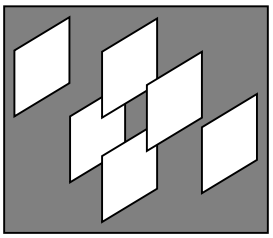
Performance of the Acoustic Wave Propagation Problem in High Performance Fortran

by

Kevin P. Roe and Tomasz Haupt

12 July 1995



Northeast Parallel Architectures Center
at Syracuse University
Science and Technology Center
111 College Place
Syracuse, NY 13244-4100

Contents

SCCS 700

Performance of the Acoustic Wave Propagation Problem in High Performance Fortran

Kevin P. Roe and Tomasz Haupt

12 July 1995

Northeast Parallel Architectures Center
at Syracuse University
Science and Technology Center
111 College Place
Syracuse, NY 13244-4100

email: kproe@npac.syr.edu
http://www.npac.syr.edu/users/kproe/homepage/index.html

Abstract

The physical problem of wave propagation is examined as well as the numerical methods used in its solution. The wave propagation problem was implemented in sequential FORTRAN 77, sequential FORTRAN 90, and High Performance Fortran (HPF). Performance of each implementation was examined and compared. The results of four HPF compilers (Digital, Applied Parallel Research, IBM, and the Portland Group Inc.) are compared to show how each compiler handles this type of code.

1 The Acoustic Equations

1.1 Derivation of the Acoustic Equations

The acoustic wave propagation problem is the simplest example of a fluid mechanics problem which involves the solution of a system of first-order linear partial differential equations.
The governing equations for fluid flow are the laws of conservation of mass, momentum, and energy. If inviscid, unsteady, one dimensional flow is considered then the basic laws can be expressed as:

$$Mass : \frac{\partial \rho}{\partial t} + \rho \frac{\partial u}{\partial x} + u \frac{\partial \rho}{\partial x} = 0, \tag{1}$$

$$Momentum : \rho \frac{\partial u}{\partial t} + \rho u \frac{\partial u}{\partial x} + \frac{\partial p}{\partial x} = 0, \tag{2}$$

and

$$Energy : \frac{\partial p}{\partial t} + u \frac{\partial p}{\partial x} - a^2 \left(\frac{\partial \rho}{\partial t} + u \frac{\partial \rho}{\partial x} \right) = 0, \tag{3}$$

where ρ is the density, u is the fluid velocity, p is the static pressure, and a is the speed of sound. Since acoustic problems examine the motion of small amplitude disturbances in a fluid medium, small perturbations in density, velocity, and pressure are considered

$$\rho = \rho_0 + \rho', \tag{4}$$

$$p = p_0 + p', \tag{5}$$

$$u = u_0 + u' = u', \tag{6}$$

where (ρ_0, p_0, u_0) are the undisturbed properties of the fluid and (ρ', p', u') are small perturbations. The condition $u_0 = 0$ is used for a stagnant fluid. The speed of sound is determined from

$$a = \sqrt{\frac{\gamma p}{\rho}} \tag{7}$$

Substituting the above relations into the governing equations for fluid flow and neglecting second-order terms, the linearized equations governing acoustics are

$$\frac{\partial u'}{\partial t} + \left(\frac{1}{\rho_0}\right)\frac{\partial p'}{\partial x} = 0 \tag{8}$$

and

$$\frac{\partial p'}{\partial t} + \left(\rho_0 a_0^2\right)\frac{\partial u'}{\partial x} = 0. \tag{9}$$

The case of an infinitely long one-dimensional duct is examined so that the b.c.'s do not enter into the solution. The initial conditions consist of stagnant fluid and a pressure perturbation. The propagation speed, c, is fixed and the perturbation is allowed to propagate up and down the tube for a fixed number of time steps.

2 Numerical Techniques

The acoustic equations are an example of a system of two coupled first-order hyperbolic equations [1] of the general form:

$$f_t + c g_x = 0 \tag{10}$$

$$g_t + c f_x = 0 \tag{11}$$

where

$$f \equiv u' \qquad g \equiv \left(\frac{1}{\rho_0 a_0}\right) p' \qquad c \equiv a_0 \tag{12}$$

The above system of partial differential equations is solved using the Lax method. This method is an explicit, first-order accurate, central difference scheme. This method was chosen because it is the simplest method used to solve this system of equations and will be a baseline test case for the performance of HPF. If more complex algorithms are used then the performance of a parallel program should also increase. Lax's method uses a central difference scheme which requires information from one grid point to the right and one to the left (see below stencil).

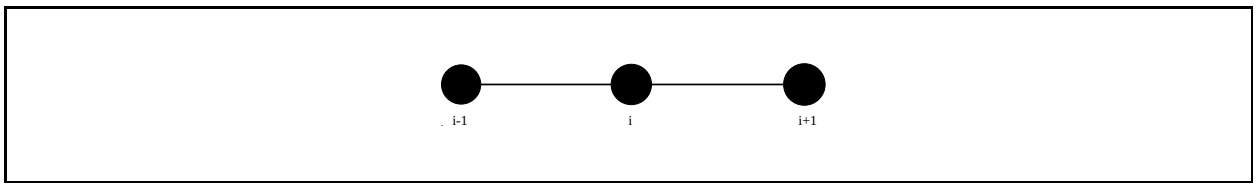


Figure 1: Nearest Neighbor Communication is Required for the 3 Point Stencil

3 Development of Fortran Code

The program was initially written in sequential FORTRAN 77 and then converted into sequential FORTRAN 90. The code was then parallelized using High Performance FORTRAN.

3.1 Sequential FORTRAN 77

The code was compiled with the DEC, the Intel, and the IBM SP-2 FORTRAN 77 compiler. The program was written with and without subroutines to show that there was no significant difference in performance timings. The main reason for writing the code without subroutines was because the available version of Digital's compiler did not support the descriptive ALIGN command [2, 3], which will later be needed for HPF.

3.2 Sequential FORTRAN 90

The code was compiled with the DEC FORTRAN 90/HPF and the IBM SP-2 FORTRAN 90 compiler. Three versions were written, however both were written with no subroutines. The first version uses do-loops in exactly the same way FORTRAN 77 uses them. The second version uses array sections in place of do-loops. The third version is the same as the first, but the do-loops have been replaced by forall statements and the HPF directives have been added. This version could only be done with the DEC FORTRAN 90/HPF compiler because the IBM SP-2 compiler does not accept FORALL statements. There should be no problem with the addition of HPF directives since they will just be seen as comments. The use of FORALL statements produce problems [2] and will be discussed in more detail in the following sections.

```
do i=1,mx-1
   do l=1,lmax
      d(l,i)=0.5*(dx/dt)*(u(l,i+1)-u(l,i))
   end do
end do

  ⇓         ⇓         ⇓

d(1:lmax,1:mx-1)=0.5*(dx/dt)*(u(1:lmax,2:mx)-u(1:lmax,1:mx-1))

  ⇓         ⇓         ⇓

forall(i=1:mx-1,l=1:lmax)
@   d(l,i)=0.5*(dx/dt)*(u(l,i+1)-u(l,i))
```

Figure 2: Conversion of DO-Loops to Array Sections and FORALL Statements

3.3 High Performance FORTRAN

The code was compiled with Digital's FORTRAN 90/HPF compiler, PGI HPF compiler, and IBM's HPF compiler. Using Digital's FORTRAN 90/HPF compiler, the second version of the FORTRAN 90 code was used (array sections). The HPF code used in the above three compilers is exactly the same as the third version of the FORTRAN 90 code. When the same code is compiled with the HPF options, the HPF directives are now understood by the compiler. The directive PROCESSORS [2, 3] was used to set the number of processors at compile time. The DISTRIBUTE [2, 3] directive is used to distribute one array over the processors. The ALIGN [2, 3] directive is used to align the other arrays with the array that is already distributed; this is done to reduce interprocessor communication.

```
      real x(imax),u(lmax,imax)
      real res(lmax,imax),e(lmax,imax),d(lmax,imax)
      ..........
!HPF$ distribute (block) onto pp :: x
      ...........
!HPF$ align (*,:) with x(:)      :: u,res,e,d
      ...........
```

Figure 3: Code Fragment for HPF Data Mapping

The conversion to HPF with APR's (Applied Parallel Research) compiler was done in a different way. The HPF code was sent through APR's precompiler xHPF and the ouputted source code was then compiled with FORTRAN 77 using APR's libraries.

4 Performance Timings of FORTRAN Code

All timings on the DEC alpha farm were done at the Northeast Parallel Architecture Center (NPAC) at Syracuse University with reserved time (exclusive use of machines). The DEC alpha farm used consists of 8 workstations. These workstations are connected together via a FDDI network which connects to a DEC gigaswitch. Timings for the PGI compiler were done on the 56 node Paragon at JPL (Joint Propulsion Laboratories), the 15 node Paragon at PGI, the 64 node Paragon at NASA Ames, and the 72 node Paragon at NASA Langley. Timings for the APR compiler were done on the Paragon at NASA Langley. Timings for the IBM, PGI, and APR compilers were done on the IBM SP-2 at NASA Langley and NASA Ames. All timings are an average of five to ten runs and the timing for each run is from the node with the highest execution time.

4.1 Timing Sequential FORTRAN 77 Code

Timings were done in two ways: the first with the *time* statement to get the CPU time from the outputted *user* plus *system* time [4], the second was through the use of the functions (*etime, timef,* and *dclock()*) [5]. The code was optimized with the default optimization (-O). On the DEC Alpha cluster, optimization was automatically done by the compiler. On the IBM SP-2, the optimization had to be specified at compile time. On the Paragon the code was compiled with the (-O) option as well as the (-Knoieee) option.

4.2 Timing Sequential FORTRAN 90 Code

Timings were done in three ways: the first with the *time* statement to get the CPU time from the outputted *user* plus *system* time [4], the second was through the use of the function *etime* [5] for the IBM SP-2, the third was done with the FORTRAN 90 function *"secnds"* on the DEC Alpha Farm. In this case, all methods returned approximately the same CPU time. This means that the methods used for FORTRAN 77 timings are comparable to those of FORTRAN 90.

4.3 Timing High Performance FORTRAN Code

On the DEC Alpha Farm, timings were done with the FORTRAN 90 *"secnds"* which was distributed onto each processor. This is comparable to the timings taken in FORTRAN 90 with the same function *"secnds"*. Since the timing results for all methods in FORTRAN 90 yielded approximately the same results, the other timing methods can be considered to be comparable. This then extends the compatibility of the timing method for HPF to the timing methods of FORTRAN 77 on the DEC Alpha Farm. Timing on the Paragon was done with the function *dclock()*. Timing on the IBM SP-2 was done with the function *timef*.

5 Results

5.1 DEC Alpha Farm

Digital's HPF run on the Alpha Farm did not produce as good a results as expected. Speedup times relative to FORTRAN 77 show very poor results for HPF and a small improvement for FORTRAN 90. The FORALL statement is accepted by Digital's FORTRAN 90 compiler but not well optimized as can be seen from the results [2].

Compiler	CPU Time (sec)	Speedup Relative to FORTRAN 77	Speedup Relative to HPF with 2 Processors
F77	14.067	1.000	
F90 (with FORALL)	157.875	0.089	
F90 (without FORALL)	13.682	1.028	
HPF with FORALL			
1 Processor	23.535	0.598	
2 Processors	94.707	0.149	1.0
3 Processors	73.886	0.190	1.282
4 Processors	56.970	0.247	1.662
5 Processors	43.990	0.320	2.153
6 Processors	41.761	0.337	2.268
7 Processors	40.501	0.347	2.338
8 Processors	37.572	0.375	2.521
HPF with Array Sections			
1 Processor	14.300	0.984	
2 Processors	39.531	0.356	1.0
3 Processors	34.262	0.411	1.153
4 Processors	28.675	0.491	1.378
5 Processors	25.432	0.553	1.554
6 Processors	23.977	0.587	1.648
7 Processors	22.953	0.613	1.722
8 Processors	21.481	0.655	1.840

Table 1: Application with 5001 grid points on DEC Alpha Farm

HPF using FORALL statements with 1 processor shows a drop in speedup most likely due to the extra work involved in data distribution and mapping. HPF with two processors shows a dramatic decrease in speedup due to the communication now existing between processors. The performance slowly increases with the number of processors, but even with eight processors the performance never reaches a speedup of one compared to the sequential FORTRAN 77. Although there is an increase in performance as the mesh size

is doubled, the eight processor case still only reaches a speedup of approximately half that of FORTRAN 77. HPF using array sections follows similar trends as in the use of FORALL statements, but with quicker execution times. HPF with array sections shows better speedup relative to FORTRAN 77.
If speedup relative to HPF with 2 processors is examined, the results are much more promising. The speedup curves appear to be relatively linear initially and then deviating as the number of processors is increased. The figure also shows how as the mesh size is increased, the speedup becomes more linear for higher number of processors. This is because more work is being done by each processor while communication remains the same.
There are many possible reasons for HPF's poor performance. However, the most likely reason is due to the compiler's infancy. The communication between processors may not be well optimized. Another reason for the poor performance may be due to the hardware. The DEC Alpha chip is very fast and communication is most likely quite expensive compared to computation.

Compiler	CPU Time (sec)	Speedup Relative to FORTRAN 77	Speedup Relative to HPF with 2 Processors
F77	59.075	1.000	
F90 (with FORALL)	677.555	0.087	
F90 (without FORALL)	54.054	1.093	
1 Processor	96.555	0.612	
2 Processors	363.833	0.163	1.0
3 Processors	255.571	0.231	1.418
4 Processors	202.129	0.292	1.793
5 Processors	167.180	0.353	2.168
6 Processors	126.011	0.469	2.577
7 Processors	120.681	0.490	3.015
8 Processors	116.960	0.505	3.099
HPF with Array Sections			
1 Processor	56.984	1.037	
2 Processors	157.971	0.374	1.0
3 Processors	107.866	0.547	1.465
4 Processors	89.022	0.664	1.775
5 Processors	77.753	0.760	2.031
6 Processors	69.718	0.847	2.266
7 Processors	63.132	0.936	2.502
8 Processors	60.422	0.978	2.614

Table 2: Application with 10001 grid points on DEC Alpha Farm

5.2 IBM SP-2

IBM's FORTRAN 90 performed slightly worse than FORTRAN 77. IBM's HPF run on the IBM SP-2 also did not show very promising results. The HPF code ran slower than the sequential FORTRAN 77 and FORTRAN 90 codes as expected due to the increase in work from data distribution and mapping. As the number of processors was increased from 1 to 2 the execution time greatly increased; this again is most likely due to the present communication. However, there was little change in time when the number of processors was increased.

Compiler	CPU Time (sec)	Speedup Relative to FORTRAN 77	Speedup Relative to HPF with 2 Processors
F77	20.340	1.000	
F90	24.640	0.825	
HPF with FORALL			
1 Processor	29.595	0.687	
2 Processors	74.764	0.272	1.000
4 Processors	73.571	0.276	1.016
8 Processors	70.670	0.288	1.058

Table 3: Application with 5001 grid points on IBM SP-2

Compiler	CPU Time (sec)	Speedup Relative to FORTRAN 77	Speedup Relative to HPF with 2 Processors
F77	83.210	1.000	
F90	92.830	0.896	
HPF with FORALL			
1 Processor	118.409	0.703	
2 Processors	300.430	0.277	1.000
4 Processors	299.560	0.278	1.003
8 Processors	295.789	0.281	1.016

Table 4: Application with 10001 grid points on IBM SP-2 using IBM's HPF compiler

APR's xHPF compiler performed very well with a few processors, but results tended to taper off quickly as the more than 4 processors were used. As can be seen from the table below, the code precompiled with xHPF and compiled with mpxlf produced slightly better results for 1 processor than the sequential code using xlf.

Compiler	CPU Time (sec)	Speedup Relative to FORTRAN 77	Speedup Relative to HPF with 1 Processor
F77	66.25	1.00	
HPF with FORALL			
1 Processor	65.98	1.00	1.00
2 Processors	37.81	1.75	1.75
4 Processors	25.94	2.54	2.55
8 Processors	21.25	3.10	3.12
16 Processors	21.79	3.03	3.04
32 Processors	28.42	2.32	2.33

Table 5: Application with 10001 grid points on IBM SP-2 using APR's xHPF compiler

Compiler	CPU Time (sec)	Speedup Relative to FORTRAN 77	Speedup Relative to HPF with 1 Processor
F77 HPF with FORALL	274.89	1.00	
1 Processor	260.03	1.06	1.00
2 Processors	140.17	1.96	1.86
4 Processors	87.17	3.15	2.98
8 Processors	63.13	4.35	4.12
16 Processors	54.80	5.02	4.75
32 Processors	62.74	4.38	4.14

Table 6: Application with 20001 grid points on IBM SP-2 using APR's xHPF compiler

PGI's HPF compiler produced better results than APR's xHPF compiler at higher processor numbers as can be seen on the tables below.

Compiler	CPU Time (sec)	Speedup Relative to FORTRAN 77	Speedup Relative to HPF with 1 Processor
F77 HPF with FORALL	66.25	1.00	
1 Processor	70.70	0.94	1.00
2 Processors	38.17	1.74	1.85
4 Processors	22.61	2.93	3.13
8 Processors	17.35	3.82	4.07
16 Processors	12.93	5.12	5.46
32 Processors	17.71	3.74	3.99

Table 7: Application with 10001 grid points on IBM SP-2 using PGI's HPF compiler

Compiler	CPU Time (sec)	Speedup Relative to FORTRAN 77	Speedup Relative to HPF with 1 Processor
F77 HPF with FORALL	274.89	1.00	
1 Processor	288.24	0.95	1.00
2 Processors	149.68	1.84	1.93
4 Processors	86.85	3.17	3.32
8 Processors	55.49	4.95	5.19
16 Processors	37.43	7.34	7.70
32 Processors	38.77	7.09	7.43

Table 8: Application with 20001 grid points on IBM SP-2 using PGI's HPF compiler

5.3 Intel Paragon

Two HPF compilers were used on the Intel Paragon. The first that will be discussed is the Applied Parallel Research (APR) HPF compiler. The timing results from this compiler showed a reasonable speedup up until large number of processors were used. When the number of processors were increased above 8, communication began to dominate and speedup went down. This is mainly because the outputted source code shows communication before each operation; unnecessary communication occurs because of the compiler. Note that since the Paragon runs much slower than the DEC Alpha Farm and the IBM SP-2 the code was only run for 1/6 the number of iterations. When the code is run for the same number of iterations as on the DEC Alpha Farm, the timings are six times as large with little variation and the speedup does not change.

Compiler	CPU Time (sec)	Speedup Relative to FORTRAN 77	Speedup Relative to HPF with 1 Processor
F77 HPF with FORALL	50.11	1.00	
1 Processor	54.99	0.91	1.00
2 Processors	31.17	1.61	1.76
4 Processors	20.05	2.50	2.74
8 Processors	15.27	3.28	3.60
16 Processors	15.75	3.18	3.49
32 Processors	21.36	2.35	2.57

Table 9: Application with 10001 grid points on Intel Paragon Using APR's HPF Compiler

The second HPF compiler used was from Portland Group Inc. (PGI), in which two version were used (v1.01 and v1.3). The timing results showed reasonably good speedups. The outputted source code does show similar unnecessary communication as from APR's compiler for version 1.1. However, communication is much better optimized (unnecessary communication has been removed) in the latest version of the compiler (v1.3).

Compiler	CPU Time (sec)	Speedup Relative to FORTRAN 77	Speedup Relative to HPF with 1 Processor
F77 HPF version 1.01	12.51	1.00	
1 Processor	30.29	0.41	1.00
2 Processors	16.03	0.78	1.89
4 Processors	8.68	1.44	3.49
8 Processors	5.01	2.50	6.05
16 Processors	3.37	3.71	8.99
32 Processors	3.04	4.12	9.96

Table 10: Application with 5001 grid points on Intel Paragon Using PGI's HPF Compiler

Compiler	CPU Time (sec)	Speedup Relative to FORTRAN 77	Speedup Relative to HPF with 1 Processor
F77 HPF version 1.01	50.10		
1 Processor	120.86	0.41	1.00
2 Processors	62.47	0.80	1.93
4 Processors	32.83	1.53	3.68
8 Processors	17.87	2.80	6.76
16 Processors	10.71	4.68	11.28
32 Processors	7.83	6.40	15.44
HPF version 1.3			
1 Processor	49.13	1.02	1.00
2 Processors	25.49	1.97	1.93
4 Processors	13.35	3.75	3.68
8 Processors	7.29	6.87	6.74
16 Processors	5.53	9.06	8.88
32 Processors	3.12	16.05	15.75

Table 11: Application with 10001 grid points on Intel Paragon Using PGI's HPF Compiler

Note that there is change in speedup for the 10001 grid point application from 8 to 16 processors. This is because the Paragon at PGI only has 15 processors available for use. The code was compiled for 16 and 32 processors there and transfered to the Paragon at NASA Ames. This transfer may explain the change in the speedup trend.

5.4 Optimization of Communication

It was difficult to determine how communication was being handled with Digital's and IBM's compilers because neither could output a source code to be examined. Although the code could be profiled, it was still difficult to determine how communication was being handled. APR's pre-compiler, xHPF, allowed the user to see where communication occurs in the outputted source code and gain a little more insight from profiling the code. APR's pre-compiler also allowed the user to change the outputted source code before compiling with mpxlf. PGI's compiler can output a source code that shows where and what is being communicated but does not allow for modification of the source code before compilation.
In both APR's and PGI's outputted source code, communication occurs in unnecessary places. The array **U** is used in the *flux* and *dissipation* routines and is only set after these routines are completed. Communication of **U** for nearest neighbor processors should occur before it is used in the *flux* routine. For both compilers communication occurs once for each FORALL statement that uses the array **U**. However, the new version of the PGI compiler (v1.3) does handle this problem, communicating the array **U** once before it is used and not again until after it has been reset.

6 Conclusions

The physical problem of wave propagation as well as the numerical used in its solution have been examined. The development of the sequential FORTRAN 77 and FORTRAN 90 codes have been discussed as well as the parallel HPF version. The method of timings has been examined and discussed. The results of the timings for Digital's and IBM's HPF show poor performance relative to the sequential FORTRAN 77 and FORTRAN 90 version. Possible explanations for HPF's poor performance have been suggested. HPF's performance relative to the case of HPF with 2 processors does show a semi-linear speedup, which is promising. Unfortunately, it appears that the use of Digital's and IBM's HPF for this type of application does not show an advantage over the sequential code with the present version of the compiler. It is most likely that DEC's poor results can be attributed to the higher cost of communication between workstations. IBM's poor results is most likely do to the fact that this is the first version of the HPF compiler. Since neither compiler allowed for an examination of the outputted source code it was difficult to determine how the compiler was handling communication. APR's and PGI's HPF compilers resulted in a more efficient code. Although there are still better way to optimize communication in APR's compiler, PGI's latest compiler seems to optimize the communication quite well. From the above discussion it appears that the optimization of communication is one of the major issues facing the use of HPF compilers.
Although the use of FORALL statements as a data parallel operation is fine for this particular CFD application, it would not be for more complex problems. The solution of the Euler and Navier Stokes equations involves the use of many 'scratch' values that are scalar. To convert these codes into HPF, these scalar values must be converted to arrays (scalar expansion) for the use of the FORALL. This increase in the number of arrays increases the amount of memory required by the program. If the INDEPENDENT was implemented, the difficulty involved in converting sequential FORTRAN 77 codes into HPF would be greatly decreased as well as removing the additional memory required in scalar expansion.

Acknowledgments

This work was done using the DEC Alpha Farm at the Northeast Parallel Architectures Center (NPAC) at Syracuse University, the Paragons at JPL, NASA Ames, and NASA Langley, and the IBM SP-2 at NASA Langley and NASA Ames. I would like to thank Dr. Piyush Mehrotra of ICASE (Institute for Computer Application in Science and Engineering) for helpful suggestions with HPF. I would also like to

thank Dr. Thong Dang of the MAME (Mechanical, Aerospace, and Manufacturing Engineering) department of Syracuse University for help with the numerical algorithms. Finally, I would like to thank Doug Miles of PGI (Portland Group Inc.) for running the code with the latest version of the PGI compiler on the Intel Paragon.
Kevin Roe is supported as a Research Assistant by the NASA Center for Hypersonics at Syracuse University. Work for this paper was done both at Syracuse University and at ICASE.

References

[1] LeVeque, R.J., **Numerical Methods for Conservative Laws**, Birkhãuser-Verlag, Berlin, 1992.

[2] Koelbel, C.H., Loveman, D.B., Schreiber, R.S., Steele, G.L., Zosel, M.E., **The High Performance FORTRAN Handbook**, The MIT Press, Cambridge, 1994.

[3] High Performance FORTRAN Forum 1993, High Performance FORTRAN Language Specification Version 1.0

[4] Loukides, M., **UNIX for FORTRAN Programmers**, O'Reilly & Associates, Inc., 1990.

[5] Dowd, Kevin, **High Performance Computing**, O'Reilly & Associates, Inc., 1993.