# Evaluation of HPF Products on the Cray-T3D

*Michael Ess,* Arctic Region Supercomputing Center

**ABSTRACT:** *High Performance Fortran is the emerging standard beyond Fortran 90 and is targeted for parallel machines. Although CRI is monitoring this standard closely, there is no HPF product currently available from CRI. There are two commercial offerings of HPF for the T3D, one from the Portland Group and the other from Applied Parallel Research. Both of these products work in conjunction with CRI's existing Fortran compilers. This report surveys and evaluates these two products. Comparisons are made to the existing CRI products of Craft Fortran and Fortran 90. The evaluation covers functionality, ease of use and performance.*

## Fortran Background

From the beginnings in 1957[1], Fortran has been a popular programming language. With a few hours of effort, an engineer or scientist could become functional and start producing useful results. After a year of working with Fortran, a user could feel quite comfortable. In the 1960s and 70s, each vendor implemented their own version of Fortran with minor differences. But with the release of Fortran 77[2], many of these additions were added to the standard. Fortran 77 retained the simplicity of the original language and because the additions in Fortran 77 were relatively small and may had been implemented already, Fortran 77 compilers were available from each vendor quickly after the standard was adopted.

With the introduction of Fortran 90[3], this situation changed. The small, simple Fortran language became a modern, big language with many more features and capabilities. Commonly accepted features like dynamic memory allocation and pointers were obvious features added to Fortran 90. Additional capabilities like: modules, generalized precision, and overloaded operators made the language much bigger and more complex. The underlying model of Fortran 77 had been a simple von Neumann machine, but Fortran 90 had to assume a more general model to service the new architectures. The addition of array syntax to Fortran 90 was a step in this direction. Because Fortran 90 was so much bigger and more complex than the original Fortran 77, many current architectures still do not have complete Fortran 90 compilers.

High Performance Fortran, HPF,[4] comes after Fortran 90 and is targeted for parallelism distributed among processors. HPF builds upon the capabilities of Fortran 90, like array syntax, but goes on to add capabilities to achieve high performance on parallel machines. Unlike vectorization, which exploits the parallelism at the lowest level of a DO loop, parallelization across multiple processors is most efficient when implemented at the highest code level possible. Because a compiler has too many choices to make at such a high level, the features of HPF allow the user to direct the compiler strategy. This produces a conflict between the simple Fortran of the past and HPF with its need for additional information from the users. Built on top of the incomplete implementations of Fortran 90, it is hard to get comfortable with these languages because of their changing capabilities.

Even with this uncertain situation, it still seems inevitable that HPF must be part of the future of parallel computing. There is so much legacy Fortran and yet so much demand for additional performance, that a solution other than HPF is hard to imagine.

## T3D Background

The T3D[5] is the typical MIMD machine and therefore a target for HPF. Currently on the T3D, there are two programming paradigms: message passing and worksharing. Message passing is implemented with library calls which instrument the user's code. The library calls are implemented with a mechanism at the operating system level to send and deliver these messages. With a message passing solution, users must modify their code, ensure correctness of these changes and optimize the size and speed of the messages passed between processors. These are difficult tasks and they have slowed parallel implementations and reduced user interest in parallel machines.

Worksharing has been CRI's proposed solution to retaining the Fortran model on the T3D and is implemented in the Craft Fortran compiler, *cf77*. The defining document[GGG] was released on January 21, 1992, long before the HPF standard was finalized, but in time for the delivery of the T3D. Craft Fortran finds parallelism in DO loop nests with array references. The user chooses these nests by specifying the memory distribution for the arrays and the distribution of loop indices among the

processors. The burden of instrumenting the code with calls to a message passing library is handled by the compiler, the compiler ensures correct results and incurs low overhead in calling the message passing library. The user's tasks are now to optimize the distribution of the arrays and determine which loops should execute in parallel.

The implementation of Craft Fortran faces the same problems of HPF and HPF follows many of the same strategies as Craft Fortran. Although the two approaches have much in common, unfortunately they have different syntaxes and different capabilities.

## CRI products

Although Craft Fortran[6] has great potential, most "high performance" applications on the T3D have been message passing implementations. While Craft Fortran has been relatively easy to use, the restriction to loops nests, the power of 2 requirement for array dimensions, and the quality of code produced have hindered acceptance. Because the HPF standard has a different syntax than Craft Fortran, users are not anxious to use this nonstandard implementation. Craft Fortran for the T3D is now in maintenance mode with no future features planned.

Parallel to the Craft Fortran effort, which was based on Fortran 77, CRI has developed a new Fortran 90 compiler, *f90*. CRI used this Fortran 90 compiler to support early use of the T3D in the seismic industry[7]. Its most popular feature is that it supports a 32 bit floating point data type, accessible with the REAL*4 declaration.This product will eventually replace the Fortran 77 and Craft Fortran compiler, *cf77*, on both the YMP and T3E product lines.

CRI's future plans for Fortran and HPF on the T3D and T3E are explained in the Cray Research Service Bulletin [8]. CRI's attitude is that HPF, as currently defined and implemented, has neither high performance nor portability. For the T3E, CRI plans to release Craft90 as an extension of its current *f90* compiler with most, but not all, of the current Craft Fortran features. A shorten list of features and changes from Craft Fortran to Craft 90 is:

1. DOSHARED construct from Craft
2. Load time value for N$PES
3. MASTER/ENDMASTER directives
4. BARRIER/NOBARRIER directives
5. PE_RESIDENT directive
6. Shared array intrinsics
7. Compatibility with SHMEM and PVM
8. No power of 2 restrictions on array sizes
9. No :BLOCK(N) distribution (only :BLOCK or : )

Certainly Craft90 will not become the industry standard and eventually users of Craft Fortran and Craft90 will have to reconcile their codes with HPF.

## Portland Group Product

The Portland Group, started in 1989, producing a i860 compiler for Intel Supercomputers. In 1993 they introduced a parallelizing Fortran compiler for intel's Paragon computer. Soon afterwards, they produced parallelizing compilers for Meiko's CS-2, Fujitsu's AP1000, and workstation clusters running Solaris. PGI was a founding member of HPF Forum and introduced their retargetable HPF compiler[9] in November 1993. Their first T3D HPF product was available for testing in May, 1995.

PGI's compiler, called *pghpf*, takes Fortran 77, Fortran 90 and HPF source as input. *pghpf*, produces a program written in Fortran 77 with calls to PGI's library of communications and utility routines which is then compiled and linked by CRI's Fortran 77 compiler, *cf77*. A typical use of *pghpf* is:

```
pghfp -Mkeepftn -Mautopar -Minfo hello.hpf
```

This command would produce an executable for the T3D. Although *pghpf* is capable of producing the executable, the -Hkeepftn switch produces the Fortran 77 source for the user's inspection. The switch -Hautopar directs *pghpf* to parallelize the original source if possible and the switch -Hinfo produces compiler statistics and describes the code transformations were used. The libraries provided by PGI are implemented using the SHMEM technology[11]. The version of *pghpf* used in this report was Release 1.3 which was a beta copy for testing purposes only. The first production release will be Release 2.0 and will be out by next month, the results presented here are for a beta version.

## Applied Parallel Research Product[10]

APR was founded in 1991 and was also a founding member of the HPF Forum. APR had previously produced Fortran browsers and parallelizers as part of Pacific-Sierra Research. In April 1993, they produced their first HPF product for clusters of workstations using PVM, the Intel Paragon and the IBM SP1. The first T3D version using SHMEM technology, was available in August,1993. APR's HPF compiler, *xhpf*, is actually part of a much larger product, *FORGEX*, which includes a Fortran browser and analyzer. *FORGEX* also has the ability to use runtime statistics to help *xhpf* generate optimal code.

This evaluation only considered the automatic parallelization of Fortran programs and their implementation of HPF, but APR's approach may be correct in that optimal parallelization is not possible without including runtime information.

A typical use of the APR compiler might look like:

```
xhpf -plist=report -Auto=1 -ompf greeting.f
```

This command would produce the executable for the T3D. The Fortran 77 program produced for CRI's *cf77* compiler by *xhpf* is in a user accessible file _mpf.f. The -Auto=1(the default) has told *xhpf* to compile the source program choosing the most

important loop as the template for the data layout and then use that layout for the remaining loops in the unit. The -plist switch has *xhpf* write a package of information for further study with the FORGEX tools. The -ompf switch is needed to produce a message passing version implemented with SHMEMs. The version of *xhpf* used for this report was version 2.0(29).

## Example #1 - Simple relaxation

As a simple test program to exercise each HPF compiler, I tried a lab problem from my introductory T3D class. This program consists of a simple 2 dimensional 100 by 100 grid whose values are initialized to zero except for one spike in the center. Then a simple relaxation operator sweeps over the grid:

```
p(i,j) = (p(i+1,j)+p(i-1,j)+p(i,j+1)+p(i,j-1)+4*p(i,j))/8
```

This relaxation is applied between two copies of the 2 dimensional grid until the difference between them is small. So the flow of the program looks like:

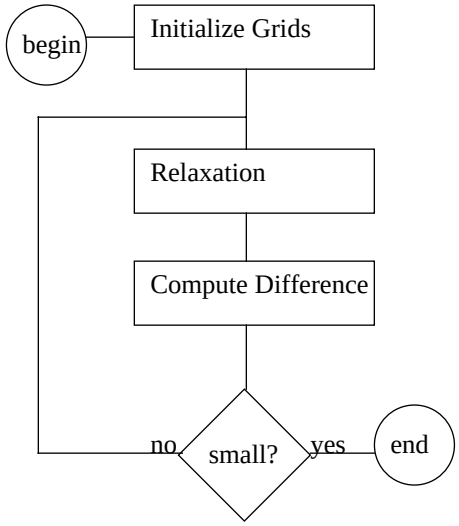


**Figure 1: Flow chart for Relaxation Problem**

For this program, the execution time was measured for each three computational areas:

1. Initialization of the 2D grid

```
      DO 20 J = 1, N
         DO 10 I = 1, N
            P0(I,J) = 0.0
   10 CONTINUE
   20 CONTINUE
```

2. The relaxation:

```
      DO 40 J = 2, N-1
         DO 30 I = 2, N-1
            P1(I,J) = (P0(I+1,J)+P0(I-1,J)+
                      P0(I,J-1)+P(I,J+1)+4.0*P(I,J))/8.0
   30 CONTINUE
   40 CONTINUE
```

3. The residual computation:

```
      ERROR = 0.0
      DO 80 J = 2, N-1
         DO 70 I = 2, N-1
               ERROR = ERROR + (P0(I,J)-P1(I,J)) ** 2
   70 CONTINUE
   80 CONTINUE
```

For this lab problem, the value of N was 100 and the problem was run on 8 PEs. The table below is a summary of execution times for each of the three timed code segments. The program executing on a single PE took 1.8445 seconds.

Table 1. Compiler results on Relaxation Lab Problem

| Code segment | Craft Fortran | pghpf PGI | xhpf APR |
|---|---|---|---|
| Array Initialization | .0023 | .0009 | .0032 |
| Total Relaxation Time | .5372 | .3304 | .3625 |
| Total Residual Compute Time | .0841 | .3977 | .2664 |

The Craft Fortran modifications included:

1. Resize arrays to 128 by 128
2. Declare arrays as CDIR$ SHARED P0(:,:block ), P1(:,:block )
3. Add DO SHARED constructs to each loop nest
4. Rewrite the residual computation to form partial sums on each PE

For me, it was a surprise that both HPF compilers did all of these optimizations and more using only the default parallelization switches and with no modifications to the original source code.

## Example #2 - NAS EP Benchmark

Another program that is not as easy for the HPF compilers is the EP benchmark from the NAS Parallel Benchmark(NPB) Suite[12]. Examples of the Fortran 77, Fortran 90 and HPF source codes for this and some of the other benchmarks in the suite have been written by David Serafini of Computer Science Corporation and Rice University and are available on at:

http://www.nas.gov.NAS/NPB/software/npb-software.html

Table 2. shows the initial performance on these sources for several compilers running on a single PEs. It is an interesting exercise to examine the three sources and see how both Fortran 90 and HPF Fortran sources are different from Fortran 77 source.

Table 2. Initial compiler results on Serafini's source codes, class A

| compiler | Fortran 77 execution time (seconds) | Fortran 90 | HPF Fortran |
|---|---|---|---|
| cf77 (CRI) | 1663.6 | Not appropriate | Not Appropriate |
| f90 (CRI) | 1363.3 | Needed features: implementation deferred(1) | Not Appropriate |
| pghpf (PGI) | 1363.3 | With source code corrections(2) | With source code corrections(2) |
| xhpf (APR) | 1368.5 | Free format source code not yet implemented(3) | Free format source code not yet implemented(3) |

Notes:

1. The CRI Fortran 90 compiler does not currently
2. implement modules.
3. There were two modifications to the source code.
   a. The source code incorrectly places an "implicit none" statement.
   b. The source code incorrectly tries to align a static array with an allocated array.
4. The APR compiler currently does not accept free formatted source code, but will later this year.

Although all auto parallelizing directives were used, neither the PGI nor APR HPF compilers generated code that made use of more than one PE on the Fortran 77, Fortran 90 or HPF Fortran sources.

Both PGI and APR had kindly provided their HPF sources for this benchmark and together with the reported results from CRI[13] we have Table 3. CRI has not released the hand crafted source for their results, so their reported results have no educational value for their users (or their competitors), and the times presented for CRI are not gotten with Craft Fortran but this calls to Shmem. The times reported for *pghpf* and *xhpf* are from runs on the ARSC's T3D running the 1.2 PE.

The PGI effort to optimize the EP benchmark was a significant rewrite from the original Fortran source. The original problem contained $2^{28}$ samples divided into nested loops of $2^{16}$ and $2^{12}$ iterations each. In the PGI effort, the innermost DO loop

Table 3. Execution speeds on the NAS EP benchmark, Class A

| Number of PEs | CRI reported results | pghpf PGI | xhpf APR |
|---|---|---|---|
| 8 | Not reported | 170.7 | 171.2 |
| 16 | 22.74 | 85.7 | 85.6 |
| 32 | 11.37 | 43.7 | 42.8 |
| 64 | 5.68 | 23.2 | 21.4 |
| 128 | 2.87 | 13.8 | 10.8 |

was further divided into "slices" of samples of size $2^8$, $2^9$ or $2^{10}$. Operations on these slices were then handled with array syntax. Conditional operations within the DO loop nest were handled with the

```
WHERE - ELSEWHERE - ENDWHERE
```

construct. This implementation also used the KIND intrinsic to specify floating point constants and the SUM intrinsic to do sum reductions on the "slices".

The APR source for the EP benchmark contained only a single HPF directive:

```
!HPF INDEPENDENT
```

before the major loop.

## Example #3 - NAS FT Benchmark

This benchmark code uses FFTs to solve a three dimensional partial differential equation. It is typical of spectral methods and is challenging for distributive memory machines because the FFTs require many non local memory accesses. The problem consists of a 256x256x128 parallelepiped of points transformed into the frequency space with a 3 dimensional FFT. After being scaled in the frequency space, the points are transformed back with an inverse FFT. This benchmark is exceptional for the NPB because it allows the use of library routines to perform the FFTs. A summary of results is shown in Table 4..

Table 4. Execution speeds on the NAS FT benchmark, Class A

| Number of PEs | CRI reported results | pghpf PGI | xhpf APR |
|---|---|---|---|
| 16 | 11.80 | out of memory | out of memory |
| 32 | 5.90 | out of memory | 42.88 |
| 64 | 2.99 | 6.30 | 22.33 |
| 128 | 1.52 | 3.85 | 12.33 |

The PGI effort to optimize the FT benchmark makes extensive use of the HPF data layout capabilities. Below are all of the declarations for REAL and COMPLEX arrays at the main program level:

```
COMPLEX(KIND=C16), DIMENSION(N1,N2,N3) :: X1, X2, WORK
REAL(KIND=R8), DIMENSION(N1,N2,N3) :: X3
COMPLEX(KIND=C16), DIMENSION(N1+N2+N3) :: TABLE

!HPF$ TEMPLATE T1(N3)
!HPF$ DISTRIBUTE T1(BLOCK)
!HPF$ ALIGN (*,*,:) WITH T1(:) :: X1, X2, X3, K3
HPF$ TEMPLATE T2(N2)
!HPF$ DISTRIBUTE T2(BLOCK)
!HPF$ ALIGN (*,:,*) WITH T2(:) :: WORK
!HPF$ DISTRIBUTE T(CYCLIC)
```

In the original code the 3D FFT was implemented with 3 calls to a series of 2 dimensional FFTs and 3 transposes. This code had been replaced with a call to a PGI library routine. The data layout choices were made to accommodate this library routine. PGI also makes use of the RANDOM_NUMBER intrinsic of Fortran 90 and it replaces the subroutines provided in the original F77 source.

APR makes extensive use of the APR extensions to the HPF data layout features. These features allow each of the three passes of the 3 dimensional FFT to be done with a nest of similar DO loops but with a different set of directives prefixing each DO loop nest. In more detail, we have the three passes of 1 dimensional FFTs as:

```
CAPR$ PARTITION X1IMAG(*,*,SHRUNKBLOCK)
CAPR$ PARTITION X1REAL (*,*,SHRUNKBLOCK)
CAPR$ DO PAR ON X1REAL<:,:1~1>
CAPR$ IGNORE ALL INHIBITORS
CAPR$ IGNORE ALL COM
C first nest of 1D FFTs
.
.
.
CAPR$ DO PAR ON X1REAL<:,:,1~1>
CAPF$ IGNORE ALL INHIBITORS
CAPR$ IGNORE ALL COM
C second nest of 1D FFTs.
.
.
.
CAPR$ PARTITION X1IMAG (*,SHRUNKBLOCK,*)
CAPR$ PARTITION X1REAL (*,SHRUNKBLOCK,*)
CAPR$ DO PAR ON X1 REAL<:,1~1,:>
CAPR$ IGNORE ALL INHIBITORS
CAPR$ IGNORE ALL COM
C third nest of 1D FFTs
.
.
.
```

This use of directives to implement a redistribution of the points within a subroutine is an extension beyond HPF. Similarly the added IGNORE directives provide for situations when the single:

```
!HPF INDEPENDENT
```

directive of HPF did not convey enough information to the compiler. The APR effort retained the random number generator of the original Fortran source and implements the 1 dimensional FFT with straight forward Fortran 77 code.

## Conclusions

Users are interested in HPF because it holds out the possibility of high performance when using a standard portable product. Designers of Fortran 77 had ten years of experience before deciding what features would become part of that standard. F90 and HPF designers have not had enough time to implement what has become accepted and effective. They have moved more aggressively to define what should be in the standard before it was acceptable to users and vendors or proven to ensure high performance. This has resulted in incomplete and immature implementations of both Fortran 90 and HPF. In this situation users will wait until the claims of both portability and high performance are convincing.

## References


1. J. Backus et. al. (1957) *The Fortran automatic coding system*, Western Joint Computer Conference.

2. American National Standards Institute, Inc., New York, NY. *American National Standard Programming Language FORTRAN*, ANSI X3.9-1978

3. International Organization for Standardization and International Electrotechnical Commission. *Fortran 90*, ISO/IEC 1539: 1991(E)], also as ANSI X3.198,1992

4. High Performance Fortran Forum. High Performance Fortran Journal of Development. *High Performance Fortran language specification, version 1.0.* TR93300, CPRC, Rice University, Houston, TX 1992.

5. Subhash Saini and Horst D. Simon, High Performance Programming Using Explicit Shared Memory Model on the Cray T3D, Proceedings of the Cray User's Group Meeting, Spring 1994

6. D. Pase, T. MacDonald, and A. Meltzer. *MPP Fortran Programming Model*, SN-2513, Cray Research, Eagan, MN

7. *Signal-Processing(32-bit precision) Support for the CRAY T3D Systems*, SN-2191 1.2 Cray Research, Eagan MN

8. *Fortran parallel programming strategies and directions for the CRAY T3D and CRAY T3E systems*, Cray Research Service Bulletin, Vol 5, #5, May 1995

9. PGHPF User's Guide, The Portland Group Inc. Wilsonvile, OR

10. John M. Levesque, *Using High Performance Fortran on the Cray T3D*, Proceedings of the Cray User's Group meeting, Spring 1994

11. SHMEM Technical Note for Fortran, SN-2516 2.3, Cray Research, Eagan MN

12. D. Bailey et. al. The NAS PARALLEL BENCHMARKS, Technical Report RNR-94-007, March, 1994

13. Subhash Saini and Dave Bailey, NAS Parallel Benchmarks Results 3-95, Report NAS-95-011, April, 1995