HPF Implementation of ARC3D

Michael Frumkin, Jerry Yan*

Numerical Aerospace Simulation Systems Division
NASA Ames Research Center

Abstract

We present an HPF implementation of ARC3D code along with the profiling and performance data on SGI Origin 2000. Advantages and limitations of HPF as a parallel programming language for CFD applications are discussed. For achieving good performance results we used the data distributions optimized for implementation of implicit and explicit operators of the solver and boundary conditions. We compare the results with MPI and directive based implementations.

1. Introduction

In this study we continue an evaluation of High Performance Fortran (HPF) as a choice for machine independent parallelization of aerophysics applications initiated in [2]. For this study we chose ARC3D, see [12]. This application can be characterized as a numerically intensive calculation on a regular 3D grid with local access patterns to internal grid points and piecewise local calculation of the boundary conditions. ARC3D employs the same factorization of the Navier-Stokes equations as SP NAS benchmark, see [1], but makes few steps further to realistic fluid dynamic codes in several respects: it

- has a number of realistic boundary conditions
- uses a curvilinear coordinate system
- includes viscosity and turbulence
- applies an artificial viscosity for stabilization of the algorithm
- uses spatially variable time step

HPF provides us with a data parallel model of computations [4], sometimes referred also as SPMD model [9]. In this model calculations are performed concurrently with data distributed across processors. Each processor processes the segment of data which it owns (owner computes rule). The sections of distributed data can be processed in parallel if there are no dependencies between them.

The data parallel model of HPF appears to be a good paradigm for aerophysics appli-

*MRJ Technology Solutions, Inc. M/S T27A-2, NASA Ames Research Center, Moffett Field, CA 94035-1000; e-mail: frumkin@nas.nasa.gov, jyan@mail.arc.nasa.gov

cations working with data defined on large 3D grids. A decomposition of grids into sections of closely located points followed by a distribution of these sections across processors fits into the HPF model of computations. In order to be processed efficiently these sections should be well balanced in size, independent and regular. In our HPF implementation of ARC3D we addressed these issues and suggested data distributions satisfying these requirements.

HPF has a limitation in expressing pipelined computations which are essential for parallel processing of distributed data having dependencies between sections. This limitation obliges us to redistribute data in the direction orthogonal to the dependencies. It requires one to keep scratch arrays with an alternate distribution.

A practical evaluation of the HPF versions of ARC3D was done with the Portland Group `pghpf 2.4` compiler [9] on SGI Origin 2000. In the course of the implementation we had to address a few technical problems: an overhead introduced by the compiler, an unknown performance of operations with distributed arrays, and additional memory required for storing arrays with alternative distributions. To address these problems we used an empirical HPF performance model presented in [2]. In this respect our experience confirms two known problems with HPF compilers [7]: a lack of theoretical performance model and the simplicity of overlooking programming constructs leading to a poor code performance. A significant advantage of using HPF is that the conversion from F77 to HPF results in a well structured easily maintained portable program. An HPF code can be developed on one machine and ran on another (more then 50% of our development was done on NAS Pentium cluster Whitney).

In section 2 we consider a spectrum of choices HPF gives for code parallelization. In section 3 we characterize the algorithmic nature of ARC3D, then we describe an HPF implementation in Section 4. In Section 5 we consider HPF implementation of calculation of boundary conditions which do not scale down as the rest of the code and can be a significant road block for the code performance if not treated correctly. In section 6 we compare our performance results with MPI and compiler directives versions of ARC3D. Related work and conclusions are discussed in section 7.

2. HPF Programming Paradigm

In the data parallel model of HPF calculations are performed concurrently over data distributed across processors*. Each processor processes the segment of data which it owns. In many cases HPF compiler can detect concurrency of calculations with distributed data. HPF advises a two level strategy for data distribution. First, arrays should be coaligned with ALIGN directive. Then each group of coaligned arrays should be distributed onto abstract processors with the DISTRIBUTE directive.

HPF has several ways to express parallelism: f90 style of array expressions, FORALL and WHERE constructs, INDEPENDENT directive and HPF library intrinsics [6], [8]. In array expressions operations are performed concurrently with distributed segments of data. The compiler takes care of communicating data between processors and updating ghost values (copy faces) if necessary. FORALL statement performs computations for all values of the index (indices) of the statement without guaranteeing any particular ordering of the indices. It can be considered as a generalization of f90 array assignment statement and allows a compiler to overlap computations with communications.

INDEPENDENT directive states that there is no dependencies between different iterations of a loop and the iterations can be performed concurrently. In particular it asserts that Bernstein's conditions are satisfied: set of read and set of written memory locations on different loop iterations does not overlap and no memory location is written on different loop iterations, see [6], p. 193. All loop variables which do not satisfy the condition should be declared as new and are replicated by the compiler in order the loop to be parallelized.

Many HPF intrinsic library routines work with arrays and are executed in parallel (see [5] for specification of HPF intrinsic functions). For example, `random_number` subroutine initializes an array of random numbers, reduction, prefix and `MAXLOC` functions.

*The expression "data distributed across processors" commonly used in papers on HPF is not very precise since data resides in memory. This expression can be confusing for a shared memory machine. The use of this expression assumes that there is a mapping of memory to processors.

3. ARC3D Computational Algorithm

ARC3D solves a 3D discretization of Navier-Stokes equation

$$K(u^{t+1} - u^t) = Ru^t \qquad (1)$$

where u is a 5D vector defined in the points of a 3D regular grid, implicit operator K is a 7 diagonal block matrix of 5x5 blocks and explicit operator R has a matrix of similar structure, t is the iteration number.

ARC3D uses Beam and Warming approximate factorization of the implicit operator of equation (1):

$$K \cong T_x \cdot P_x \cdot T_x^{-1} \cdot T_y \cdot P_y \cdot T_y^{-1} \cdot T_z \cdot P_z \cdot T_z^{-1} \qquad (2)$$

where T_x, T_y and T_z are block diagonal matrices of 5x5 blocks, P_x, P_y and P_z are block pentadiagonal matrices of 5x5 diagonal blocks. The system (2) is solved by inverting block diagonal matrices T_x, $T_x^{-1} \cdot T_y$, $T_y^{-1} \cdot T_z$ and T_z^{-1} and solving the block pentadiagonal systems with matrices P_x, P_y and P_z. ARC3D code contains a number of enhancements relative to the SP benchmark.

ARC3D works in a *curvilinear coordinate system* which enables it use for a variety of physical geometries and grids. A uniformly spaced 3D regular grid points (ξ,η,ζ) are mapped with a nonlinear map $(x,y,z) = X(\xi,\eta,\zeta)$ into the physical space (x,y,z), see Figure 1. The Jacobian of the mapping, the components of the metric tensor and the chain rule expansions are used to transform the partial derivatives from physical space to the grid space. The Jacobian and the components of the metric tensor are computed in the subroutines `Jacob`, `XXM`, `YYM` and `ZZM`. The linear combinations of metric components are used to perform derivation in the grid space. The algorithm uses *a stationary grid* and some metric tensor components are precomputed.

ARC3D uses the *thin layer approximation* of the Navier-Stokes equation. This means that only the viscous terms in the direction normal to the body surface are retained and the solver in this direction is more expensive than in the directions parallel to the surface.

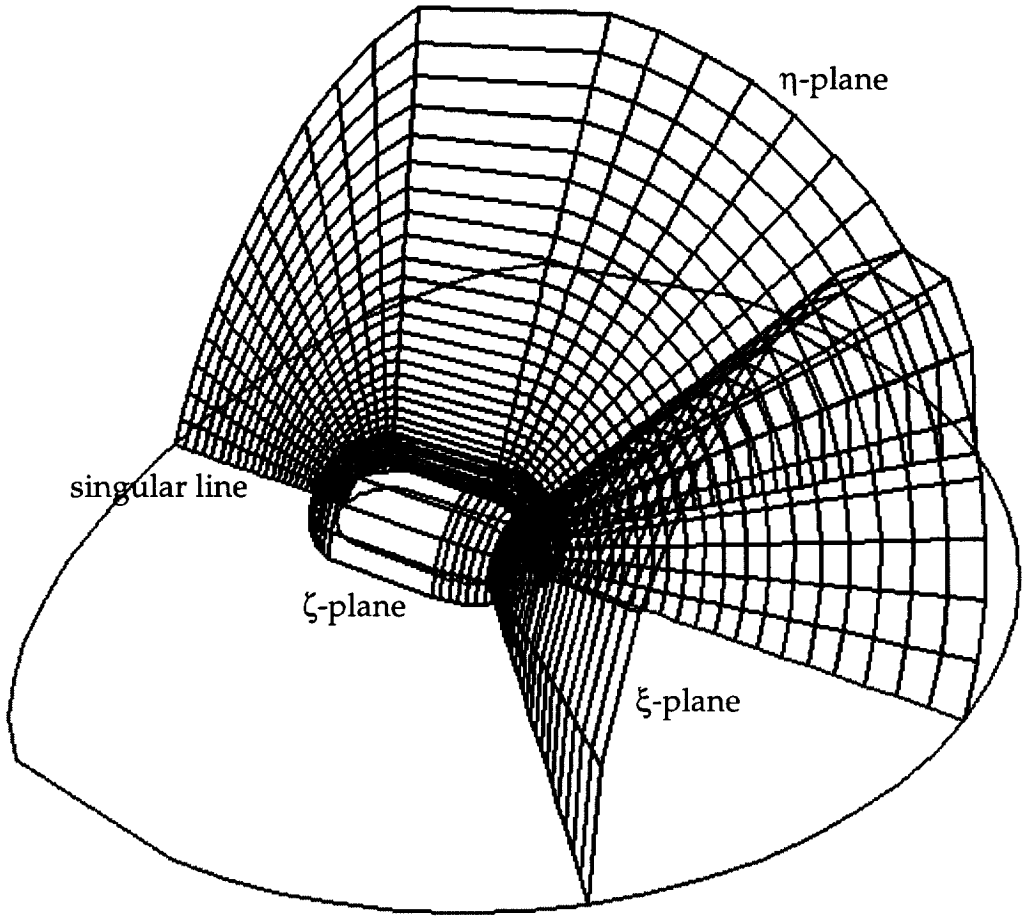

FIGURE 1. Hemisphere-cylinder-hemisphere curvilinear coordinate system.

Theoretically the factorization (2) leads to a numerically unstable algorithm especially in strongly nonlinear cases such as shocks ([12], Chapter VI, Section 12.2). Actually if we are interested in a steady state solution only ($u^{t+1} = u^t$) the implicit operator in (1) can be approximated in an arbitrary way as soon as the solution of the system converges and the explicit operator for calculation of RHS is preserved. It allows to use a variety of different approximations of K for improving the convergency speed and the algorithm stability.

For suppressing the numerical instability an *artificial dissipation* is used as a combination of the fourth and second order flux derivatives scaled with a spectral radius of the flux Jacobian and the pressure coefficient. The artificial dissipation is applied to the RHS (FILTER3D) as well as to the left hand side in the calculation of the coefficients of the im-

plicit operators of the solver in each direction. The form of the dissipation was chosen in such a way that it does not affect the pentadigonal structure of the matrices P_x, P_y and P_z.

ARC3D uses *spatially variable time step* for accelerating the convergency. The integration time step can be chosen in such a way the it satisfies CLF stability condition and depends on the factorization of the operator K. The condition depends on the spectral radius of the flux Jacobian and vary form point to point. ARC3D takes advantage of this and calculates the time step trough eigenvalues of the flux Jacobian at each grid point.

A number of *realistic boundary conditions* are used in ARC3D. The operators of the algorithm are at most second order (meaning that the stencil size is at most 5x5x5). These operators can be applied to any grid point which is at least on the distance 2 from the boundary. For grid points on the distance 0 or 1 from the boundary the flux values should be computed according to special formulas. This requires a careful partition of the points on the boundary and next to the boundary and applying appropriate operators at these points (BC subroutine). The partition and the boundary operators depend on the grid type (O, C or H). The operators reflect the physics phenomena at these points such as in/out flow, far flow, symmetry and tangential or no slip (for viscous flow) condition on the body surfaces. Some boundary operators a rather complex and involve solution of a PDE on the surface.

The main iteration loop of ARC3D, see Figure 2, starts with computation of the time step (VDTCALC) followed by the boundary conditions (BC) and the implicit operator RHS (VISRHS in viscose case) followed by a filtering subroutine FILTER3D. The solver is implemented as an interleaved inversion of block diagonal and block pentadiagonal matrices and is concluded with updating of the flux (ADD).

```
do step = 1,niter
   call VDTCALC
   call BC
   call RHS
   call VISRHS
   call FILTER3D
   call TKINV
   call X_SOLVE
   call NPINV
   call Y_SOLVE
```

```
      call NPINV
      call Z_SOLVE
      call TK
      call ADD
   end do
```

FIGURE 2. The main iteration loop of ARC3D

The computations in ARC3D are vectorized as sweeps through planes of the grid. Three types of the sweeps are used:

- horizontal plane in x-direction
- horizontal plane in y-direction
- plane orthogonal to y axis in z-direction

Each operator (implicit or explicit, excluding the boundary operators) processes one line in one of the planes at a time. This organization of computations is a good compromise between usage of the memory and vectorization of the operations: computations across a plane can be vectorized at the cost of few 2D arrays for additional storage. It also cache friendly since the planes can be easily be fragmented into stripes fitting into the cache.

As mentioned above the algorithm uses *a stationary grid* and the metric tensor components are precomputed. Precomputing of some other metric components such as metrics squares in x-, y- and z- directions would save the computational time. This however would increase the memory requirements and should be considered depending on the hardware available.

In the first approximation memory requirements of ARC3D can be calculated as the number of variables per grid point times the total number of grid points. ARC3D uses 22 real variables per grid point and we use an extra 13 variables for keeping arrays with an alternative distribution. The number of 2D arrays adds a second order term to the memory requirements.

The code is well structured. There is no equivalence statements or array redimensioning are used. The HPF implementation required a minimum code modifications.

4. Parallelization and HPF Implementation

ARC3D has a lot of parallelism to be exploited on a parallel computer. ARC3D computations can be considered as a sequence of operators each applied to a set of variables defined in a particular grid domain. For parallelization purposes the operators can be classified as implicit, having data dependencies, and explicit without any data dependencies. The explicit operators are easy to parallelize because the operations they perform are independent at each domain point. A parallelization of an implicit operator is constrained by the dependencies between the operations in different points of the domain.

There are 5 implicit operators in ARC3D: X_SOLVE, Y_SOLVE, Z_SOLVE in STEP3D and solving tridiagonal systems in x- and y- directions used for solving the pressure equation in BC. The dependencies of the operations in these operators are simple dependency in the direction of the operator.

In order to parallelize an explicit operator using HPF data parallel model it is sufficient to create a template for the domain of the operator, align arrays used in the operator with the template, distribute the template and declare the loops over the distributed dimensions as INDEPENDENT. This parallelization is efficient as soon as the template is evenly distributed and the update of the ghost points does not create too much overhead. This type of parallelism can be expressed with array assignments or FORALL statements. (We have used INDEPENDENT directive to minimize the code modification).

In order to parallelize an implicit operator the distribution of the operator's domain should be consistent with operator's dependencies. Any dependency between sections distributed on different processors would prevent parallelism. If a data distribution is "orthogonal" to the dependencies of an explicit operator then the loop which implements the operator can be declared as INDEPENDENT.

Bearing all these possibilities and restrictions in mind we chose the following data distributions: all operators except Y_SOLVE and one boundary operator (see Section 5) work with data distributed blockwise in y-direction. The Y_SOLVE works with data distributed blockwise in z-direction. The redistributions of the flux, `rhs`, `vardt` and `spect` are performed just before application of Y_SOLVE and back redistribution of `rhs` is performed immediately after Y_SOLVE. Note that Z_SOLVE includes a viscosity model and redistribution before Z_SOLVE would require one more 3D array to be redistributed. We

have used 1D distributions since 2D distributions would require more redistributions and would slower down the code, cf. [2]. Some additional redistributions are performed at in BC, see Section 5.

This data distribution allowed us to parallelize the outmost loops in implicit operators and most of explicit operators of RHS and FILTER3D. Explicit operators in the direction across data partition require some attention since an HPF compiler automatically updates ghost values of the arrays. This copy faces operation can cause a significant overhead. The redistribution for implementing these operations would be far too costly. As a compromise we chose an array syntax for implementing these operations because the compiler generates less overhead in these cases.

We used only a basic set of HPF directives as specified in [5]. The overall code modification was relatively small and included inserting HPF directives, writing interfaces for pentadiagonal and tridiagonal solvers called inside independent loops. Overall structure of the program was improved by removing extra copy operations for consolidating arguments of vectorized pentadiagonal solvers.

We profiled ARC3D for 3 problem sizes: 64^3, 128^3 and 194^3 (see Figure 3). The profile suggests few conclusions. The the implicit solver and explicit operators scale down nicely. The filter does not scale down as well. The redistribution time vary slightly with the number of processors and is the second major factor affecting scaling of the code. The communications involved in compute FILTER3D in z-direction also effect the scaling. The BC scales up and becomes the dominant term of the execution time on 32 processors.

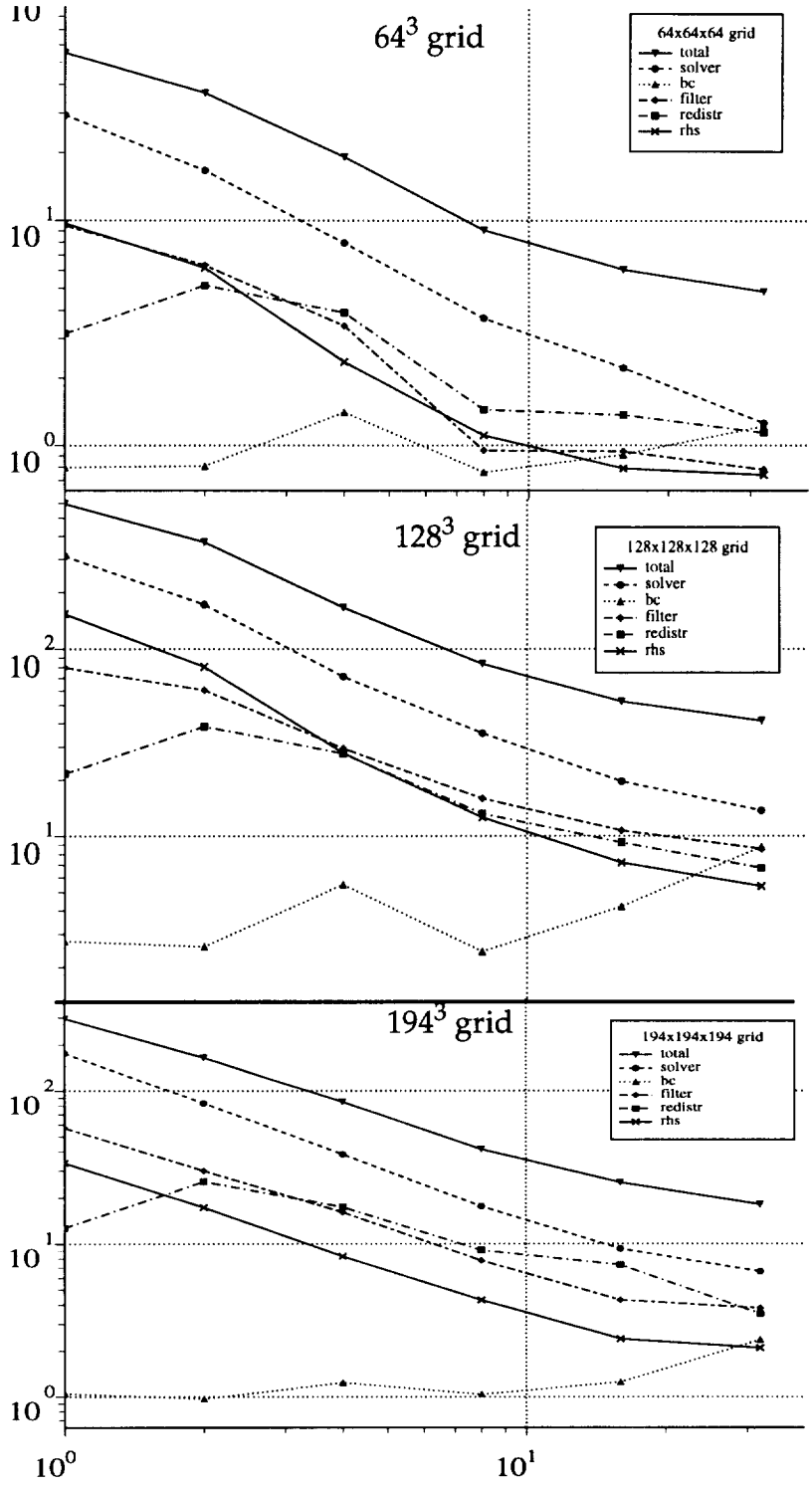

FIGURE 3. ARC3D profile for 64^3, 128^3 and 194^3 grids on SGI Origin 2000. The redistribution and copy faces communications in `rhsz` and `FILTER3D` affect the scaling.

5. Treating the Boundary Conditions in HPF

The computation of the boundary conditions on a single processor takes less then 0.5 percent of the total time. On multiple processors the inherited distributions of the arrays at boundary points can be in conflict with the directions of operators applied at the boundary points. The effect can be seen as increasing the time for boundary conditions calculations on the Figure 3. Even small deficiency in the parallelization of BC causes a significant performance penalty. We had a case when BC time on 32 processors consumed more then 70% of the total time.

There are several reasons why BC slows down as the number of processor increases. First, there are a number of boundary operators to be applied successively. Second, the amount of work to be performed by each processor is small increasing relative significance of the compiler introduced overhead. Third, the data distribution chosen for the solver, filter and the explicit operators sometimes are in a conflict with requirements for BC operators. This is especially the case for the implicit operators of tangential boundary condition.

There are three group of BC operators which should be specially considered. The calculation of the values at the singular lines, implicit operators used for solving the pressure equation and the mirror operator for computing symmetry BC. For the computation of the flux on the singular line we used HPF intrinsic function `sum`. For parallelization of the implicit operator for solving a second order recurrence relation we used a redistribution of the data on the plane $\zeta=0$ in the similar way how we redistributed data for parallelization of Y_SOLVE. The mirror operator of the symmetry boundary conditions was not parallelized since it just copies the values on the planes $\eta=0$ and $\eta=maxj$ distributed on the first and last processor respectively.

6. Computational Results and Comparison with other Implementations

ARC3D is well known code for solving a single zone CFD problems. It forms the foundation of the production CFD code `overflow`. HPF implementation of ARC3D is an important step in accessing HPF applicability to the production CFD applications. The timing results of HPF code for 3 grid sizes are summarized inTable 1. Into this table we

also included the timing results of two other parallel implementations of ARC3D: directive based version by J. Taft and MPI version generated by CAPTools.

TABLE 1. Timing of HPF version of ARC3D on 195 MHz SGI Origin 2000.

problem size \ nprocs	1	2	4	8	16	32
64^3 - HPF 10 iterations	55.3	36.7	19.1	9.1	6.1	4.8
64^3 - CAPTools-mpi 10 iterations	43.8	24.1	12.3	6.8	3.5	2.8
128^3 - HPF 10 iterations	594.8	371.6	166.3	83.4	52.3	41.2
194^3 - HPF 2 iterations[a]	295.4	165.0	84.8	41.4	25.2	18.1
194^3 - directives 2 iterations (J. Taft)	364.5	-	103.5	55.22	29.2	15.1

a. the results were obtained on a 250 MHz machine

A cache warming approach for data localization was implemented by Taft[13] in SGI compiler's directive version of ARC3D. This approach is to copy in data used in a loop or in a subroutine so the data reside in cache when they are needed. The cache warming was a key for achieving a good performance of the directive based version.

In order to calculate the boundary conditions efficiently we use the method proposed by J. Taft: creating a copy of array to be processed then process it and copy the results back. This approach allowed us to improve performance of the calculation of the tangential boundary condition.

7. Related Work and Conclusions

A development of a large application in a data parallel programming environment called *Fx* is reported in [11]. The authors showed that an air pollution model Airshed fits into the HPF programming paradigm and good performance of the code can be achieved on up to 64 processors of Cray T3D and Cray T3E.

The portability and scalability of HPF programs are studied in [7]. EP, FT and MG are used for comparison of a number of compilers, MPI and ZPL (a data parallel language developed at the University of Washington) implementations. One of the conclusions is that a consistent HPF performance model is important for scalability and portability of HPF programs.

An HPF implementation of reservoir simulation is reported in [3]. Two compiler

were compared and good scalability results were achieved on a number of platforms. Some other codes are used for testing of HPF compiler performance as well: CG 2D solver and Shallow Water code from NCAR: www.digital.com/info/hpc/fortranS.

An HPF implementation of NAS parallel benchmarks is reported in [2]. The implementation based on an empirical HPF performance model of CFD specific operations with distributed arrays. Advantages and disadvantages of using HPF are discussed and a comparison with MPI versions of NPB is given.

HPF gives user a high-level programming language constructs for expressing parallelism existed in a sequential code. It allows to port sequential code to a parallel environment with a moderate effort and results in a well structured parallel program. The machine architecture can be accounted for by using appropriate lower level message passing library as specified by `-Mmpi`, `-Msmp` or `-Mrmp` flags to `pghpf` compiler and requires a minimal effort from the user.

The hiding of distributed array handling results in uncertainty of the overhead of primitive operations with distributed arrays. Currently there is no HPF language constructs which can convey this overhead to the user. For example, data movement between processors can not be expressed in terms of HPF language. The problem is soften by `pghpf` compiler directives `-Minfo` and `-Mkeepftn` as well as by `pgprof` (HPF code profiler) ability to show message size and number. A clear performance model of handling distributed arrays would allow the user to steer the code to a better performance.

The HPF model of parallelism appears to be adequate for expressing parallelism existed in ARC3D with one exception. Due to inability of HPF organize pipelined computations an extra redistributions were required to make data distributions orthogonal to dependencies of implicit operators.

At the current level of HPF compiler maturity it generates code which runs about 2 times slower on a single processor than the original serial code. On multiple processors the code speeds up almost linearly until the point where the redistribution creates a significant overhead. We have plans to evaluate the HPF code performance on other architectures and with other HPF compilers.

Acknowledgments: The authors wish to acknowledge NAS scientists involved in the

effort of parallelization of ARC3D: Haquang Jin, Jim Taft and Thomas Pulliam. Insight to `pghpf` implementation of distributed array operations was provided by Douglas Miles and Mark Young from Portland Group. The work presented in the paper is supported under NASA High Performance Computing and Communication Program.

References

[1] D. Bailey, T. Harris, W. Sahpir, R. van der Wijngaart, A. Woo, M. Yarrow. *The NAS Parallel Benchmarks 2.0.* Report NAS-95-020, Dec. 1995. http://science.nas.nasa.gov/Software/NPB

[2] M.Frumkin, H. Jin, J. Yan. *HPF Implementation of NPB.* NAS Technical report 98-00-XXX. A short version of the report to be published in Proceedings of PDCS98 in Chicago, September 1-4, 1998.

[3] K. Gary Li, N. M. Zamel. *An Evaluation of HPF Compilers and the Implementation of a Parallel Linear equation Solver Using HPF and MPI.*Technical paper presented at Supercomputing 97, November 97, San Jose, CA.

[4] C.H. Koelbel, D.B. Loverman, R. Shreiber, GL. Steele Jr., M.E. Zosel. *The High Performance Fortran Handbook.* MIT Press, 1994.

[5] *High Performance Fortran Language Specification.* High Performance Fortran Forum, Version 2.0, CRPC-TR92225, January 1997, http://www.crpc.rice.edu/CRPC/softlib/TRs_online.html

[6] C.H. Koelbel. *An Introduction to HPF 2.0. High Performance Fortran - Practice and Experience.* Tutorial Notes of Supercomputing 97. November 97, San Jose, CA.

[7] T. Ngo, L. Snyder, B. Chamberlain. *Portable Performance of Data Parallel Languages.* Technical paper presented at Supercomputing 97, November 97, San Jose, CA.

[8] P. Mehrotra, J. Van Rosendale, H. Zima. *High Performance Fortran: History, status and future.* Parallel computing. May1998 v. 24 n. 3 / 4. pp. 325-354.

[9] The Portland Group. *pghpf Reference Manual.* February 1997.http://www.pgroup.com/ref_manual/hpfref.htm. 142 P.

[10] S. Saini, D. Bailey. *NAS Parallel Benchmark (Version 1.0) Results 11-96.* Report NAS-96-18, November 1996.

[11] J. Subholk, P. Steenkiste, J. Stichnoth, P. Lieu. *Airshed Pollution Modeling: A Case Study in Application Development in an HPF Environment.* IPPS/SPDP 98. Proceedings. Orlando, March 30- April 3, 1998, pp. 701-710.

[12] T.H. Pulliam. *Efficient Solution Methods for The Navier-Stokes Equations.* Lecture Notes for the von K'arm'an Institute For Fluid Dynamics Lecture Series: Numerical Techniques for Viscous Flow Computation In Turbomachinery Bladings, von K'arm'an Institute, Rhode-St-Genese, Belgium, 1985.

[13] J. Taft. *Initial SGI Origin 2000 Test Show Promise for CFD Codes.* NAS News, July-August, 1997, p. 1. http:/science.nas.nasa.gov/Pubs/NASnews/97/07/article01.html.