immediate impact on design processes at Boeing, and is influencing our near term research projects. The results were so good that the project engineers demanded that the new algorithms be implemented in TRANAIR right away. A crash program was instigated to accomplish this just in time for the 747 rewing project. Naturally this activity has received favorable attention from high-level Boeing mangers. In addition the TRANAIR speed ups should be well received by NAS officials who have lodged repeated complaints about how slow TRANAIR runs on the NAS C90.

These excellent results were achieved by having different groups working on similar tasks communicating with each other via e-mail. In this manner "critical mass" was brought together, each group providing a different set of skills and ideas and providing motivation and encouragement to the other groups. This was an almost text book case of how NASA/Industry/University collaboration should work. Specific details follow.

1 Summary of Significant Technical Results

A list of the some of the most significant technical results achieved during the first year of the contract are enumerated here. We will follow with sections which discuss each of these results in detail.

1. Fast Distance Calculation.
2. BILU routines.
3. DGMRES Algorithm.
4. Block GMRES.
5. TRANAIR Speed Up.

As compared to the draft of the final report dated December 21, 1995 but not submitted until January, 1996 because of the government shutdown, the section on TRANAIR Speed Up is new. Only minor corrections have been made to the other sections. The reader may be amused by comparing some of the bold predictions made in the draft of the final report with the reality, as it exists so far, described in the TRANAIR speed up section.

2 Fast Distance Calculation

I will first give a discussion taken from an abstract I submitted for consideration for the upcoming 15th International Conference on Numerical Methods in Fluid Dynamics, to be held June 24-28, 1996 in Monterey. I will then provide an answer to questions brought up during my oral presentation at NASA Ames on December 11, 1995.

Abstract Submitted for Monterey Conference

Recently developed turbulence models such as Baldwin-Barth ([1]) and Spalart-Allmaras ([2]) require the user to compute the distance from each point in the field grid to the configuration under consideration. For calculations involving millions of grid points, naive methods for computing the distance function can easily consume hours of CPU time even on Cray C-90 class computers. These distance calculations are so expensive that some code developers have chosen to avoid performing a proper distance calculation thus imperiling the accuracy and convergence characteristics of their codes. In this paper we wish to discuss efficient methods for computing the distance function. This methodology not only has applications for Navier-Stokes calculations using recently developed one-equation turbulence models, but to grid generation as well.

Naive Algorithm

For each of the NF field grid points the naive algorithm simply calculates the distance to each of the NS surface points and selects the minimum of these distances.
Cost: $NF * NS$

Faster Algorithm

Construct roughly $\sqrt{NS}$ boxes each containing roughly $\sqrt{NS}$ surface points. For each of the field grid points compute distance to each box. Select closest box and compute distance to each surface point in box. If the minimum distance to points contained in closest box is smaller than distance to remaining boxes we are done. Otherwise examine second closest box etc. Experience has shown that on average one must look at 1.5 boxes.
Cost: $NF * \sqrt{NS}$

Construction of Boxes

No doubt there are many methods for constructing the boxes required by the faster algorithm. The method we actually use is as follows:
Start with a big box containing all the surface points. Divide box in longest direction. Choose dividing plane so that half the surface points lie on each side. Proceed recursively. Stop when box has $\sqrt{NS}$ or fewer surface points. As final embellishment, look at minimum and maximum values of coordinates of surface points contained in box to see if box can be reduced in size.
A 2 dimensional example of boxes is shown in figure (1). Of course the concept works just as well in 3D.

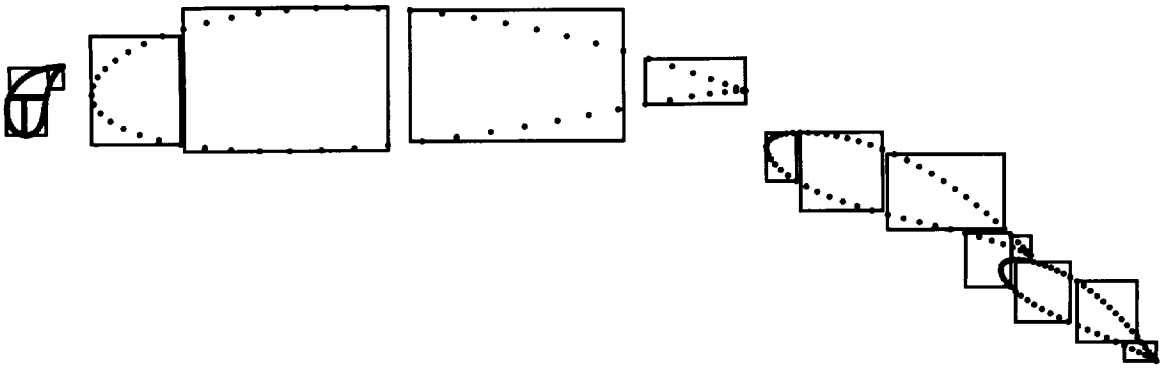

Figure 1: Example of Boxes used in Distance Calculation

Discussion

For a recent Navier-Stokes calculation involving 5 million field grid points and 40,000 surface points, the naive distance calculation required 3 hours on the C-90. The faster algorithm took just 5 minutes!

We are really interested in computing the distance between each field grid point and a triangulation of the configuration. At the present time we locate the closest of the surface points we are given and then examine the nearby triangles. In some pathological cases the closest point is not adjacent to the closest triangle, so we get the wrong result. For each field grid point we really should be computing the distance to each triangle and then take the minimum of these distances. In the full paper we will discuss fast methods for performing this more proper calculation of the distance function.

Questions brought up during Oral Presentation at NASA Ames

During my oral presentation some surprise was registered that so much computer time was being taken with the distance calculation. After all, we are really only interested in the distance function for field grid points in the viscous layer. For these points we need only trace along a grid line back to the surface. We can then compute the distance between the field grid point and the surface point found by tracing a grid line. However it is not always easy to properly account for a non orthogonal grid. More seriously this short-cut procedure does not work in the wake region. I am told that in OVERFLOW the distance function for points in the wake is simply set to some large number (like 1.0e20). Not only is this wrong, it sometimes causes convergence problems with the OVERFLOW flow solver. Also I should point out that in any case, it is difficult to implement a short-cut procedure on an unstructured grid.

Our general dissatisfaction with short-cut procedures for computing the distance function convinced us of the need for performing this calculation in a proper manner. However for very large grids, the most obvious methods for properly computing the distance function were becoming prohibitively expensive (for an n by n by n grid, there are n^3 field grid points and n^2 points on the surface, so naive algorithms have cost proportional to n^5). Thus we are grateful to have the new, reasonably fast algorithm, available to us. As mentioned in the abstract, we realize that there is still more work to be done ensure

the accuracy of the algorithm in pathological cases. Also no doubt there are ways to further reduce the cost of the algorithm (put boxes inside boxes for example). However, in the meantime, we are making our current algorithm available to Navier-Stokes code developers. The new routines have already been placed by Veer Vatsa into TLNS3D and by Bob Biedron in CFL3D.

Tim Barth Connection

Tim Barth's code uses boxes and trees in a manner which is suggestive of the algorithm which is finally employed in our "faster" distance algorithm. Tim may not immediately recognize the precise manner in which I construct and use boxes in my algorithm. Also unlike Tim I take care not to use recursive subroutine calls during the final distance calculation (I found many cases where recursive subroutines worked well on workstations but not so well on Cray computers. I believe that my code will work well on all computer platforms). However I have been working on the distance calculation on and off for a few years, and I did not make significant progress until I saw Tim's code. So it is fair to say that the "faster" algorithm is motivated by Tim' code.

Another aspect of Tim's codes which has proven to be useful is his method for doing memory management. Tim uses his own `my_malloc` and `my_free` routines to allocate and free storage in a manner reminiscent of the malloc and free routines built into C. However the FORTRAN code in `my_malloc` and `my_free` is machine dependent. For workstations they call system routines malloc and free, but on the Cray they call hpalloc and hpdeallc. Also `my_malloc` and `my_free` use common blocks to keep track of overall memory allocation to make it relatively easy for the user to trace down memory leaks. When allocating memory, Tim makes use of pointers. Though not standard, pointers seem to be available with all new FORTRAN compilers. Tim follows a useful construction for pointer names. He takes the name of array he is creating and pre-appends "`p_`" to get the corresponding pointer name.

At any rate, Tim's method for memory management was heavily used by me and proved very useful in modifying and writing codes. The reader may find it instructive to look at the memory management routines in file `memory.f` and the distance calculating subroutine `bbdist.f`.

3 BILU Routines

Yousef Saad wrote preliminary versions of the BILU (Block incomplete LU factorization routines) under NASA contract monitored by Alex Woo. However errors in these codes were detected by comparing with results from the out-of-core sparse library package SPSLIB which is in TRANAIR. Corrected version of the BILU routines were written by myself using some of Saad's ILU routines as templates. Heavy use was made of Tim Barth's memory management methodology. Memory management simplified the calling sequences to the routines because the user no longer had to create and pass in various work arrays. Also since memory could be allocated dynamically by the BILU routines, the user did not have to guess in advance how many entries would be in the approximate factorizations created by the BILU routines.

It is fair to say that Industry, NASA and Universities all played an essential role in the creation of the BILU routines.

We now have both basic types of BILU routines in place, BILU(k) where the fill in locations are essentially pre-specified and BILUT routines where the fill in is determined based on a drop tolerance. The best way to implement the drop tolerance procedure is still being investigated, so all results in this report are based on BILU(k). The reader may find it instructive to look at routines `biluk_mem` and `blusol_mem` in file `bilu_mem.f`.

4 DGMRES

One of the main tasks to be accomplished under this contract was the implementation of the DGMRES (Deflated GMRES) algorithm. The main idea of DGMRES is based on the observation that small eigenvalues tend to slow GMRES. Therefore as GMRES iterations are performed, DGMRES calculates small eigenvalues and corresponding eigenvectors. These eigenvectors are added to the Krylov space.

Implementation of DGMRES was carried out by Andrew Chapman under the direction of Yousef Saad. Corrections were made in response to errors found when I ran DGMRES through some test cases. One of the most notable errors was caused by the fact that subroutine RG in EisPack does not normalize the eigenvectors it finds. Thus repeated calls to RG led to arithmetic underflows.

For technical details of the DGMRES algorithm the reader is referred to the paper ([3]) written by Chapman and Saad. In this section we will simply exhibit the ability of DGMRES to solve some test matrices supplied by Tim Barth associated with 2D unstructured-grid Navier-Stokes calculations.

Tim adds his one-equation turbulence model to the 4 flow equations (conservation of mass, momentum in the x and y directions, and energy). Thus Tim has 5 equations for each point in the grid. Also Tim usually solves equations associated with a second-order accurate discretization but forms the preconditioner from a first-order accurate discretization. Thus for the purpose of these tests Tim supplied a second-order matrix which we were supposed to use for matrix-vector products and a first-order matrix from which we were to construct the preconditioner. Tim supplied matrices for 2 different grids. The "small" grid with 3,147 points gave rise to the first-order matrix SDIST1 and

second-order matrix SDIST2 with:

Name	n	nnz
SDIST1	15,735	498,620
SDIST2	15,735	1,105,164

Table 1: Characteristics of "small" grid Matrices

where n is the number of rows in the matrix and nnz is the number of non zero elements. The "big" grid with 37,874 points gave rise to the first-order matrix BDIST1 and second-order matrix BDIST2 with:

Name	n	nnz
BDIST1	189,370	6,260,236
BDIST2	189,370	13,848,412

Table 2: Characteristics of "big" grid Matrices

In all the tests shown in this section we either run GMRES(50), which is GMRES with 50 search directions, or DGMRES(50,5) which is DGMRES with 50 search directions of which the last 5 are approximate eigenvectors associated with the smallest eigenvalues. The eigenvector information calculated by DGMRES becomes more and more accurate as the solution process proceeds. The CPU time required by DGMRES to calculate the eigenvector information is essentially negligible, while the storage required by DGMRES(50,5) is approximately $55n$ (where n is the order of the matrix), while that required by GMRES(50) is approximately $50n$.

Tim usually solves his second-order matrices using a preconditioner formed by applying BILU(0), mb=5 to the first-order matrix. Here mb refers to the size of the blocks used in the BILU decomposition. A natural choice is mb=5 since there are 5 equations per grid point. Unfortunately I found that BILU(0), mb=5 did not work well on the SDIST2, SDIST1 matrix pair. I found much better convergence when I used BILU(0), mb=10. However as shown in figure (2) even with this more powerful preconditioner in place, GMRES(50) is quite slow to converge. Indeed after 250 iterations, GMRES(50) is only able to reduce the residual 2 orders of magnitude. The eigenvector information available to DGMRES(50,5) is quite valuable in this case. The first time we apply DGMRES(50,5) the eigenvector information is beginning to be effective by the end of the third cycle (iteration 150). By the end of the fifth cycle (iteration 250) the residual has been reduced more than 7 orders. I then ran DGMRES again, retaining the eigenvector information gained during the first run with DGMRES. In this case convergence is helped during the very first cycle. By the end of the second cycle the error is reduced 4 orders of magnitude, and 8 orders is achieved in only 180 iterations.

This example is intended to show that as the iteration process progresses, DGMRES gains more information about the matrix which it is able to apply to aid convergence.

This is very useful in cases where we are solving matrix problems with multiple right hand sides, or in iterative processes where we do not update the matrix.

In his actual code, Tim applies a CFL correction to his matrices. The CFL correction adds a term inversely proportional to the CFL number to the diagonal entries of both the first and second-order matrices. This damps the overall Newton iteration and makes the matrices easier to solve. I simulated this effect by calculating the spectral radius of each of the 5 by 5 block diagonal matrices associated with the first-order matrices. I took this spectral radius, divided by the CFL number and added this to the diagonal entries of the block diagonal matrices in both the first and second-order matrices.

As shown in figure (3) the CFL correction allows us to solve the SDIST2 matrix using a preconditioner formed by applying BILU(0), mb=5 to SDIST1. In this case we have no need to resort to mb=10 when forming the preconditioner. Also, as expected, convergence is quite a bit faster for CFL=100 than it is for CFL=1000. In these calculations we are essentially duplicating the methodology in Tim's code except that we are using DGMRES in place of GMRES. It was noted that the advantage of using DGMRES in place of GMRES increased as the CFL number increased. However at extremely large CFL numbers even DGMRES was not able to ensure convergence when solving SDIST2 using BILU(0), mb=5 on SDIST1 to form preconditioner.

The big Barth matrix BDIST2 was considerably tougher to solve then SDIST2. In order to obtain satisfactory convergence it was necessary to use CFL=100 and even then I had to use mb=10 when forming the preconditioner (from BDIST1). Convergence histories using BILU(k), mb=10 for k=0,1, and 2 are shown in figure (4). Convergence is satisfactory and it does (slowly) get better as we increase k. As shown in figure (5) when CFL is increased to 1000 even forming the preconditioner by applying BILU(2), mb=10 to BDIST1 did not lead to satisfactory convergence. In this case DGMRES(50,5) was run twice. In the second run DGMRES used information gained in the first run. The convergence rate in the second run is better than in the first run, but as I said, still not satisfactory.

It is of some interest to see if DGMRES plus the preconditioner we are using is good enough to solve the BDIST1 matrix. In figure (6) we apply DGMRES(50,0) (this is the same as GMRES(50)) and then make 2 runs with DGMRES(50,5) to solve BDIST1 at CFL=10000 using BILU(2), mb=10 preconditioning. The convergence with pure GMRES may not be deemed satisfactory but the convergence with DGMRES certainly is especially for the second run which makes use of information gained in the first run. I should mention that as we reduce the CFL number to say 1000 or 100, the convergence becomes much faster yet. The point to this example is that even if we do have a good preconditioner for the BDIST1 matrix, it may not lead to satisfactory convergence for the BDIST2 matrix.

Tim claims that he gets satisfactory convergence for these type of matrices using pure GMRES with BILU(0), mb=5 preconditioning. We (me and Andrew Chapman) have Tim's BILU(0) subroutine and have verified that is behaves the same way as my BILU(k) routine using k=0. This raises the possibility that there is an error in the matrices Tim gave us. In the first matrix examples Tim gave to Andrew the grid had a collapsed cell. Other errors are possible (for example not normalizing the matrix by the local cell volume). This issue will have to be investigated further.

However none of this detracts from the conclusion resulting from a battery of tests

that DGMRES represents a very worth while advance over the pure GMRES algorithm we now typically use in our CFD codes.

5 Block GMRES

During my oral presentation I made the statement that I did not mention Block GMRES in my contract proposal. This is wrong, it is discussed in section 3.6. However it is fair to say that at the time I wrote the proposal, I intended to focus my attention on eigenvalue translation/deflation, my ideas on how to proceed with Block GMRES were not well formulated. However in the last few months Block GMRES did indeed attract my attention and the results turned out to be quite exciting.

The idea behind Block GMRES is that when solving problems with multiple right hand sides one can combine the Krylov spaces associated with all the right hand sides. That is, for each right hand side, one finds the best solution over the space spanned by all the Krylov spaces, not just the Krylov space associated with that particular right hand side. As compared with standard GMRES, Block GMRES involves more sdot and saxpy operations to keep all the vectors orthogonal to each other but it reduces the number of preconditioned matrix vector multiplies, typically by a factor of 2-10 (factor of 2 for easy cases and factor of 10 for hard cases). In terms of CPU time this is a big win for our CFD applications because the pre-conditioned matrix vector multiply operations dominate the cost. However to take advantage of Block GMRES one must be willing to simultaneously store all the Krylov vectors.

While giving a series of lectures ([4]), Yeremin claimed that the synergism from using multiple Krylov spaces is so powerful that given a problem with one right hand side, it is a good idea to artificially add other right hand sides and use Block GMRES in place of GMRES! I have not yet tested this theory but I did give Yeremin some TRANAIR test matrices and right hand sides associated with design calculations. I wanted to compare the performance of Yeremin's BGMRES routine with DGMRES. In order to properly do these comparisons I gave Yeremin the matrix-vector multiply routine as well as corrected versions of Saad's ILU routine and Saad's forward-backward substitution routine "lusol". I had to make sure that the runs Yeremin made with his BGMRES code could be directly compared with runs I made with DGMRES. In terms of operation count and CPU time BGMRES proved to be significantly better than DGMRES. However if storage really is at a premium one may still prefer to use DGMRES which requires very little more storage than standard GMRES.

Upon hearing the results of comparisons between Yeremin's BGMRES code and Chapman's DGMRES code, Yousef Saad gave me source code to his own Block GMRES subroutine `bgmr`. In addition Saad gave me PostScript files `block1.ps` and `block2.ps` which contain sections of his new book ([6]). These sections discuss Block Krylov methods.

In terms of the number of Krylov vectors required to solve the TRANAIR test problems Saad's `bgmr` routine did just as well as Yeremin's BGMRES routine. The CPU issue is less clear. Saad's code was faster than the first results given to me by Yeremin. Subsequently Yeremin started quoting faster and faster CPU times even though the number of Krylov vectors was not changing. Finally he was quoting total CPU times which were

less than would have been required to do preconditioned matrix-vector multiplies using the subroutines I gave him.

I know that it is possible to significantly speed up the preconditioned matrix-vector multiply routines. For example, by using longer vector lengths afforded by a "Jagged-Diagonal" format as discussed in ([5]) it is possible to speed up the preconditioned matrix-vector multiply operations by a factor of 3 on Cray computers as compared with the routines I gave Yeremin. (During my oral presentation Alex Woo indicated that he was familiar with the jagged-diagonal format, so I will not go into details. However I should mention that I did not know about the jagged-diagonal format until this year and it is not yet been exploited by the TRANAIR sparse matrix library SPSLIB). A further factor of 2 is achievable on Cray computers by processing multiple right hand sides inside the loops (reducing memory operations on matrix elements and elements in the ILU factorization). Yeremin refused to give me the results of Flowtrace so that I could see how much time was being spent in the various subroutines, he simply gave me the total CPU time. However if I support Saad's `bgmr` code with jagged-diagonal routines and process multiple right hand sides inside the inner loops, I can easily match Yeremin's total CPU times.

Of course it is possible to further improve Saad's `bgmr` routine by adding DGMRES and by dropping unneeded Krylov spaces (some right hand sides are solved faster than others). However the TRANAIR team was already so impressed with `bgmr` that they wanted me to create an out-of-core version and put it into TRANAIR right away. I took time off from the contract to accomplish this task during the two week period preceding my oral presentation.

Naturally exploiting the jagged-diagonal format and processing multiple right hand sides inside loops will be the next TRANAIR project. Overall, for design calculations involving many right hand sides, we are expecting an order of magnitude improvement over the standard version of TRANAIR which relies on pure GMRES to solve one right hand side at a time and uses row-wise storage of the matrix and ILU factorization. These algorithmic improvements are quite significant to Boeing because we use a very large amount of Cray CPU time running TRANAIR for our new airplane designs. These algorithmic improvements promise to be even more significant than the hardware improvement realized when we replaced our Y/MP with a Triton.

The Block GMRES concept also lends itself to coarse grained parallelism. Certainly the preconditioned matrix-vector multiplies for multiple right hand sides can be done in parallel. On Cray class computers the idea of placing multiple right hand sides inside inner loops during matrix-vector multiplies (or forward-backward substitutions) so as to minimize I/O activity with the matrix elements saves about a factor of 2. Saad was surprised that it was only a factor of 2, but this is because Cray computers have such a good memory subsystem supported by SSD so it is not so very important to reduce I/O activity on Cray class computers. I have a discussion of this point in the next section. On other types of computers the reduced I/O activity (whether between disk and memory or between memory and cache) is likely to produce much greater savings.

Compared with pure GMRES, using Block GMRES to solve problems with multiple right hand sides reduces the overall CPU time, because fewer pre-conditioned matrix-vector multiply operations are needed, and these operations can be performed more efficiently because I/O activity with matrix elements can be reduced (this latter effect is

most important for non Cray class computers). However, as already mentioned, the price that must be paid for using Block GMRES is that one must be willing to simultaneously store multiple Krylov vector spaces.

For a real-life TRANAIR design problem solved on the Triton with 500,000 unknowns, pure GMRES required 50 Krylov vectors to solve each right hand side. When I used `bgmr` to solve the problem 10 right hand sides at a time, only 23 Krylov vectors per right hand side were required. The CPU time was reduced by a factor of 2 (much more than this will be saved when we add jagged diagonal) but the required storage increased by 90Mw. Since we typically use 200Mw of SSD for TRANAIR runs anyway, we do not consider an additional 90Mw to be overly burdensome considering the CPU savings (especially since more CPU savings are in the offing).

Non Cray computers do not have SSD so we may have to use disk instead. However Yeremin claims that with his BGMRES code he can get by with using disk even on Cray class computers with no noticeable degradation in performance. Also Yeremin claims to solve 3000 right hand sides at a time. With this many right hand sides the orthogonalization costs in `bgmr` would be prohibitive. Yeremin's claims indicate that the Block GMRES methodology can be implemented in a manner which minimizes orthogonalization costs and permits one to use disk even on a Cray class computer. This indicates that there is still much more we can learn about Block GMRES methodology. In a latter section I will discuss my thoughts on supporting Block GMRES with disk.

Multiple Right Hand Sides Inside Loops

This is part of an e-mail message I sent to Saad concerning the impact of processing multiple right hand sides inside loops on the Cray.

We always say that the Cray is a vector machine, but it really is a pipeline machine. It is able to pipeline:

```
2 fetches from memory
1 addition
1 multiply
1 store to memory
```

After an initialization cost of starting this pipeline it is able to produce the results in one clock period per result. Cray refers to all the operations which can be pipelined together as a chime, and they like to measure the execution cost of executing a vector loop in terms of chimes. So if we look at the loop (this is the type of loop which one encounters when doing a matrix vector multiply using a jagged-diagonal format):

```
      do 10 i=1,n
      k=ip(i)
      y(i)=y(i)+a(i)*x(k)
10    continue
```

In one chime the Cray can fetch ip(i) and a(i). At this point it can not do any multiply or add operations, so this is all that happens during the first chime. During the second chime the Cray will fetch y(i) and x(k) (2 fetches), multiply a(i) by x(k), add to y(i) and store to y(i). During the second chime it actually does 2 fetches a multiply, an add and a store. Bottom line is the single right hand side loop does 2 floating point operations in 2 chimes.

A multiple right hand side loop:

```
      do 20 i=1,n
      k=ip(i)
      y(i,1)=y(i,1)+a(i)*x(k,1)
      y(i,2)=y(i,2)+a(i)*x(k,2)
      y(i,3)=y(i,3)+a(i)*x(k,3)
      y(i,4)=y(i,4)+a(i)*x(k,4)
20    continue
```

We do 8 floating point operations in 5 chimes. Thus multiple right hand side loop with 4 right hand sides handled simultaneously is 1.6 times as fast as single right hand side mode.

If we take SSD transfer of matrix information of "a" and "ip" into account, then we have to add 2 chimes to each of these loops (one chime to read a, one chime to read ip, we will store x and y in core). In this case the do 10 loop does 2 floating point operations in 4 chimes, while the do 20 loop will do 8 floating point operations in 7 chimes. In this

case multiple right hand side is more efficient by a factor of $(8/7)/(2/4) = 32/14$ which is why I say placing multiple right hand sides inside inner loops is worth a factor of 2.

In this e-mail message I should have mentioned that asymptotically for a very large number of right hand sides placed inside the inner loop we can save a factor of 3 on Cray class computers, but of course this depends on having a lot of memory. Also for non Cray class computers (no SSD) the effects of reducing the I/O operations will be more significant.

Reducing I/O for orthogonalization operations

A copy of part of another e-mail message I sent to Yousef Saad concerning I/O associated with orthogonalizing a set of vectors.

I think I figured out how Yeremin supports BGMRES with disk on the Cray. The disk, using striping, is about 20 times slower than the SSD (on the Triton SSD transfer rate is about 600Mw/sec, versus 30Mw/sec transfer rate for disk with striping). However if we work on 2 groups of k vectors at a time for the orthogonalization operations (sdot and saxpy), reading a new group requires reading k vectors but then we will have to do k^2 sdot and saxpy operations. Ratio of floating point operations to read operations is $k^2/k = k$. If we use SSD then $k = 1$ works fine as verified by our pure GMRES runs on the Cray. If we use disk to support Block GMRES then $k = 20$ should work fine.

This method would require us to store 40 vectors simultaneously in core. On the Cray, we could use SSD to reduce the in core storage required. One possible strategy is to pass vectors in groups of 20 between disk and SSD and in groups of say 5 from SSD to memory. In this way the memory only has to hold 10 vectors at a time, SSD only has to hold 40 vectors at a time and disk can hold as many as needed. The main point is that even if we happen to use thousands of Krylov vectors with Block GMRES, these can be stored on disk. The amount of SSD required will be no more than that used by pure GMRES.

Even if we can not hold $2 * k$ vectors each of length n in core at one time, then we can split the vectors up into sections of length ns. In this case we would read in $2 * k * ns$ numbers associated with the first section of each vector. We would then do the parts of the sdot operations associated with the first section of each vector. We then read in the second section from each vector and so on. We would then have to make a second pass to do all the saxpy operations. Reading vectors in sections would require 4 times the number of read operations of the first method, but should still work efficiently using disk if we can take $k = 4 * 20 = 80$.

In the above message to Saad I was talking about using Cray computers, but it applies equally well to the SP2. In fact, one of the problems in porting TRANAIR to the SP2 is that the I/O associated with orthogonalizing vectors for GMRES (in this case passing information between processors) dominates the cost of the floating point operations. Using a block version of GMRES with the strategy discussed above can overcome this problem. Thus the Block GMRES idea gives us a way to take full advantage of both Cray class computers and distributed memory parallel computers.

Saad responded that the ideas I presented are very much in the spirit of the BLAS3 package where one trys to efficiently support floating point operations with a hierarchical memory. Saad points out that a detailed discussion of orthogonalization operations using hierarchical memories appears on pages 55 and 56 of the book ([7]).

6 TRANAIR Speed Up

TRANAIR is the primary design/optimization code used at Boeing. The cost of running TRANAIR in design mode is dominated by the cost of solving matrix problems with multiple right hand sides. These multiple right hand side problems arise because TRANAIR must compute the change in the solution caused by changing each of the design variables (so called "sensitivity" calculation). Thus the number of right hand sides depends on the number of design variables used to represent changes in the geometry.

In order to properly represent an entire wing we would like to use about 500-1000 design variables. However even large 50000 second runs on our newly acquired Triton only allow us to use 100-200 design variables for transonic optimization runs. With the all important 747 rewing effort coming up, the project people were very interested when they discovered that as a result of this NASA contract I had some technology in hand which could significantly reduce the cost of running TRANAIR.

The process of upgrading TRANAIR was high risk because TRANAIR is a complex out-of-core code, and the modifications had to be made very quickly/correctly in a high pressure environment. The modifications were made step by step and thoroughly tested after each step.

Naturally the first improvement was to add `bgmr` in an effort to reduce the number of iterations required to solve the problems as compared to using pure GMRES. As expected `bgmr` did make a substantial difference for cases which were hard for the pure GMRES algorithm. Sometimes `bgmr` by itself improved convergence by a factor of 6. These were usually transonic cases involving boundary-layer coupling. Very easy cases such as supersonic HSCT cases were helped very little, but this hardly matters because pure GMRES was already converging in only 4 iterations for these cases anyway.

The other changes were aimed at reducing the cost of performing iterations and to reduce I/O. The reduction in I/O has obvious advantages on SP2 class computers but could also be helpful on the CRAY because it could allow one to use disk in place of SSD for very large calculations. At any rate, jagged diagonals were added to the matrix-vector multiply routines. On the Triton this is a factor of 4 faster than the standard routine based on a row formatted storage scheme. It is also more than a factor of 2 faster than the "fast" matrix-vector multiply routine which was recently introduced into TRANAIR by Forester Johnson. An in core code which compares these different methods for matrix-vector multiply is available on request.

As a first step towards putting jagged diagonals into the backward/forward substitutions (equivalent to triangular matrix solves) we did introduce a column format to replace the row format. Since short saxpy operations are faster than short sdot operations on the CRAY we found that the new triangular matrix solves are 35%-40% faster than the old row formatted triangular matrix solves.

For both the matrix-vector multiplies and the triangular matrix solves we do process multiple right hand sides inside inner loops. I was expecting this to improve performance by a factor of 2. In fact, using the cf77 compiler the performance was improved by only 25%-33%. A Cray representative agreed that we should have seen at least a 50% improvement and suggested that I try the new f90 compiler. Cray Research has been focusing most of its recent attention to the f90 compiler since it is supposed to replace

cf77. Much to our surprise, f90 performed even worse than cf77! This was not supposed to happen, so my coding examples have been brought to the attention of the compiler writers. In the mean time it looks like if we are to derive full CPU reduction benefit from placing multiple right hand sides inside inner loops we must resort to CAL.

Processing multiple right hand sides simultaneously has of course reduced I/O activity. The old version of TRANAIR had to read in the matrix and its decomposition from SSD each time it performed a GMRES iteration on one right hand side. The parallel version of `bgmr` allows us to reduce this I/O activity by a factor of *nrhs*, the number of right hand sides we choose to solve simultaneously. Actually the gain is more than this because `bgmr` takes fewer iterations to converge than does pure GMRES. To complement the reduced I/O activity associated with reading in the matrix and its decomposition, I have also implemented the suggestions in the section on reducing I/O for orthgonalization operations. Actually it turns out that instead of doing orthogonalizations block against block one can simply hold the *nrhs* newly created vectors in core, and then read the other Krylov vectors from SSD one by one. Each time a Krylov vector is read in, *nrhs* saxpy and sdot operations are performed to orthogonalize the *nrhs* new vectors against this old Krylov vector. After all this the *nrhs* new vectors are orthonormalized against each other. At any rate, compared with the original modified Gram-Schmidt subroutine the parallel version reduces I/O by a factor of *nrhs*.

Giving TRANAIR the ability to process multiple right hand sides simultaneously required writing "parallel" versions of key subroutines, the first step towards an effective port to the SP2.

The process of upgrading TRANAIR is still on going. In the near future we plan to add eigenvector deflation to `bgmr` and to add jagged diagonal to the forward/backward substitutions. However enough has already been accomplished that it has met with the enthusiastic approval of the project engineers. Speaking conservatively, for the 747 rewing runs, the number of iterations for solving multiple right hand side problems is being reduced by a factor of 2. In addition the CPU operations are being speeded up by a factor of 2 so we are getting an overall factor of 4 reduction in CPU time for the multiple right hand side problem. Again, speaking conservatively, the multiple right hand side problem accounted for 70% of the TRANAIR CPU time during a design run. Thus the improvements we are making are already reducing the cost of a complete TRANAIR design run by a factor of 2. All this has attracted the attention of upper-level Boeing managers. A factor of 2 improvement is all we got when we replaced the Y/MP with the Triton!

7 Significance of Results Obtained During this Contract

During the first year of this contract, emphasis was placed on algorithm development. This was done because an opportunity to meaningfully improve our algorithms presented itself. Also this strategy guaranteed that something useful would come out the contract should it not be continued into the second and third year (a very prudent precaution in these uncertain times). From the Boeing perspective the algorithms produced during the first year of the contract are indeed useful. This is evidenced by the successful crash effort to put them into TRANAIR right away. We are fortunate that what is supposed to be a 3 year contract produced high profile tangible results during the very first year. This is something the Boeing managers always ask for, but we are seldom able to deliver.

Although this research activity was motivated by the desire to improve the pure GMRES methodology in Tim Barth's unstructured grid Navier-Stokes code, the new DGMRES and Block GMRES algorithms have proven so effective that they were incorporated immediately into TRANAIR for use in our on-going airplane design processes. Moreover, in the near term, we plan to engage in follow up research to improve these algorithms. We fully anticipate much needed order of magnitude improvements in the numerical linear algebra routines used in our TRANAIR airplane design code.

The fast distance calculation was an unexpected bonus which has already been incorporated in our Navier-Stokes codes at Boeing and is being given to Navier-Stokes code developers at NASA. Veer Vatsa supplied the following testimonial:

```
Dear Larry,

 These are some of the findings from full-fledged run using your latest
distance function for an HSCT wing/body configuration:

 1. The new procedure took about 150 secs of cpu on YMP vs. 2000 secs.
with the old procedure.
 2. The integrated force and moment coefficients are within 0.2% of each other
 3. The convergence rate was unaffected

 I am asking Ed Parlette to look into converting the C-routines into Fortran
so that I can have a pure Fortran code, which is easier for me to maintain
and port to other machines. I have also passed these routines to
Chris Rumsey and Bob Biedron, our CFL3D group.

 Thanks again for sharing this breakthrough with us. It will help promote
the use of S-A model in CFD codes.

Sincerely,
Veer Vatsa
```

I believe that the real value of the fast distance calculation lies in the fact that Navier-Stokes code developers will no longer have to be tempted by "short-cut" procedures which

jeopardize the accuracy and convergence characteristics of thier codes.

As for the main thrust of the effort devoted to enhancing GMRES, I believe that both DGMRES and Block GMRES will be very well received by the CFD community. As was shown with the Tim Barth test matrices, DGMRES often offers significant convergence improvements over the commonly used pure GMRES algorithm. Moreover the benefits provided by DGMRES require very little more in terms of storage and CPU time than pure GMRES. When solving problems with multiple right hand sides, Block GMRES seems to be even more effective than DGMRES. Block GMRES does require considerable storage to simultaneously store Krylov vectors associated with each right hand side. However Block GMRES offers much opportunity for coarse grained parallelism, and strategies can be devised which allow one to efficiently use disk for the Krylov vectors even on Cray class computers.

8 What about Follow on Contract

Much has happened in the 2 years since the contract proposal was submitted. It was our original intention to use Steve Allmaras and Mark Drela in the second year of the contract to further development of the Tim Barth code. However because of unexpected pressure on man power, Boeing engineers are forced to focus their attention on projects of near term benefit to the company.

In order to convert Tim Barth's code into a practical engineering tool we would have to make advances in grid generation, upgrade the flux routines to handle non-tetrahedral meshes, and implement a transpiration boundary condition or some sort of grid adjustment strategy for design. It is not clear that all these items could be accomplished in the near future with the resources we have available under this contract. We have not given up on the idea of working on an unstructured grid Navier-Stokes code. Indeed Jou is planning to ask upper level Boeing managers for funding to support a critical mass of Boeing engineers to work with outside people to work on just such a project.

In the mean time it would be a good strategy to have the contract follow up on the first-year successes by supporting our effort to parallelize the enhanced version of TRANAIR. NASA is already supporting a contract for a straight port of the standard version of TRANAIR to the SP/2. However the new technology developed as part of this contract, the Block GMRES algorithm in particular, gives us a better opportunity to take advantage of parallel computers. Boeing is well motivated to pursue this project because TRANAIR is our engineering design tool of choice, and we are constantly looking for ways to improve TRANAIR and increase the number of design variables we can use. NASA's support of this project would allow it to share in the advanced technology which will naturally result.

References

[1] Baldwin, B. S., and Barth, T. J., "One-equation turbulence transport model for high Reynolds number wall-bounded flows," AIAA paper 91-0610.

[2] Spalart, P. R., Allmaras S. R., "A One-equation turbulence transport model for Aerodynamic Flows," AIAA paper 92-0439.

[3] Andrew Chapman, and Yousef Saad, "Deflated and augmented Krylov subspace techniques," October 10, 1995.

[4] Alex Yeremin, "Recent Advances in Iterative Solution Methods", notes passed out at lectures presented at Embassy Suites Hotel in Santa Clara, November 8-9, 1993.

[5] Anderson and Saad "Solving Sparse Triangular Systems on Parallel Computers," International Journal Of High-Speed Computing, vol. 1., num. 1, pp. 73-95 (1989).

[6] Yousef Saad, "Iterative Methods for Sparse Linear Systems", Book published by PWS Publishing Company, January 1996. Information available at http:/www.cs.umn.edu/ saad/book.html

[7] K. A. Gallivan and M. T. Heath and E. Ng and J. M. Ortega and R. J. Plemmons and C. H. Romine and A. H. Sameh and R. G. Voigt, "Parallel Algorithms for Matrix Computations", book published by SIAM, 1990.

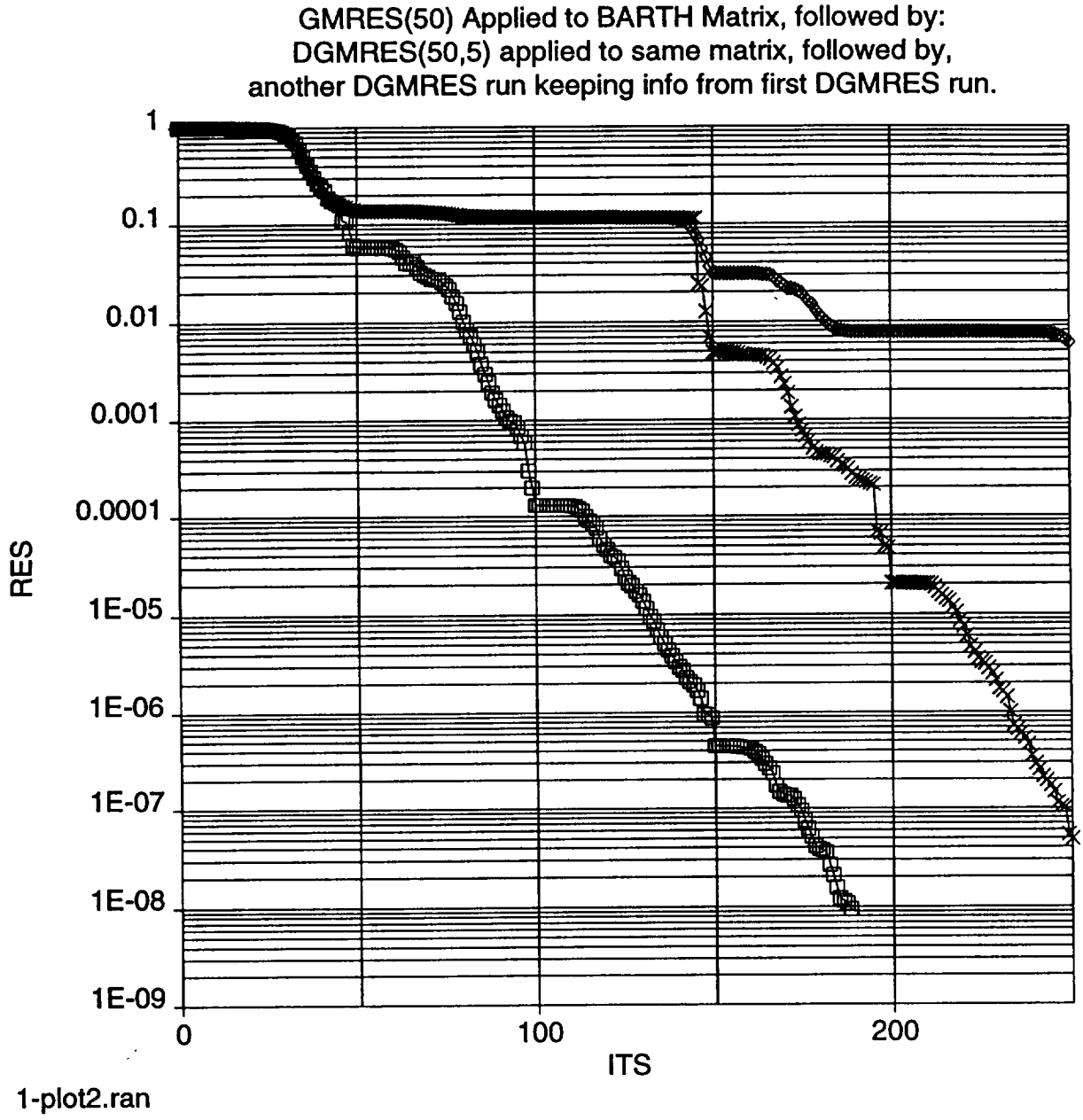

Figure 2: Solving SDIST2 using BILU(0), mb=10 applied to SDIST1 as preconditioner

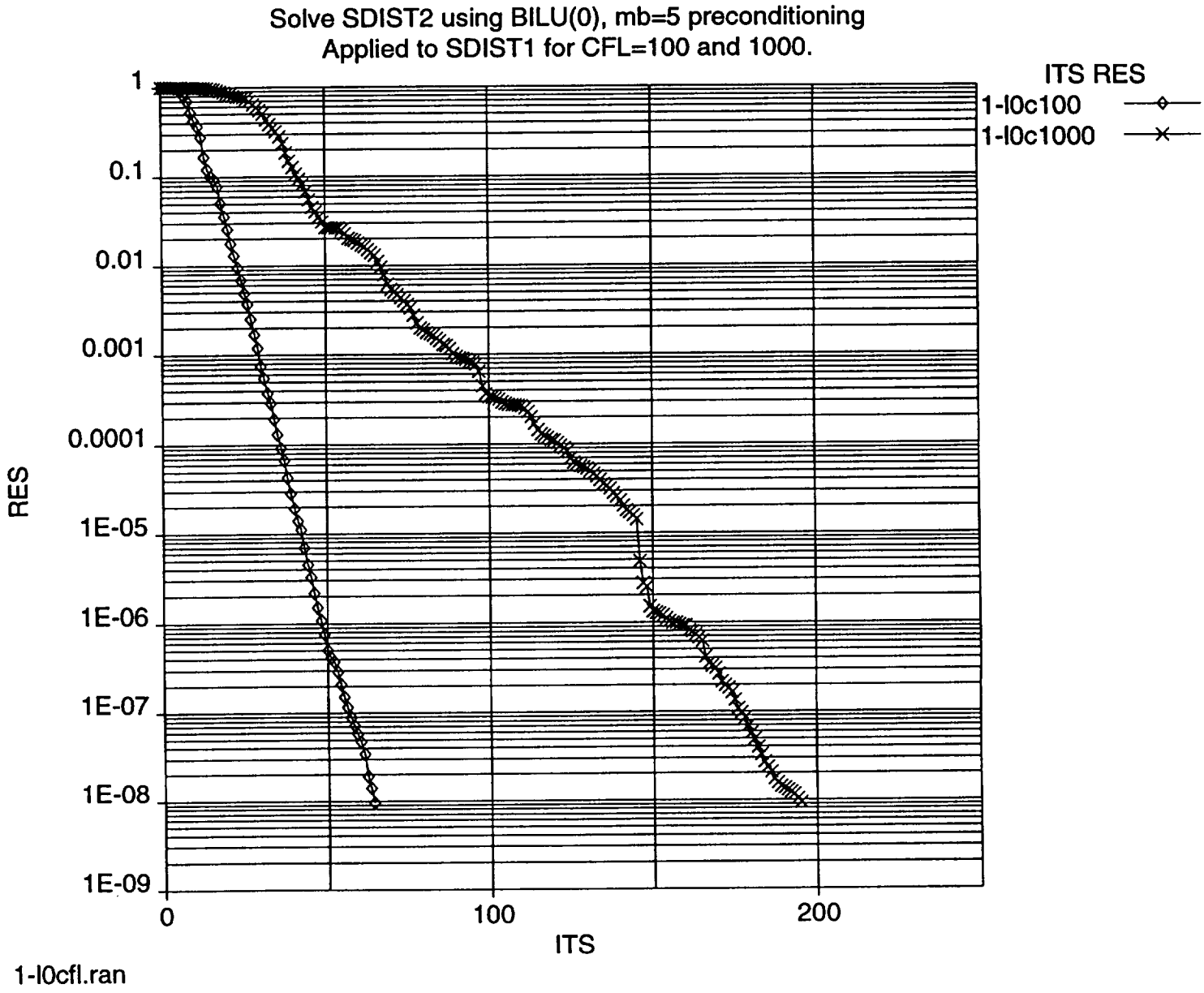

Figure 3: The effect of CFL in solving SDIST2 using BILU(0), mb=5 applied to SDIST1 as preconditioner

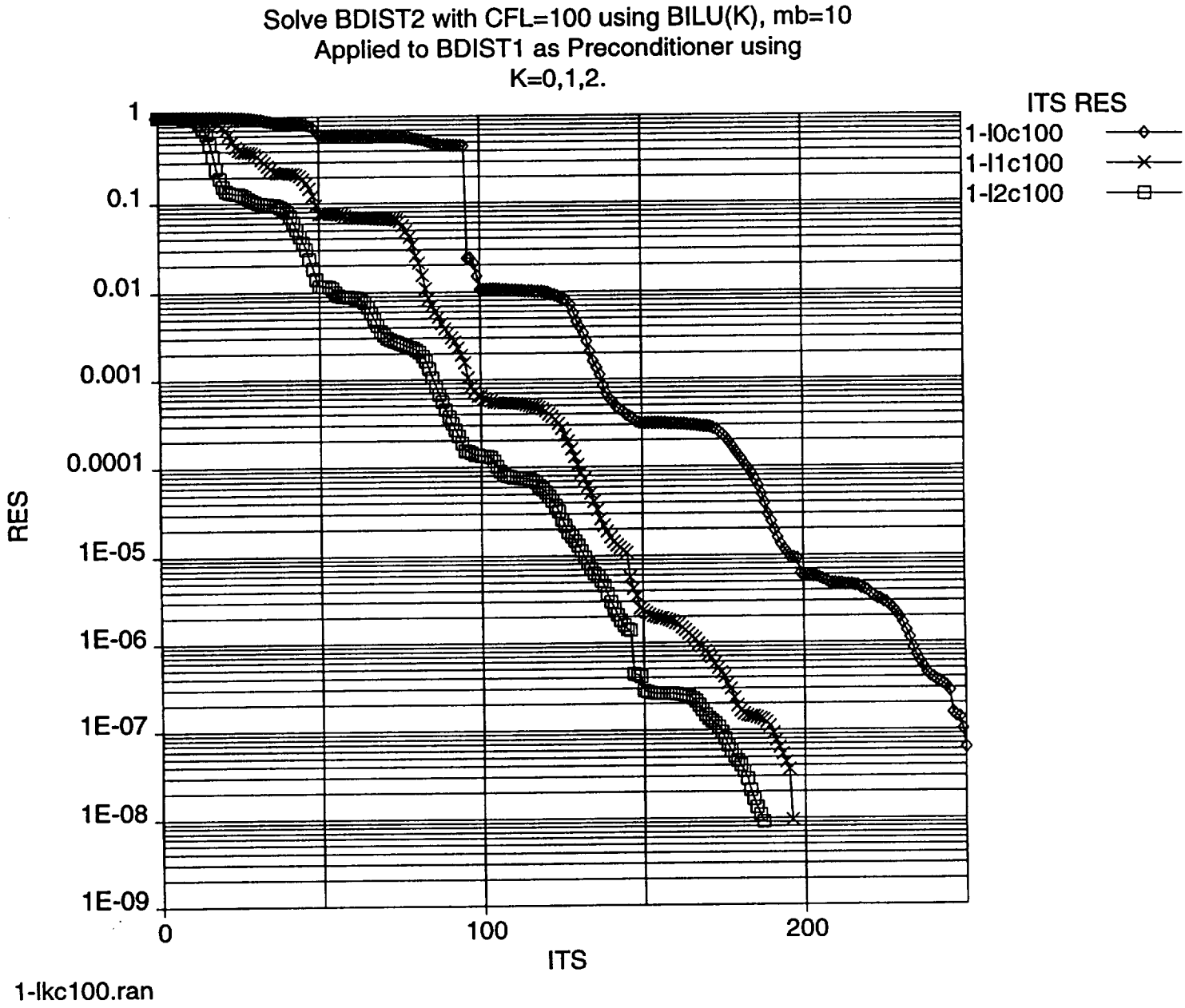

Figure 4: The effect of k in solving BDIST2 using BILU(k), mb=10 applied to BDIST1 as preconditioner at CFL=100

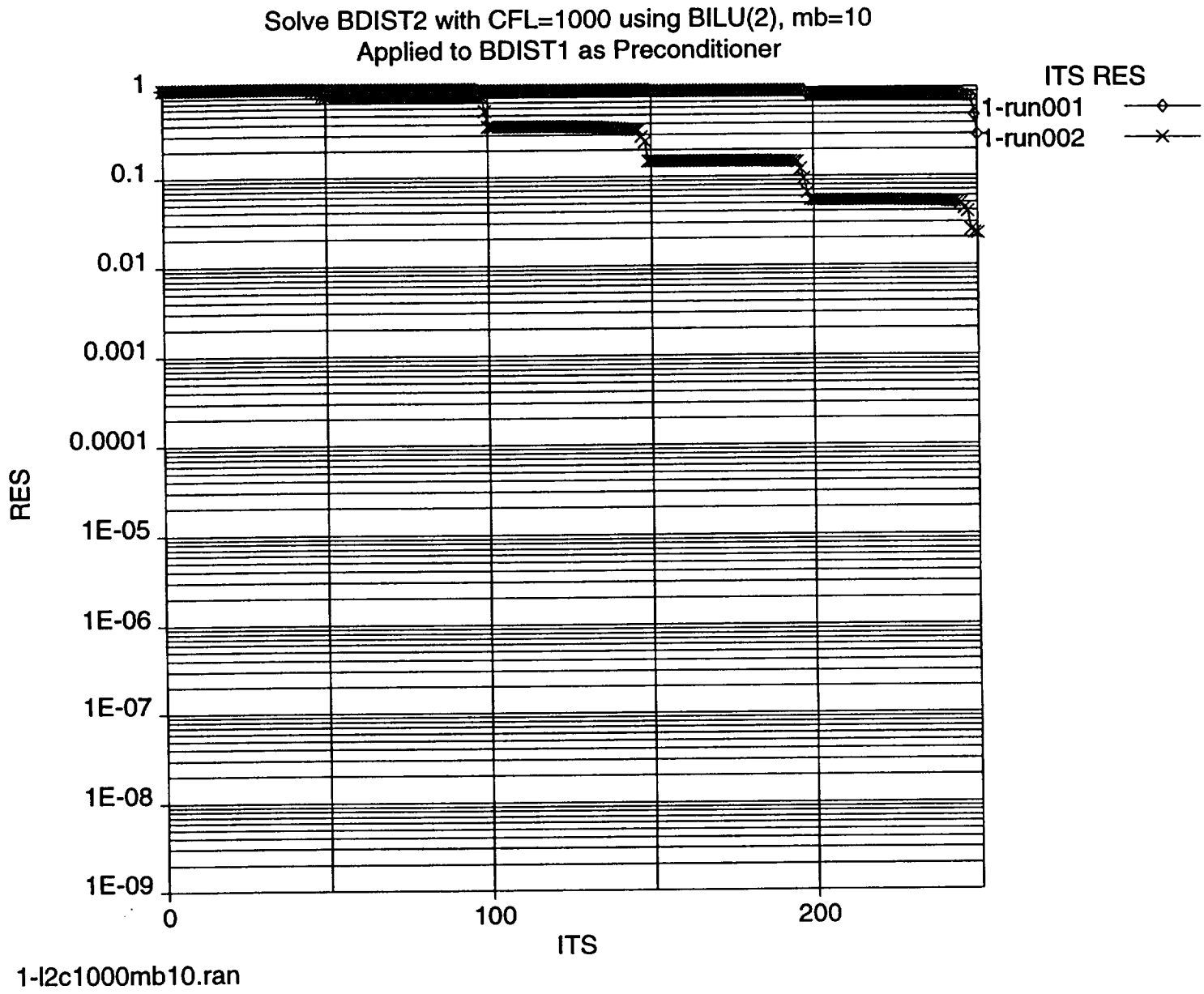

Figure 5: Attempt to solve BDIST2 at CFL=1000 using BILU(2), mb=10 applied to BDIST1 as preconditioner. Use DGMRES(50,5) twice. DGMRES does slightly better the second time around.

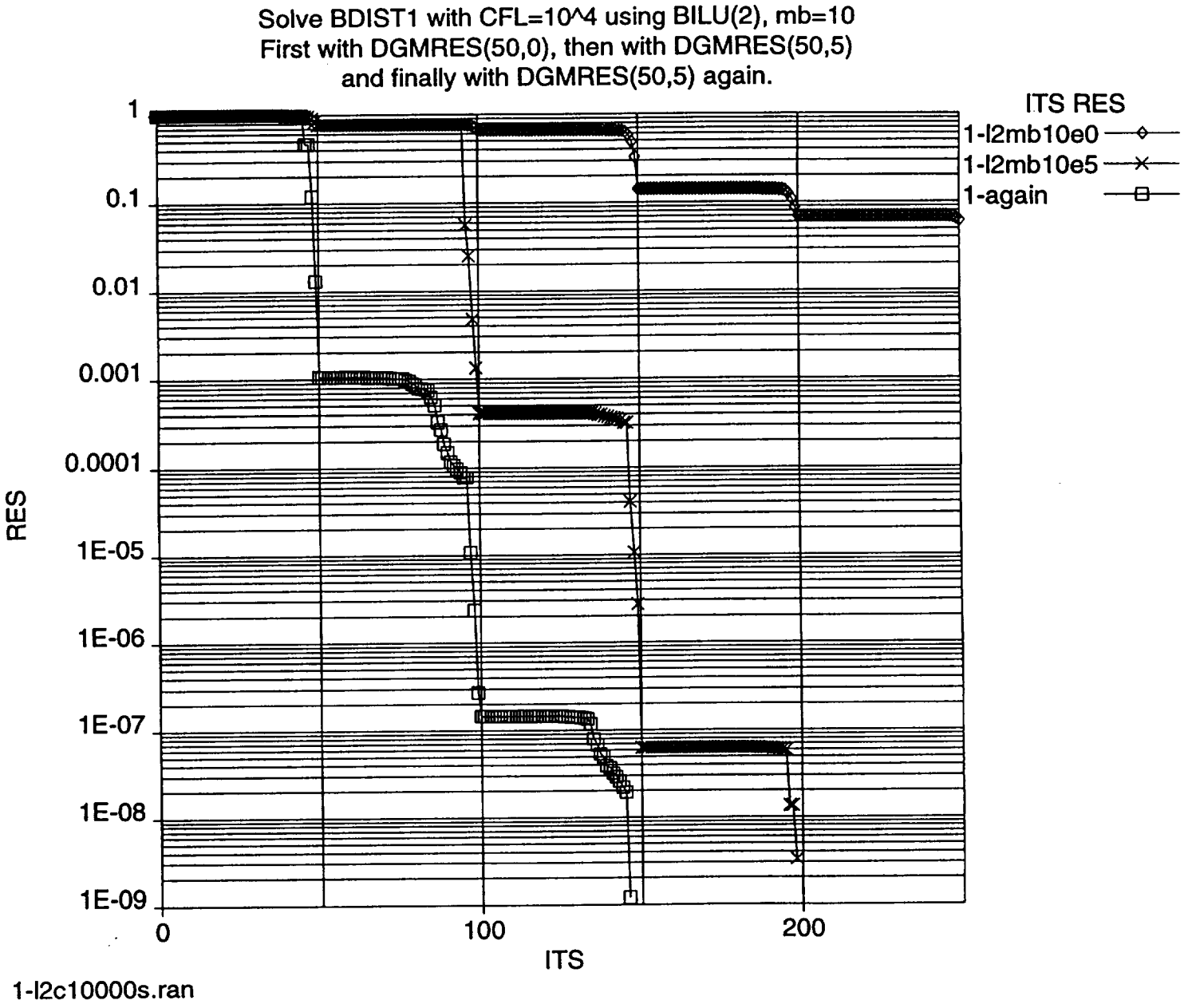

Figure 6: Solve BDIST1 at CFL=10000 using BILU(2), mb=10 applied to BDIST1 as preconditioner. Use pure GMRES then try DGMRES(50,5) twice.