Preliminary Evaluation of High-Performance Fortran as a Language for Computational Fluid Dynamics

E.A. Bogucz[*] G.C. Fox[†] T. Haupt[‡] K.A. Hawick[§] S. Ranka[¶]
Northeast Parallel Architectures Center[||]
Syracuse University
Syracuse, NY 13244

Abstract

This paper evaluates the High-Performance Fortran (HPF) language as a candidate for implementing computational fluid dynamics (CFD) software on parallel-architecture computer systems. The paper reviews major HPF language features and discusses general algorithmic issues common to broad classes of CFD codes. Broader application areas, such as those covered by the NAS parallel benchmarks are also reviewed for suitability for HPF implementation. HPF is shown to provide convenient language structures for implementing several widely-used CFD algorithms, including finite-difference and finite-volume solvers that use regular grids. Other CFD algorithms — including multi-block, multi-grid and unstructured-mesh approaches — are most conveniently expressed using extensions to the initial HPF language specification.

1 Introduction

Aerospace industry interest in hardware and software for parallel computing has heightened recently, driven by a variety of complementary factors. Contributing factors include: successes in implementing research and applied codes on massively-parallel systems; maturing compilers and software development tools; the commitment of NASA and other federal labs to foster development of high-performance computing and communications (HPCC) technologies; and an expanding workforce of individuals who have relevant prior training and experience. Perhaps most important, the proliferation of engineering workstations typical at aerospace firms is widely recognized as a potentially significant, yet underused, distributed computing resource.

This paper evaluates the suitability of a new means for utilizing networked workstations and other parallel computing systems for CFD simulations of current interest to aerospace firms. The topic addressed here is the language High-Performance Fortran (HPF), which is defined in a specification adopted in 1993 [12]. The present paper reports results of a preliminary evaluation of strengths and limitations of HPF for implementing CFD codes that are candidates for use in multidisciplinary design applications. The present effort is part of a larger evaluation of NASA codes that is being conducted by the Multidisciplinary Analysis and Design Industrial Consortium (MADIC). MADIC was formed in March 1992 to accelerate the development of a new simulation and design environment, which takes full advantage of recent advances in HPCC technologies. The simulation and design environment envisioned by MADIC members includes multiple simulation modules executing simultaneously in a heterogeneous computing environment comprised of a mixture of machine architectures. Full development of the integrated system is expected to reduce design cycle time, increase accuracy of individual simulations, and account for multidisciplinary effects.

In addition to the effort reported in this paper, the MADIC NASA code evaluation project includes: definition of a process for industry certification of physical simulation codes [17]; evaluations of individual NASA codes; comparisons of results obtained with several codes; and evaluation of ADIFOR, which enables automatic differentiation of Fortran programs [6]. In the current phase of this project, five NASA CFD codes are being evaluated: CFL3D [26], TLNS3D [16], OVERFLOW [22], ENSAERO [27], and ADPAC [3]. The current phase of the project also includes comparisons of results obtained from several codes for wing and wing-

[*]Associate Director; Associate Professor, Department of Mechanical, Aerospace and Manufacturing Engineering, Syracuse University. Member, AIAA.

[†]Director; Professor, Computer Science and Physics, Syracuse University.

[‡]Fortran 90D Project Leader and Research Scientist.

[§]Research Scientist.

[¶]Fortran 90D Project Leader; Associate Professor, Computer and Information Science, Syracuse University.

[||]

```
      real    A(N,N), B(N,N)
      ..
c---- point-Jacobi iteration: A (old), B(new)
      do j=2,N-1
        do i=2,N-1
          B(i,j) = 0.25*( A(i,j-1) + A(i,j+1)
     $                  + A(i-1,j) + A(i+1,j) )
        end do
      end do
```

Figure 1: Fortran 77 code fragment for point-Jacobi iteration of the Laplace equation on a uniform rectangular grid.

body configurations [21] and aft-end/nozzle configurations [23].

This paper is organized as follows: Section 2 contains background information concerning parallel computing and the motivation for the HPF language. The main features of the HPF language specification are outlined in Section 3. Section 4 describes HPF implementations of some of the application kernels in the NAS benchmarks. Section 5 contains a general discussion of strengths and weaknesses of HPF for CFD and other applications. Issues relevant to the five NASA CFD codes currently being evaluated by MADIC are discussed in Section 6. Final remarks are presented in Section 7.

```
      real    A(N,N_OVER_P), B(N,0:N_OVER_P+1)
      ..
c---- point-Jacobi iteration: A (old), B(new)
c---- communicate with neighboring regions:
      if(MY_ID.NE.0)
     $   SEND(A(1:N,1),MY_ID-1)             !west
      if(MY_ID.NE.P-1)
     $   SEND(A(1:N,N_OVER_P),MY_ID+1)      !east
      if(MY_ID.NE.P-1)
     $   RECEIVE(A(1:N,N_OVER_P+1),MY_ID+1) !east
      if(MY_ID.NE.0)
     $   RECEIVE(A(1:N,0),MY_ID-1)          !west

c---- loop limits:
      LOW = 1
      if(MY_ID.EQ.0)   LOW=2
      HIGH = N_OVER_P
      if(MY_ID.EQ.P-1) HIGH = N_OVER_P-1

c---- compute:
      do j=LOW,HIGH
        do i=2,N-1
          B(i,j) = 0.25*( A(i,j-1) + A(i,j+1)
     $                  + A(i-1,j) + A(i+1,j) )
        end do
      end do
```

Figure 2: Message-passing code fragment for point-Jacobi iterative solution of the Laplace equation on a uniform rectangular grid.

2 Background

Experience in parallel computing to date indicates that the likely dominant approach for large-scale simulation codes will involve multiple-instruction, multiple-data (MIMD) distributed-memory systems. The distributed-memory MIMD class of parallel systems includes dedicated systems — such as the IBM SP series, the Thinking Machines CM-5, and the Intel Paragon — as well as networks of engineering workstations. To date, codes for distributed-memory MIMD systems have been written using subroutine calls to message-passing libraries, which require the programmer to "hand-code" communications between processors.

The usual message-passing approach is illustrated in Figures 1 and 2, which show kernels of a point-Jacobi iterative solution of the Laplace equation. Figure 1 shows a Fortran 77 code fragment for a calculation on an `N` $\times$ `N` uniform rectangular grid. Figure 2 shows an equivalent message-passing version, written for `P` processors in the usual "single-program, multiple data" (SPMD) style.[1] In SPMD programming, the same program is sent to all processors, but each processor executes selected statements based on its identification index `MY_ID` (an integer that varies from `0` to `P-1`). Note that the message-passing version of the code includes calls to subroutines `SEND` and `RECEIVE`, which transfer data between processors that are computing solutions for adjacent regions of the domain. (In this example, the simplest possible domain decomposition is used: the region is divided into `P` vertical blocks, each of which has `N` rows and `N_OVER_P = N/P` columns.)

Using the relatively-limited software-development tools available to date, message-passing programming generally can be tedious and error-prone. Portability of message-passing codes can also be a problem when libraries unique to a particular hardware vendor are used. The recent development of machine-independent standards such as MPI should relieve many portability problems, and software-development tools also are improving significantly. Nevertheless, programming using message-passing is likely to remain more labor-intensive than programming serial-architecture machines using traditional high-level languages, such as Fortran 77 and C.

[1]Alternate codings of examples presented here are possible, and may be desirable for enhanced computational performance. The intent here is to show equivalent code fragments that illustrate main features of each language.

```
      real    A(N,N), B(N,N)
      ..
c---- point-Jacobi iteration: A (old), B(new)
      B(2:N-1,2:N-1) = 0.25*(
     $           A(2:N-1,1:N-2) + A(2:N-1,3:N)
     $         + A(1:N-2,2:N-1) + A(3:N,2:N-1) )
```

Figure 3: Fortran 90 code fragment for point-Jacobi iteration of the Laplace equation on a uniform rectangular grid.

An alternative to hand-coded message-passing is to develop compilers capable of generating automatically the message-passing instructions required by MIMD systems. Unfortunately, attempts to develop parallelizing compilers for widely-used languages like C and Fortran have not been successful. Although fully automatic parallelizing compilers can produce excellent results on isolated loop nests, they often fail to exploit available parallelism in realistic application codes due to false dependencies. As a result, efficient use of MIMD systems requires either the use of message-passing programming or the development of new languages.

One new language of interest in parallel computing is Fortran 90 [18], which serves as the basis for variants including CM-Fortran (developed by Thinking Machines Corporation) and MPfortran (developed by Maspar). Fortran 90 is an extension of Fortran 77 that allows new features to be integrated into existing code in a controlled, evolutionary manner. Fortran 90 includes major new features in a broad range of areas, including new control structures, array processing, dynamic memory management, data types and structures, operators, subprograms, and global data. In effect, Fortran 90 captures decades of experience in programming vector supercomputers as well as advances in computer programming theory and practice. In the context of the present paper, the most significant extension provided by Fortran 90 is the treatment of arrays as individual data objects. This feature is inherently well-suited to data-parallel operations, which are readily processed by the single-instruction, multiple-data (SIMD) class of massively-parallel computers (such as the Connection Machine CM-2 and the Maspar MP-1).

Figure 3 shows a Fortran 90 implementation of the point-Jacobi code shown previously. This code makes use of "array section" notation, which permits a compact representation of the point-Jacobi computational template. For scientific and engineering programming, Fortran 90 clearly is a significant extension of Fortran 77. However, experience in using Fortran 90 on parallel computing systems indicates that it lacks constructs and directives required for effective use of MIMD systems. Based on this realization, a specification for a

```
      real    A(N,N), B(N,N)

c---- DISTRIBUTION SPECIFICATION
!HPF$ PROCESSORS PROC(P)
!HPF$ TEMPLATE   TEMP(N,N)
!HPF$ DISTRIBUTE TEMP(*,BLOCK)
!HPF$ ALIGN      A(i,j) with TEMP(i,j)
!HPF$ ALIGN      B(i,j) with TEMP(i,j)
      ..
c---- point-Jacobi iteration: A (old), B(new)
      FORALL (i=2:N-1,j=2:N-1)
     $    B(i,j) = 0.25*( A(i,j-1) + A(i,j+1)
     $                  + A(i-1,j) + A(i+1,j) )
```

Figure 4: HPF code fragment for point-Jacobi iterative solution of the Laplace equation on a uniform rectangular grid.

language called High-Performance Fortran (HPF) was adopted in 1993 [12]. The HPF language specification is based in part on experience gained with the research languages Fortran D [13, 14] and Vienna Fortran [7]. Four companies are expected to release HPF compilers during 1994.

A critical issue for parallel computing is the problem of scalability and portability of the software, since this is the key for protection of software investment. This issue is partially addressed by HPF. HPF provides a minimal set of extensions to Fortran90 to support the data parallel programming model, which is defined as single-threaded, loosely-synchronous parallel computation using a global name space. The purpose of HPF is to provide compilers and software development tools that produce top-performance codes for both MIMD and single-instruction, multiple data (SIMD) computers that have non-uniform memory access cost. Portability of HPF codes means that the efficiency of the code is preserved for different machines with comparable number of processors.

One fundamental extension to Fortran 90 provided by HPF concerns compiler directives that specify the distribution of data among multiple processors. Figure 4 shows an HPF code fragment equivalent to the examples shown in Figures 1 - 3. The HPF code includes the compiler directives `PROCESSORS`, `TEMPLATE`, `DISTRIBUTE`, and `ALIGN`, which serve to allocate a separate block of columns of arrays `A` and `B` to each processor. The code also includes the HPF statement `FORALL`, which provides the template for computation of each element.

The current HPF language specification is thought to support data parallel programming for half to three-quarters of practical applications, but it has limited support for unstructured problems that require irregular communications. Support for irregular communications is a matter of current research interest. A new round of discussions concerning a second HPF specification ("HPF-II") will begin in January 1994. Ultimately, HPF is expected to complement — not replace — programming using message-passing libraries. For many applications, HPF is expected to provide convenient, compact and relatively-simple programs without recourse to explicit message-passing instructions. However, other applications are expected to require hand-coded message passing to yield acceptable utilization of distributed-memory MIMD systems. This paper is intended to contribute to the understanding of which applications are best suited to HPF, and which are not.

3 HPF language features

HPF consists of extensions to Fortran 90. One advantage of building a new language on Fortran is that it is expected to facilitate reuse of the enormous inventory of existing codes written in Fortran 77 (or earlier versions). In addition, this approach leverages existing skills of experienced Fortran programmers, as well as the significant investment of computer manufacturers, which have developed highly-optimized Fortran compilers. This section includes a brief overview of Fortran 90 and the major extensions to Fortran 90 contained in the HPF specification. Complete details concerning HPF may be found in references [12, 15].

3.1 Fortran 90 features

Although Fortran 90 includes major extensions to Fortran 77 in a broad range of areas, the principal features of interest in the present paper involve array processing. In this area, notable extensions include:

- Processing of arrays as individual data objects. For example, the Fortran 90 statement `A = B + C` adds corresponding elements of arrays `B` and `C` without requiring `DO`-loop indexing over each element of the arrays.
- New syntax for "array sections," which contain a subset of elements from an array.
- Masked array assignments, which permit selective processing of array elements. For example, the Fortran 90 statement `WHERE(A=0) B=C` assigns elements of the array `B` to be equal to corresponding elements of the array `C` only for those locations in which the corresponding elements of the array `A` are zero.
- Use of arrays and array sections as arguments to a broad case of elemental intrinsic functions (e.g., `B=ABS(A)` sets each element of array `B` to be equal to the absolute value of the corresponding element of array `A`.
- New intrinsic functions that perform transformations or reductions on arrays or array sections. Examples include `SUM`, which returns a sum of array elements and `MAXVAL`, which returns the value of the largest array element.

The processing and manipulation of arrays as individual data objects allows the programmer to represent bulk data parallelism in a form that can be exploited relatively easily.

3.2 HPF Extensions

The HPF extensions provide the compiler with information about locality as well as concurrency present in the application code. This information is of utmost importance in achieving high performance on distributed memory machines. These extensions fall into four categories: compiler directives, new parallel constructs, library routines, and an escape mechanism for interfacing with other languages and libraries.[2]

3.2.1 Compiler Directives

Compiler directives are structured comments that suggest implementation strategies or assert facts about a program to the compiler. They may affect the efficiency of the computation performed, but they do not change the semantics of the program. Analogous to Fortran 90 statements, there are declarative directives, to be placed in the declaration part of a scoping unit, and executable directives, to be placed among the executable Fortran 90 statements. Efficient execution of data-parallel applications on distributed memory machines requires distribution of data such that data locality and/or load balance is achieved. HPF data alignment and distribution directives allow the programmer to advise the compiler how to assign data objects (typically array elements) to the memories of processors.

HPF uses a two-level mapping of data objects to "abstract processors." as depicted in Figure 5. Objects (typically arrays) are first aligned relative to one another, and then groups of aligned objects are distributed onto a programmer-defined rectilinear arrangement of abstract processors (using the `PROCESSORS` directive).

[2]HPF also restricts somewhat alternatives for sequence and storage association available in Fortran 90 due to incompatibility with the data distribution directives (see [12] for further details).

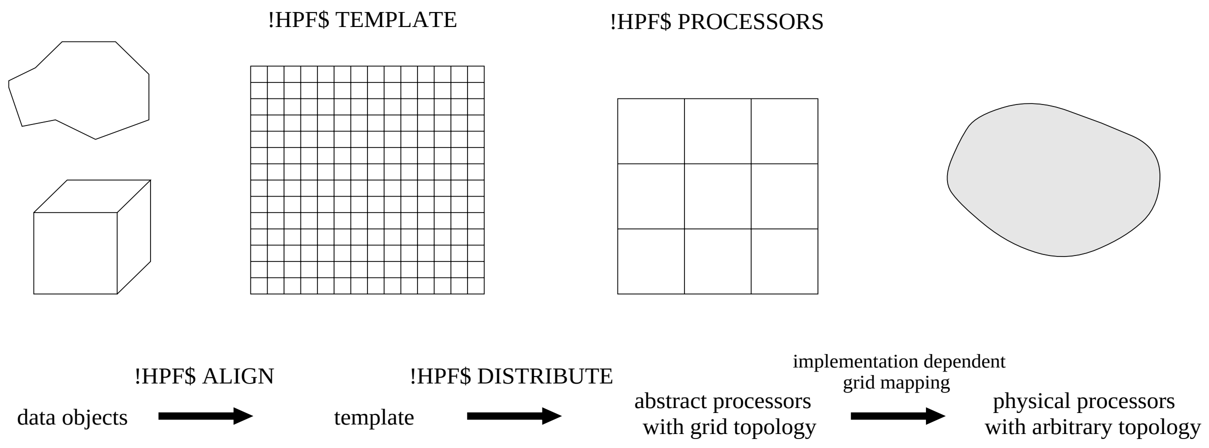
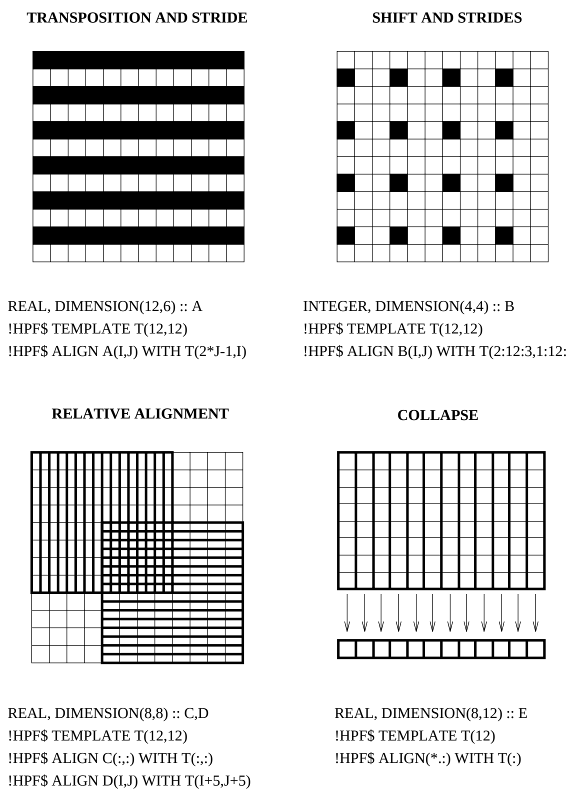

!HPF$ TEMPLATE
!HPF$ PROCESSORS
!HPF$ ALIGN
!HPF$ DISTRIBUTE
implementation dependent grid mapping
data objects
template
abstract processors with grid topology
physical processors with arbitrary topology

Figure 5: Mapping of data objects to physical processors using HPF `ALIGN` and `DISTRIBUTE` directives.



```
REAL, DIMENSION(12,6) :: A
!HPF$ TEMPLATE T(12,12)
!HPF$ ALIGN A(I,J) WITH T(2*J-1,I)
```

```
INTEGER, DIMENSION(4,4) :: B
!HPF$ TEMPLATE T(12,12)
!HPF$ ALIGN B(I,J) WITH T(2:12:3,1:12:
```

```
REAL, DIMENSION(8,8) :: C,D
!HPF$ TEMPLATE T(12,12)
!HPF$ ALIGN C(:,:) WITH T(:,:)
!HPF$ ALIGN D(I,J) WITH T(I+5,J+5)
```

```
REAL, DIMENSION(8,12) :: E
!HPF$ TEMPLATE T(12)
!HPF$ ALIGN(*.:) WITH T(:)
```

Figure 6: Examples of HPF alignments.

The final mapping of abstract to physical processors is not specified by HPF and is implementation dependent.

The `ALIGNMENT` directive can be used to enforce the collocality of different arrays. All array elements aligned with an element of the template are mapped to the same processor. The template can either be provided explicitly by using the `TEMPLATE` directive or implicitly by using another array as a template. Examples of HPF alignments are shown in Figure 6.

The template of an array is distributed among the set of abstract processors by using `DISTRIBUTE` directive. Several patterns of partitioning are available. The important ones are `BLOCK` and `CYCLIC`. The former slices the template into uniform contiguous blocks of elements, while the latter assigns the template in a round-robin fashion. In addition, a particular dimension of the template may be collapsed or replicated onto the abstract processor grid. The HPF distributions are illustrated in Figure 7.

The combination of alignment information of an array and distribution information of its template defines the mapping of an array. Remapping of arrays during runtime can be flagged to the compiler using the `DYNAMIC` directive. The actual remapping is then triggered by executable directives `REALIGN` and `REDISTRIBUTE`.

3.2.2 Parallel Constructs

Parallel constructs are provided in HPF to enable the programmer to make explicit assertions about concurrency in a code. Available features include the `FORALL` statement, the `FORALL` construct and the `INDEPENDENT` directive.

The `FORALL` statement can be used to express assignments to sections of arrays. It is similar to the array assignment of Fortran 90, but allows more general sections and computations to be specified. The semantics of `FORALL` guarantee that array elements may be assigned in an arbitrary order. To ensure determinism it is required that each array element on the left hand side (LHS) is assigned only once. The execution of the `FORALL` statement may require intrastatement synchronization; the evaluation of the left side expression of the `FORALL` assignment must be completed for all array elements before the actual assignment is made. A

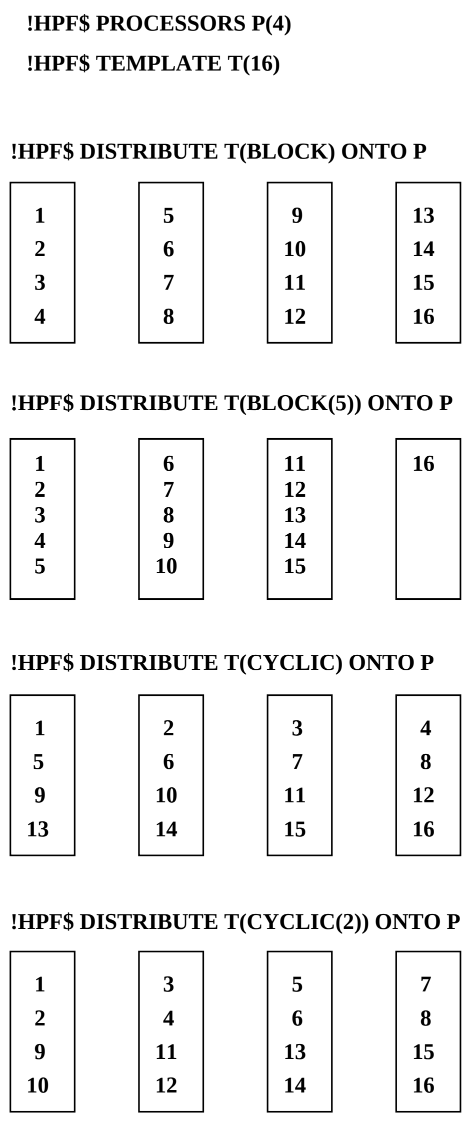


Figure 7: Examples of HPF distributions.

sequence of `FORALL` statements can be combined using the `FORALL` construct with similar semantics.

The `INDEPENDENT` directive asserts that the statements in a particular section of code do not exhibit any sequentializing dependencies. Although it does not change the semantics of the code, it can be used to provide information to the language processor in order to allow optimizations. This directive is useful in cases when traditional dependency-analysis methods may be extremely difficult, especially because of indirections.

3.2.3 Intrinsics

Experience with massively-parallel machines has identified several basic operations that are very valuable in parallel algorithm design. HPF has added several classes of parallel operations to the operations already provided in Fortran 90. Important HPF computational intrinsics include:

1. *Simple reduction functions*: These functions can be used to calculate summations and other related operations along one or more dimensions of an array.
2. *Combining scatter functions*: These allow for combining nonoverlapping subsets of array elements.
3. *Prefix and suffix functions*: These functions provide scan operations on arrays and subarrays.
4. *Array sorting functions*: These functions can be used for sorting in increasing or decreasing order.
5. *Array manipulation functions*: Many useful operations such as transposing an array or rotating an array are available.
6. *Array location functions*: These are useful in finding location of elements in arrays with maximum or minimum value.
7. *Array construction functions*: These allow construction of new arrays, potentially of new size and shape, by using the elements of another array.
8. *Vector and matrix multiplication functions*: These include dotproduct and matrix multiplication.
9. *Elemental intrinsics functions*: These functions act on individual elements of an array.

In addition, several system inquiry functions useful for controlling parallel execution are provided.

3.2.4 Extrinsics

An escape mechanism is provided in the form of an interface which can be used by other languages and possibly supporting other programming paradigms, including explicit message-passing. The HPF library also provides a set of inquiry functions that return all necessary information to fully exploit the actual data distribution generated by the HPF program.

3.3 HPF performance

Full implementations of HPF compilers have yet to be released, although four companies are expected to market commercial versions of at least partial implementations[3] in 1994. Performance experience is

[3]The HPF language specification [12] also defines a preliminary "Subset HPF" language to encourage early introduction of commercial compilers.

Program	Number of Processors 1	2	4	8	16
Gauss Hand	623.16	446.60	235.37	134.89	79.48
Gauss F90D	618.79	451.93	261.87	147.25	87.44
Nbody Hand	6.82	1.74	1.29	0.76	0.42
Nbody F90D	13.82	5.95	2.40	1.31	0.86
Option Hand	4.20	3.14	1.60	0.83	0.43
Option F90D	4.30	3.19	1.64	0.84	0.44
Pi Hand	0.398	0.200	0.100	0.053	0.030
Pi F90D	0.411	0.207	0.104	0.054	0.032

Times in milliseconds, obtained on Intel iPSC/860.
Problem sizes: Gauss, 1023 × 1024; Nbody, 1024 × 1024; Option: 8192; Pi, 65536.

Table 1: Comparison of execution times of hand-coded message-passing programs and of code generated by the Fortran 90D prototype HPF compiler for selected applications [4].

Loop: Application	F90D/HPF	Hand	Ratio
1: Hydrodynamics	2.545	2.550	0.9980
2: Incomplete Cholesky	11.783	10.440	1.1286
3: Inner product	3.253	3.249	1.0012
4: Banded linear equations	5.139	3.212	1.600
5: Tridiagonal elimination	30928.6	30897.7	1.001
6: Linear recurrence	1849.1	1886.5	0.9801
7: Equation of state	11.346	3.704	3.0632
8: A.D.I.	38.656	20.038	1.9291
9: Numerical integration	2.255	2.441	0.9238
10: Num. differentiation	9.814	4.589	2.1386

Times in milliseconds, obtained on 16-node Intel iPSC/860.
Problem size is 16K (real numbers) for each application.

Table 2: Comparison of execution times of hand-coded message-passing programs and of code generated by the Fortran 90D prototype HPF compiler for the first 10 Livermore Loops [4].

available with prototype versions of HPF compilers, which were developed during research projects concerning possible language features and compilation strategies. One such research compiler is the Fortran90D system [13, 14], which was used to obtain the results shown in Tables 1 and 2. Results for all cases shown here used the `BLOCK` data distribution. The results show that the performance of compiler-generated code is comparable to hand-coded message-passing programs; execution times for code produced by the Fortran90D compiler are within a factor of 2 of times for hand-coded message-passing programs in most cases.

4 HPF for NAS Benchmarks

The NAS benchmarks [2] are a well-known set of compact application benchmarks which have been implemented on a number of high performance-computing platforms. As part of the present project full HPF implementations of these kernels are being developed to provide a means of assessing the HPF language itself, both in terms of ease of use, ease of migration of source from existing implementations as well as compiler efficiency and hence actual performance.

Full source code of the NAS benchmarks is somewhat lengthy, even in the compact notation HPF allows, so only kernel fragments of these benchmarks are presented here.

The NAS EP benchmark is typified by the need to histogram or tabulate random deviates with a particular correlation property. The code fragment in Figure 8 illustrates how this might be achieved compactly in HPF. A distribution template is set up to spread samples from some random process across processors. Temporary arrays are set up to allow processors to make a partial summations and counts of those samples which fail some test in their own local memory. HPF allows the programmer to ensure the compiler has enough information in the source code to implement the problem in this way if it chooses.

The `WHERE` construct is used to express the potential independent processor operation to partially sum the samples. The `SUM` intrinsic function can then be used to sum the partial summations.

A more sophisticated example of histogramming data using High Performance Fortran in the context of an industrial CFD oriented code is given in [10]. The problems that arise concern the ordering of loops across the data and therefore how data is distributed across processor memories.

Another illustration of HPF notation is given in Figure 9 which is a fragment of the NAS-IS benchmark. In this example, a vector of initial key values is sorted by building up key density information in an array stored across the processors distributed memory. The `WHERE` and `FORALL` constructs are both used, as is the `INDEPENDENT` directive. This latter tells the compiler that the loop over `i` can safely be distributed across processors, and that the iterations do not have a preferred order. This is extra information to the compiler, which would not be able to deduce this fact from the source code otherwise.

Although HPF provides language features which are well-suited to expressing extra information to the compiler to allow it to make good parallel optimisations, the programmer may not have any "hints" to give the compiler in some applications. A good example is typified by the NAS-CG benchmark. This code solves a conjugate gradient problem expressed in the form of a large sparse matrix. The kernel required for efficient performance is the multiplication of this sparse matrix by a vector. Typically, the sparse matrix is too large to store in physical memory, even when stored across

```
      real,    dimension(1:n) samples
      real,    dimension(1:n) partial_sum
      integer, dimension(1:n) failed_count

!HPF$ PROCESSORS PROC(p)
!HPF$ TEMPLATE MY_TMP(1:n)
!HPF$ DISTRIBUTE MY_TMP(BLOCK) ONTO PROC
!HPF$ ALIGN (:) WITH MY_TMP(:) :: samples
!HPF$ ALIGN (:) WITH MY_TMP(:) :: partial_sum
!HPF$ ALIGN (:) WITH MY_TMP(:) :: failed_count

      failed_count = 0
      partial_sum = 0.0

      do i=1,nsamples

        call get_samples( samples )
        where( samples < 1.0 )
          partial_sum = partial_sum + samples
        else where
          failed_count = failed_count + 1
        end where

      end do

      write(6,*) 'Number failed = ',
     &            sum( failed_count )
      write(6,*) 'Total = ', sum( partial_sum )
```

Figure 8: HPF code fragment for the histogramming problem similar to that in NAS EP Benchmark.

distributed memory. The matrix is therefore stored in packed form so that only non-zero elements are stored, and a set of indexing vectors must be stored to allow reconstruction of parts of the sparse full matrix when required.

Using a conventional approach, significant compute time is wasted indexing into the correct locations for the non-zero elements, but there is no discernable extra information the programmer can give the compiler to achieve an efficient implementation. Alternative algorithms store the matrix as a fully dense structure, but only load part of it into memory at one time. Such algorithms rely on a fast and efficient parallel input/output (I/O) subsystem. Unfortunately there is no portable way to express the desired information to the compiler since HPF does not yet address the problems of parallel I/O.

5 Evaluation of HPF for CFD

HPF is targeted largely towards the solution of regular problems. These problems use multidimensional arrays as the basic data structures which can be ma-

```
      integer, dimension(0:N-1)      :: key, rank
      integer, dimension(0:MAXKEY-1) :: keyden
      integer, dimension(0:MAXKEY-1) :: keydenc,keydenc1
      integer                        :: i, j, k
      integer                        :: ibeg,iend

      INTEGER  nkeyden(0:MAXKEY-1,0:N-1)

!HPF$ PROCESSORS PROC(P)
!HPF$ TEMPLATE TMP(0:MAXKEY-1)
!HPF$ DISTRIBUTE TMP(CYCLIC) ONTO PROC
!HPF$ ALIGN (:) WITH TMP(:) :: KEYDEN,KEYDENC
!HPF$ ALIGN NKEYDEN(:,*) WITH TMP(:)

          nkeyden=0

          do j=0,MAXKEY-1
            where(key.eq.j)
             nkeyden(j,:)=nkeyden(j,:)+1
            end where
          enddo

          keydenc=0
          keyden=sum(nkeyden,dim=2)
          FORALL(i=1:MAXKEY-1)
            keydenc(i)=sum(keyden(0:i-1))
          END FORALL

!HPF$ INDEPENDENT(k,j)
          do i=0,MAXKEY-1
            k=keydenc(i)
            do j=0,N-1
            if(nkeyden(i,j).ne.0) then
              rank(j)=k
              k=k+1
            endif
           enddo
          enddo
```

Figure 9: HPF code fragment for the integer rank sorting as in NAS-IS benchmark.

nipulated by the explicit parallel constructs as well as by the intrinsics described in section 3. Data-locality and/or load balancing information can be provided by distribution directives such as `BLOCK` and `CYCLIC`. The following CFD applications use these features and can thus be successfully tackled using HPF:

1. *Structured Grid Solvers (Single Block)*: These solvers form the basis for the classical approaches for solution to computational fluid dynamics problems and are readily implemented in HPF as it is currently specifed 4.
2. *Panel Method:* The Panel Method has been found to be of great use in a variety of industrial designs of whole-aircraft simulations that neglect viscous ef-

fects. This application requires the use of full matrix manipulation algorithms which are available as intrinsic functions in HPF. Furthermore, these applications require looping over several relatively independent instances, which can be specified by using the **INDEPENDENT** directive.

3. *Vortex Methods:* Vortex methods require the calculation of pairwise interactions between all the elements involved in the computation. This is similar to the classical $O(N^2)$ N-body algorithm. This can be implemented by using a simple **DO** loop which shifts the array representing the bodies by one step in each of its N iterations. Within each **DO** loop a **FORALL** is executed to compute the N pairwise force calculations. Compilers can exploit the possibility of combining several shifts together to improve the performance. When the pairwise interactions are commutative, the total amount of computation can be reduced in half by calculating the interaction only once. This requires a considerably more difficult algorithm which may violate the owner computes rule.

The use of unstructured grids is now growing in popularity for CFD applications. These can be broadly divided into two classes based on mesh generation strategy used: fully unstructured grids; irregularly coupled structured grids. Further difficult unsteady CFD problems require the use of structured/unstructured adaptive grids. Efficient implementation of these applications on parallel machines requires data remapping so that data-locality as well as load balancing is maintained. The current HPF is not adequate for easy specification of these applications. The following extensions are required in HPF for solving such applications:

1. *Partitioner Specification:* The parallelization of unstructured grids requires partitioning the different array elements so that the load is balanced and communication is minimized. There are a variety of partitioners available in the literature. Information that can be exploited by these partitioners includes:

   - proximity in physical space (e.g. coordinates of grid nodes corresponding to an array element),
   - topological connectivity structure (interaction between different array elements),
   - a computational load associated with each array element.

   For adaptive grids, the computational structure changes from one phase to another in an incremental fashion. Efficient parallelization of these applications involves reacting quickly to minor modifications in the data structure and hence requires that the partitioning of the grid be updated as it changes over time. This also requires utilizing the partitioning information of the previous phase.

2. *Irregular Distribution of Arrays:* Arrays may need to be irregularly distributed among the processors. This embedding can be based on use of a partitioner or a repartitioner.

3. *Distribution of Arrays onto disjoint or overlapping processor groups:* In many cases the computational grid is decomposed into a set of several regular and/or irregular subgrids. Effective parallelization of these applications requires mapping of each subgrid into a subset of processors.

4. *Extensions to the* **FORALL** *loop:* Current **FORALL** loops require that there is a unique left hand side for every iteration of the loop. Solutions on unstructured grids require scatter/gather operations in which the new value of the array element is the sum of several iterations. Although this can potentially be achieved in HPF by using **REDUCE** intrinsics, it makes the code considerably more difficult. The **FORALL** loop needs to be extended to allow for multiple right hand sides to a single left hand side. In case of multiple hits, suitable reduction functions can be specified.

It should be noted that many of these extensions have been implemented in prototype compilers and are currently being discussed for potential inclusion in HPF II.

Multigrid methods, especially those involving structured grids, are now becoming widely used in CFD. Although the structured multigrid can be represented using the **FORALL** statement in HPF, additional features to optimize data layout are required. The use of unstructured multigrid methods requires **ALIGN** directives between different levels in addition to the features required by unstructured meshes.

A summary of different features required for the applications given above is summarised in Table 3. Table 4 provides a preliminary summary of applicability of HPF for *other* applications based on some experiences of implementing these applications on high performance computing systems. A full description of all applications areas listed will be provided elswhere.

It is important to observe that real industrial applications codes involve more than one basic algorithm to be applied to the problem data and that there are often conflicting data layout requirements between sub algorithms [11]. For this reason the data remapping features of HPF are particularly important.

A good example of the remapping that may be required is found in an operational CFD engineering codes and numerical weather simulation codes [10]. Some field based calculation on a regular or regular grid may be required to interact with real measured data either dynamically in the case of operational forecasting or statically in the case of complex initial conditions or boundary conditions in the case of fluid flow around a complex engineering structure such as a reactor [11]. There may be strong operational resons why the code that deals with each sub algorithmic component should be maintained as separate and not merged. The dynamic remapping of data at subroutine interfaces is therefore an important facility within HPF.

6 NASA CFD codes

In the initial phase of the MADIC code evaluation project, representatives from MADIC companies and NASA selected five codes for industry evaluation: ADPAC, CFL3D, ENSAERO, OVERFLOW and TLNS3D. Features of the codes relevant to the present paper are summarized briefly below. The primary considerations here concern data structure and communications, which are influenced strongly by the grids supported by each code (structured or unstructured; single or multiblock) and the numerical solution options (in particular, whether multigrid solutions are supported).

ADPAC is a suite of Euler/Navier-Stokes codes developed at NASA Lewis for turbomachinery simulations [19]. ADPAC includes APES, which solves average-passage equations, and AOACR, for angle-of-attack/coupled row situations. Both codes permit multiblock structured grids. Parallel versions of the codes have been implemented using APPL, a message-passing library developed at NASA Lewis for homogeneous networks of workstations.

CFL3D is a thin-layer, Reynolds-averaged Navier-Stokes solver developed at NASA Langley [24]. The code includes capabilities for multiblock structured grids, which may be matched, patched or overset. Time-accurate unsteady simulations are supported, including the use of moving meshes. Multigrid solution techniques also are supported.

ENSAERO is an integrated code for fluids, structures, controls and optimization analysis of aerospace vehicles, developed at NASA Ames [9]. The code uses finite-difference methods on patched, structured zonal grids to solve Euler/Navier-Stokes (thin-layer Reynolds-averaged equation) for the flow field and the finite-element method for structural dynamics simulations. A message-passing version of the code has been developed for the Intel iPSC/860, which, executing with 64 processors, has been shown to obtain performance equivalent to a single-processor Cray C-90 (approximately 400 MFLOPS).

OVERFLOW is a thin-layer Navier-Stokes code developed by the Computational Technology Branch at NASA Ames [5]. The code currently uses multiblock, structured, overset grids; a version for unstructured grids is planned. OVERFLOW permits the user to specify different numerical methods for each grid block.

TLNS3D is a thin-layer, Reynolds-averaged Navier-Stokes solver developed at NASA Langley [25]. The code includes capabilities for multiblock structured grids and multigrid solution techniques. A single-block version of the TLNS3D has been adapted for design purposes using ADIFOR [6] to generate sensitivity derivatives. HPF versions of this code are under development at ICASE.

A preliminary evaluation of the five codes yields the following general observations concerning possible HPF implementations:

1. All five codes use structured grids for CFD calculations. At least within a single grid block, structured-grid computations are likely to be expressed well in HPF, as described in Section 5.
2. All five codes support multiblock structured grids, which may be difficult to represent compactly in the current version of the HPF specification.
3. Two codes (CFL3D and TLNS3D) offer multigrid solution strategies, which require language features for irregular communications that is beyond the scope of the current HPF specification.
4. None of the five codes considered in this phase of the NASA code evaluation project currently support unstructured grids, but support for unstructured grids may soon become available for several codes. As described in Section 4, efficient parallel solutions on unstructured grids require support for irregular communications that is beyond the scope of the current HPF specification.

7 Summary

Approximately 20 years of user experience with serial and vector architeures went into the definition of the Fortran 90 language. As a result, many of the deficiencies of Fortran 77 as a language for developing complex industrial codes have been addressed. However, many of the performance issues concerned with simply achieving fast execution time on the current and expected future generations of HPCC systems remained and HPF was

	Application (+ Comments)	HPF Features	HPF+ Features
1	Regular Grid PDEs	FORALL, CSHIFT	—
2	Panel Method	INDEPENDENT may require MIMD to achieve performance solvers–(library functions)	—
3	Vortex Method	FORALL or CSHIFT— non optimal performance (Optimization required for combining (blocking)) several shifts together)	Best performance violates "owner computes" rule. Do with library function?
4	Unstructured Grid	Not Appropriate	FORALL with reduction Partitioner Specification
5	Domain Decomposition Methods e.g. Irregularly Coupled Regular Meshes a) Iterate over each of the meshes independently on a subset of processors b) Interactions between meshes	a) FORALL b) Not Appropriate	a,b) Specification of subset of processors b) FORALL with reduction
6	Particle in a cell a) Assignment phase b) Maxwell equations c) Force on each particle	a) Not Appropriate b) FORALL c) Not Appropriate	a) b) c) Partitioner & Incremental Partitioner Specification c) FORALL with reduction
7	Structured Multigrid	FORALL but not optimal	Added Features needed to optimize data layout (see 4)
8	Unstructured Multigrid	Not Appropriate	FORALL with reduction "Alignment" needed between grids at different levels (also needed in 9)
9	Structured and Unstructured Adaptive Mesh (maybe multigrid)	FORALL in simple cases Not Appropriate in general	Define a MESH datatype —support with underlying library (HPC+class library) FORALL w/reduction

Table 3: The applicability of HPF for different CFD applications, HPF+ = HPF + Extensions

developed as a result. Features such as the **FORALL** construct are now clearly necessary in a *High Performance* Fortran.

It appears that the mechanism used to develop the HPF language definition in a practical yet rapid manner is a good one. HPF compilers are now becoming available for true HPCC systems.

HPF allows a fairly elegant and potentially efficient implementation of several problems including some of the key algorithms required by CFD applications of current industrial interest. However, there are certain problems such as those formulated by sparse data structures for which HPF is not a sufficient solution. Similarly, a standard way of expressing efficient parallel input/output may become essential as larger and larger applications problems are addressed. In the foreseeable future there is still a need for message passing in some applications although it is likely that message passing may ultimately achieve the status of assembler language programming today.

There is scope for further research and debate about how users could communicate certain optimisation possibilities to compilers and some of these will be addressed by HPF-II which is therefore to be greatly encouraged.

This paper has largely focussed on the *performance* features of Fortran 90 and HPF. Those feaures such as interface specification and module structures in Fortran 90 can make a considerable contribution to improving the software engineerability of very large industrial CFD codes and can also make a great contribution to design cycle turnaround times by making codes easier, and therefore cheaper, to maintain, develop and verify.

Acknowledgements

The authors are pleased to acknowledge funding for this work from NASA.

	Application (+ Comments)	HPF Features	HPF+ Features
1	"Crystalline"	CSHIFT	—
	Monte Carlo, e.g., QCD		
2	Quantum Monte Carlo	INDEPENDENT	—
3	Unindexed Text	INDEPENDENT	Scalable I/O
	Search in a database		
4	Analyze Physics data at Fermilab, CERN, etc.	INDEPENDENT	Scalable I/O Statistics Library
5	Full Matrix Algorithms	Library Functions Could express in HPF but modest performance	Library Functions
6	Fast Fourier Transform Little need of support for specialized class	FORALL–does not get top performance	Library Functions
7	Low level image processing e.g., Template Matching or Filtering	SHIFT FORALL	CMU Image Library based on HPF (Fortran FX, ADAPT)
8	CHARMM like Molecular Dynamics a) Calculate non-bonded interaction list b)Iterate c),d) several steps c)Calculate non-bonded interactions d) Calculate bonded interactions	$O(N^2)$ NBody Algorithm (see 10) c, d) Not appropriate	a–d) Partitioner Specification Reduction FORALL (c) and (d)
9	Direct Simulation Monte Carlo	FORALL but non optimal	FORALL and append to collection of cells (HPC++class) Partitioner & Incremental Partitioner Specification
10	Fast Multipole	Not appropriate	Language support important but unclear Dynamic Tree and mesh data structure (HPC++)
11	Region Growing a) Split Phase b) Merge Phase with Random tie breaking	a) FORALL	b) FORALL with reduction b) Dynamic Load Balancing
12	Monte Carlo Clustering Methods (Physics near critical points	SCAN Library Function	FORALL with reduction Dynamic Load Balancing
13	Direct Sparse Methods	Not appropriate	Library functions– language support not critical. Complex direct solvers could use HPF+ support for multiple dense solvers

Table 4: The applicability of HPF for other applications, HPF+ = HPF + Extensions

References

[1] Adams, J., Brainerd, W., Martin, J., Smith, B., and Wagener, J., *Fortran 90 Handbook: Complete ANSI/ISO Reference*, McGraw-Hill, 1991.

[2] Bailey, D., Barton, J., Lasinski, T. and Simon, H., Editors, "The NAS Parallel Benchmarks", NASA Ames, NASA Technical Memorandum 103863, July 1993.

[3] Barber, T., McNulty, G., and Hall, E., "Preliminary findings in certification of ADPAC," *Paper AIAA-94-2240* presented at 25th AIAA Fluid Dynamics Conference, Colorado Springs, CO, 20-23 June 1994,

[4] Bozkus, Z., Choudhary, A., Fox, G., Haupt, T., and Ranka, S., "Fortran 90D/HPF compiler for distributed-memory MIMD computers: design, implementation, and performance results," *Proceedings of Supercomputing '93*, Portland, OR, 1993, p.351.

[5] Buning, P., "OVERFLOW flow solver overview," presentation at MADIC workshop, NASA Langley, 16 September 1993.

[6] Carle, A., Green, L., Vischof, C., and Newman, P., "Applications of automatic differentiation in CFD," *Paper AIAA-94-2197* presented at 25th AIAA Fluid Dynamics Conference, Colorado Springs, CO, 20-23 June 1994,

[7] Chapman, B., Mehrotra, P., Mortisch, H., and Zima, H., "Dynamic data distributions in Vienna Fortran," *Proceedings of Supercomputing '93*, Portland, OR, 1993, p.284.

[8] Fox, G.C., Hiranadani, S., Kennedy, K., Koelbel, C., Kremmer, U., Tseng, C.W., and Wu, M., "Fortran D language specification," Technical Report, Rice and Syracuse Universities, 1992.

[9] Guruswamy, G., Obayashi, S., Byun, C., Tu, E., and Chaderjian, N., "ENSAERO — Euler/Navier Stokes code for integrated fluids/structures/controls/optimization analysis of aerospace vehicles," presentation at MADIC workshop, NASA Langley, 16 September 1993.

[10] Hawick, K.A., Bell, R.S., Dickinson, A., Surry, P.D., Wylie, B.J.N., "Parallelisation of the Unified Model Data Assimilation Scheme", Invited paper, *Proceedings of the Fifth ECMWF Workshop on the Use of Parallel Processors in Meteorology.*

[11] Hawick, K.A., and Wallace, D.J., "High Performance Computing for Numerical Applications", Keynote address, *Proceedings of Workshop on Computational Mechanics in UK*, Association for Computational Mechanics in Engineering, Swansea, January 1993.

[12] High Performance Fortran Forum (HPFF), "High Performance Fortran Language Specification," *Scientific Programming*, vol.2 no.1, July 1993. Also available by anonymous ftp from ftp.npac.syr.edu (cd /HPFF).

[13] Hiranandani, S., Kennedy, K., and Tseng, C.W., "Compiler support for machine-independent parallel programming in FortranD," in *Compiler and Runtime Software for Scalable Multiprocessors*, 1991.

[14] Hiranandani, S., Kennedy, K., and Tseng, C.W., "Preliminary experiences with the FortranD compiler," *Proceedings of Supercomputing '93,* Portland, OR, 1993, p. 338.

[15] Koelbel, C.H., Loveman, D.B., Schreiber, R.S., Steele, G.L., Zosel, M.E., "The High Performance Fortran Handbook", MIT Press 1994.

[16] Marconi, F., Siclari, M., Chow, R., and Carpenter, G., "Comparison of TLNS3D computations with test data for a transport wing/simple body configuration," *Paper AIAA-94-2237* presented at 25th AIAA Fluid Dynamics Conference, Colorado Springs, CO, 20-23 June 1994,

[17] Melnik, R., Barber, T., and Verhoff, A., "A process for industry certification of physical simulation codes," *Paper AIAA-94-2235* presented at 25th AIAA Fluid Dynamics Conference, Colorado Springs, CO, 20-23 June 1994,

[18] Metcalf, M., Reid, J., "Fortran 90 Explained", Oxford, 1990.

[19] Miller, C.J., "Advanced Ducted Propfan Analysis Codes," presentation at MADIC workshop, NASA Langley, 16 September 1993.

[20] PARKBENCH Committee, "Public international benchmarks for parallel computers," to submitted to *Scientific Programming*, 1994. (Also available by anonymous ftp from ecs.soton.ac.uk (pub/benchmarks).)

[21] Raj, P., and Siclari, M., "Toward certifying CFD codes using wing C and M100 wing-body configurations," *Paper AIAA-94-2241* presented at 25th AIAA Fluid Dynamics Conference, Colorado Springs, CO, 20-23 June 1994,

[22] Robinson, B., LaBozzetta, W., and Verhoff, A., "Preliminary findings in certification of OVERFLOW," *Paper AIAA-94-2238* presented at 25th AIAA Fluid Dynamics Conference, Colorado Springs, CO, 20-23 June 1994,

[23] Robinson, B., and Yeh, D., "Toward certification for CFD codes for aft-end/nozzle configurations," *Paper AIAA-94-2242* presented at 25th AIAA Fluid Dynamics Conference, Colorado Springs, CO, 20-23 June 1994,

[24] Rumsey, C., "CFL3D," presentation at MADIC workshop, NASA Langley, 16 September 1993.

[25] Vasta, V.N., and Sanetrik, M.D., "TLNS3D — An overview," presentation at MADIC workshop, NASA Langley, 16 September 1993.

[26] Verhoff, A., Robinson, B., and LaBozzetta, W., "Preliminary findings in certification of CFL3D," *Paper AIAA-94-2236* presented at 25th AIAA Fluid Dynamics Conference, Colorado Springs, CO, 20-23 June 1994,

[27] Yeh, D., and Worthy, M., "Preliminary findings in certification of ENSAERO," *Paper AIAA-94-2239* presented at 25th AIAA Fluid Dynamics Conference, Colorado Springs, CO, 20-23 June 1994,