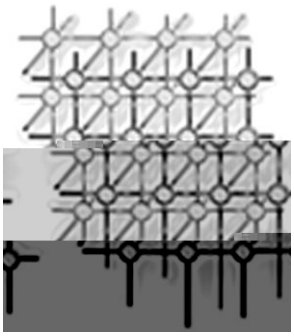
CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
Concurrency Computat.: Pract. Exper. 2003; **15**:607 621 (DOI: 10.1002/cpe.718)

A scalable HPF implementation of a finite-volume computational electromagnetics application on a CRAY T3E parallel system



Yi Pan[1,*], Joseph J. S. Shang[2] and Min i Guo[3]

[1]*Department of Computer Science, Georgia State University, Atlanta, GA 30303, U.S.A.*
[2]*Air Vehicle Directorate, Air Force Research Laboratory, WPAFB, OH 45433-7521, U.S.A.*
[3]*Department of Computer Software, The University of Aizu, Aizu-Wakamatsu City, Fukushima, 965-8580 Japan*

SUMMARY

The time-dependent Maxwell equations are one of the most important approaches to describing dynamic or wide-band frequency electromagnetic phenomena. A sequential finite-volume, characteristic-based procedure for solving the time-dependent, three-dimensional Maxwell equations has been successfully implemented in Fortran before. Due to its need for a large memory space and high demand on CPU time, it is impossible to test the code for a large array. Hence, it is essential to implement the code on a parallel computing system. In this paper, we discuss an efficient and scalable parallelization of the sequential Fortran time-dependent Maxwell equations solver using High Performance Fortran (HPF). The background to the project, the theory behind the efficiency being achieved, the parallelization methodologies employed and the experimental results obtained on the Cray T3E massively parallel computing system will be described in detail. Experimental runs show that the execution time is reduced drastically through parallel computing. The code is scalable up to 98 processors on the Cray T3E and has a performance similar to that of an MPI implementation. Based on the experimentation carried out in this research, we believe that a high-level parallel programming language such as HPF is a fast, viable and economical approach to parallelizing many existing sequential codes which exhibit a lot of parallelism. Copyright © 2003 John Wiley & Sons, Ltd.

KEY WORDS: parallel computers; e ecution time; ef cienc ; scalabilit ; loop paralleli ation; Cra T3E; high performance Fortran

*Correspondence to: Yi Pan, Department of Computer Science, Georgia State Uni ersit , Atlanta, GA 30303, U.S.A.
E-mail: pan@cs.gsu.edu
A preliminar ersion of this paper appeared in ICA3PP 2000, Hong Kong, December 2000.

Contract/grant sponsor: Air Force Of ce of Scienti c Research; contract/grant number: F49620-93-C-0063
Contract/grant sponsor: National Science Foundation; contract/grant number: CCR-9211621, OSR-9350540, CCR-9503882, ECS-0010047
Contract/grant sponsor: Air Force A ionics Laborator , Wright Laborator ; contract/grant number: F33615-C-2218
Contract/grant sponsor: Japan Societ for the Promotion of Science; contract/grant number: 14580386

Cop right © 2003 John Wile & Sons, Ltd.

Received 11 October 2001
Revised 8 August 2002

1. INTRODUCTION

Computational electromagnetics (CEM) in the time domain is the most general numerical approach for describing d namic or ide-band frequenc electromagnetic phenomena. Computational simulations are deri ed from discreti ed appro imations to the time-dependent Ma ell equations [1 10]. High numerical ef cienc of CEM simulation procedures can be attained either b algorithmic impro ements to sol e the Ma ell equations or b using scalable parallel distributed memor computer s stems. Since a massi e olume of data-processing is in ol ed in sol ing the Ma ell equations, distributed memor computer s stems are iable means to sol e the memor shortage problem on orkstations or ector computer s stems. The other ad antage is the reduced time hen parallel processing is emplo ed to sol e the Ma ell equations. Hence, paralleli ation of e isting sequential Fortran code for sol ing the Ma ell equations is an important effort to ards de eloping an ef cient and accurate CEM code to anal e refraction and diffraction phenomena for aircraft signature technolog .

Until no , three ersions of the sol er ha e been a ailable. One is the sequential Fortran ersion called MAX3D de eloped b the second author of this paper for ector machines such as the CRAY C90. The second is the MPI (Message-passing Interface) [11,12] ersion implemented b Dr Marcus Wagner of IBM [13]. The third one is the Po er Fortran Accelerator (PA) [14] ersion de eloped b the rst author hen he as an Air Force Of ce for Scienti c Research (AFOSR) Facult Research Fello at the Wright Patterson Air Force Base. The description of the programs and their performance results are summari ed in [15]. Using the message-passing paradigm has se eral disad antages: the cost of producing a message-passing code is usuall er high, the length of the code gro s signi cantl and it is much less readable and less maintainable than the sequential ersion. For these reasons, it is idel agreed that a higher le el programming paradigm is essential if parallel s stems are to be idel adopted. Pre ious results ha e sho n that the performance of the PA code on SGI Origin 2000 is e en better than the MPI counterpart besides the ad antages alread stated [15]. This suggests that a higher le el programming paradigm such as the HPF is a iable and alternati e a to paralleli e CEM codes.

Ho e er, the results achie ed so far are onl a preliminar step in achie ing a scalable ersion of the MAX3D code. The major problem ith the PA ersion is that it is not portable across different parallel computer s stems since Po er Fortran is onl a ailable on SGI Po er Challenger and Origin 2000.

The primar research objecti es of this research as to de elop a parallel code for the MAX3D using High Performance Fortran (HPF). HPF is more po erful than Po er Fortran and is a ailable in almost e er major parallel computer s stem. Our goals are to impro e the scalabilit of the Po er Fortran ersion of the MAX3D code further, in estigate the possibilit of using HPF to implement parallel CEM applications and make the parallel code portable across se eral parallel platforms. These objecti es ha e been accomplished through implementation of the parallel code for the MAX3D ia PGHPF [16] (an HPF compiler de eloped b the Portland Group). E perimental runs ha e sho n that the e ecution time is reduced drasticall through the use of HPF. The time spent for the implementation is about t o months. Compared ith MPI implementation, it is much shorter. Since HPF is a general parallel language and not targeted to an particular machine, its performance is usuall not as good as speciali ed parallel languages such as Po er Fortran on the SGI machines. Yet, the HPF code e ha e produced is still scalable up to 98 processors on the Cra T3E. Based on the e perimentation carried out in this research, e sho that HPF is a iable and alternati e a to

Cop right © 2003 John Wile & Sons, Ltd.

paralleli e man e isting sequential codes hich e hibit a lot of parallelism besides MPI. In man cases, using HPF to implement a sequential Fortran code is more economical than using MPI. HPF also sol es the problem of limited memor si e on a single processor and pro ides a fast a to port the code on parallel computer s stems.

In the follo ing sections, e ill describe the numerical formulation of the problem and the related subroutines in the sequential code. We then present the paralleli ation schemes used, the results achie ed during the research and outline the impact of the achie ed research results.

2. THE MAXWELL EQUATIONS

The beha ior of electromagnetic phenomena is modeled b the set of four equations collecti el kno n as Ma ell's equations. In differential ector form for a Cartesian reference frame, these equations are ritten as follo s (see [8,9] for more details).

Farada 's la :

$$\frac{\partial B}{\partial t} + \nabla \times E = 0 \tag{1}$$

Ampère's la :

$$\frac{\partial D}{\partial t} - \nabla \times H = -J \tag{2}$$

Gauss's la :

$$\nabla \cdot B = 0, \quad B = \mu H \tag{3}$$

$$\nabla \cdot D = 0, \quad D = \epsilon E \tag{4}$$

here ϵ and μ are the electric permitti it and the magnetic permeabilit hich relate the electric displacement to the electric eld intensit and magnetic u densit to the magnetic eld intensit , respecti el .

Using a coordinate transformation from the Cartesian coordinates, the time-dependent Ma ell equations can be cast in a general bod -conformal cur ilinear frame of reference. The resultant go erning equations in u - ector form acquire the follo ing form [17]:

$$\frac{\partial U}{\partial t} + \frac{\partial F_\xi}{\partial \xi} + \frac{\partial F_\eta}{\partial \eta} + \frac{\partial F_\zeta}{\partial \zeta} = -J \tag{5}$$

here U is the transformed dependent ariable no scaled b the local cell olume, V. F_ξ, F_η and F_ζ are the contra ariant components of the u ectors of the Cartesian coordinates.

$$U = \{B_x V, B_y V, B_z V, D_x V, D_y V, D_z V\}^{\mathrm{T}} \tag{6}$$

$$\begin{aligned} F_\xi &= \xi_x F_x + \xi_y F_y + \xi_z F_z \\ F_\eta &= \eta_x F_x + \eta_y F_y + \eta_z F_z \\ F_\zeta &= \zeta_x F_x + \zeta_y F_y + \zeta_z F_z \end{aligned} \tag{7}$$

Cop right © 2003 John Wile & Sons, Ltd.

here the u - ector components of the Cartesian frame are:

$$
\begin{aligned}
F_x &= \{0, -D_z/\epsilon, D_y/\epsilon, 0, B_z/\mu, -B_y/\mu\}^{\mathrm{T}} \\
F_y &= \{D_z/\epsilon, 0, -D_x/\epsilon, -B_z/\mu, 0, B_x/\mu\}^{\mathrm{T}} \\
F_z &= \{-D_y/\epsilon, D_x/\epsilon, 0, B_y/\mu, -B_x/\mu, 0\}^{\mathrm{T}}
\end{aligned}
\tag{8}
$$

and $\xi_x, \eta_x, \zeta_x, \xi_y, \eta_y, \zeta_y, \xi_z, \eta_z$, as ell as ζ_z are the nine metrics of the coordinate transformation.

The characteristic-based nite- olume appro imation is achie ed b splitting the u ector according to the signs of the eigen alues of the coef cient matri in each spatial direction. The u ector at an cell interface is represented b the superposition of t o components: F_ξ^+, F_ξ^-, F_η^+, F_η^-, F_ζ^+ and F_ζ^- according to the direction of the a e motion [17]. At the cell surfaces, the split u ectors are calculated b the reconstructed dependent ariables on either side of the interface according to the Kappa scheme [18]:

$$
\begin{aligned}
F_{\xi,i+\frac{1}{2}} &= F_\xi^+(U^{\mathrm{L}}_{i+\frac{1}{2}}) + F_\xi^-(U^{\mathrm{R}}_{i+\frac{1}{2}}) \\
F_{\eta,j+\frac{1}{2}} &= F_\eta^+(U^{\mathrm{L}}_{j+\frac{1}{2}}) + F_\eta^-(U^{\mathrm{R}}_{j+\frac{1}{2}}) \\
F_{\zeta,k+\frac{1}{2}} &= F_\zeta^+(U^{\mathrm{L}}_{k+\frac{1}{2}}) + F_\zeta^-(U^{\mathrm{R}}_{k+\frac{1}{2}})
\end{aligned}
\tag{9}
$$

here U^{L} and U^{R} denote the reconstructed dependent ariables at the left- and right-hand side of the cell interface.

A single-step, t o-stage Runge Kutta scheme is adopted for the temporal integration process: the resultant numerical procedure is capable of generating numerical solutions accurate in space from rst to third order and accurate in time to second order.

The initial and boundar conditions for a perfectl electrical conducting (PEC) sphere can be summari ed as follo s. The incident a e consists of a linearl polari ed harmonic eld propagating in the negati e z-a is direction [9,10]. The far- eld boundar condition for the truncated numerical domain is easil implemented b using the split u ectors from the characteristic formulation.

$$
F_\xi^-(\xi_0, \eta, \zeta) = 0 \tag{10}
$$

The appropriate boundar conditions at the surfaces of a PEC scatterer are [6]:

$$
\begin{aligned}
\vec{n} \times (E_1 - E_2) &= 0 \\
\vec{n} \times (H_1 - H_2) &= J_s \\
\vec{n} \cdot (B_1 - B_2) &= 0 \\
\vec{n} \cdot (D_1 - D_2) &= \rho_s
\end{aligned}
\tag{11}
$$

here the subscripts 1 and 2 denote elds in t o adjacent regions. Follo ing pre ious efforts, t o e trapolated numerical boundar conditions are introduced to replace the equations that contain unkno ns, J_s and ρ_s. The detailed formulation can be found in [6].

The code MAX3D contains man big loops. The main bod of the code hich is the part for sol ing sequences and Ma ell's equations is sho n here. Since NEND in the code is usuall er large (in our test case, it equalled 1562), loop 10 takes most of the time.

Cop right © 2003 John Wile & Sons, Ltd.

```
      .....
C     START THE SOLVING SEQUENCES
      DO 10 N=1,NEND
         T=FLOAT(N-1)*DT
         CALL IC(N,NEND)
         DO 12 M=1,2
            CALL FXI
            CALL GETA
            CALL HZETA
            CALL SUM(M,N,NEND)
 12      CONTINUE
         IF((N.GT.NES).OR.(N.LT.NBS)) GO TO 10
C     PERFORM FOURIER'S TRANSFORM
         WT=W*(T+DT)
         DO 14 L=1,6
         DO 14 K=1,KL
         DO 14 J=1,JL
            FU(J,K,L)=FU(J,K,L)+U1(IB,J,K,L)*CEXP(RI*WT)
 14      CONTINUE
 10   CONTINUE
      .....
```

After testing and performance pro ling of the sequential code MAX3D, e found that the subroutines FXI, GETA, HZETA and SUM occup the most time for the sequential code and deser e special attention. Subroutines FXI, GETA, and HZETA generate electromagnetic coef cients for the u ectors and SUM computes the ne time le el solution. These subroutines are related to Equations (6) (8). In our stud , e concentrate on the most time-consuming subroutines such as FXI, GETA, HZETA and SUM, and tr to paralleli e the loops ithin these subroutines ef cientl .

3. METHODOLOGIES

High Performance Fortran (HPF) is an informal standard for e tensions to Fortran to assist its implementation on parallel architectures, particularl for data-parallel computation [19]. Among other things, it includes directi es for e pressing data distribution across multiple memories, e tra facilities for e pressing data parallel and concurrent e ecution and a mechanism for interfacing HPF ith other languages and programming models. HPF is a ailable on almost all popular parallel computer s stems, such as Cra T3E, SP2 and SGI Origin 2000, and is much easier to use and faster to port than MPI. HPF allo s HPF directi es to be added to the sequential Fortran code and to appl the capabilities of a multiprocessor s stem to the e ecution of a single job. It splits the job into concurrentl e ecuting pieces, thereb decreasing the all-clock runtime of the job. Although HPF uses a lot of complicated anal sis tools to anal e the user program and can produce a corresponding parallel ersion ithout user inter ention, the qualit of the parallel code produced in such a a is usuall er poor. In our stud , e found that using the PGHPF compiler's option `-Mautopar` to paralleli e the MAX3D code automaticall does not gi e effecti e parallel results. The compiler is simpl not intelligent enough to make a smart decision. Another problem ith automatic paralleli ation is that it can create a large number of temporar arra s. Hence, automatic paralleli ation has to be combined ith hand

tuning to produce more ef cient code. For e ample, hen e paralleli e the MAX3D code using automatic paralleli ation on SGI Origin 2000, the parallel time is orse than the sequential time due to data dependence and the hea communication o erhead in the code. In our research, onl hand paralleli ation (inserting directi es manuall ersus automatic code generation b soft are) is used to a oid memor aste and increase ef cienc .

The HPF model can be seen as a collection of distinguishable, but complementar , programming st les. The models and methods e used in the implementation are data- and ork-sharing. In data-sharing, data, such as an arra , are spreaded o er the memor of all of the a ailable processors so that each processor operates primaril on its o n part. In ork-sharing, the loop iterations are distributed ithin loops among the s stem's processors ith the goal of e ecuting them in parallel. For instance, the INDEPENDENT directi e can be used to di ide the iterations of a DO loop among processors or the compiler can di ide the ork b choosing an implicit arra s nta . One natural and po erful strateg in ol es distributing an arra , data-sharing and the iterations of a DO loop that operate on that arra , ork-sharing, o er our a ailable processors. Processors e ecuting a DO loop in parallel to help reali e the po er of the CRAY T3E and other parallel computing s stems. In the follo ing sections, the details of data allocation and paralleli ation schemes are discussed.

In order for a parallel code to run ef cientl , it is also essential that the sequential code uses the cache effecti el . The o erall optimi ation steps adopted in our research are:

(1) locate the 'hot spot' of the sequential code ia pro ling tools;
(2) stud these subroutines identi ed in step 1 carefull for performance impro ement;
(3) impro e the cache localit beha ior of these subroutines through a series of s stematic loop transformations;
(4) stud the listing le produced b the HPF compiler and locate the o erhead caused b inef cient parallel loops; and
(5) insert HPF compiler directi es to direct the compiler to paralleli e the code so that most of the o erhead is eliminated and the most time-consuming loops are paralleli ed.

Note that some of the goals ma be in con ict. Hence, these steps ma need to be repeated se eral times to achie e an o erall good performance.

3.1. Distributed data allocation

In a parallel program on a distributed memor architecture, data can be shared (distributed) or pri ate (replicated). Arra s can also be distributed to speedup the code. Arra s are distributed across all processors ia the `!HPF$` DISTRIBUTE directi e in HPF. The DISTRIBUTE directi e names the ariables that are to be shared data objects and speci es ho the are to be distributed across the PEs. If the data are not speci ed as shared, the are pri ate (the default). In a program that declares a pri ate data item, each processor gets a cop of storage for that item. In man programming models, this datum is called replicated. When an arra is shared, its elements are distributed across the a ailable PEs.

The DISTRIBUTE directi e can specif arra distribution ithin the directi e b follo ing the arra name ith distribution information contained in parentheses. The distribution information ma include the ke ords BLOCK or CYCLIC, hich tell the compiler ho to distribute the arra elements among the a ailable PEs. The BLOCK distribution distributes a block of consecuti e memor locations to a processor, hile the CYCLIC distribution distributes an arra in a c clic manner.

Cop right © 2003 John Wile & Sons, Ltd.

The DISTRIBUTE directi e allo s data to be distributed o er processors in a ariet of patterns. The ALIGN directi e is used to specif that certain data objects are to be mapped in the same a as certain other data objects in order to reduce inter-processor communication. Operations bet een aligned data objects are likel to be more ef cient than operations bet een data objects that are not kno n to be aligned. In our MAX3D implementation, U0, U1, U2, F and G are all aligned and ha e the same distribution, as sho n later. Hence, e can paralleli e the code across dimension K ef cientl hen orking on these large arra s. The ONTO clause speci es the processor arrangement declared in a PROCESSORS directi e. In our case, the arrangement is PROCS.

```
!HPF$ PROCESSORS PROCS(NUMBER_OF_PROCESSORS())
!HPF$ ALIGN U1(I,J,K,L) WITH U0(I,J,K,L)
!HPF$ ALIGN U2(I,J,K,L) WITH U0(I,J,K,L)
!HPF$ ALIGN F(I,J,K,L) WITH U0(I,J,K,L)
!HPF$ ALIGN G(I,J,K,L) WITH U0(I,J,K,L)
!HPF$ DISTRIBUTE U0(*,*,BLOCK,*) ONTO PROCS
```

Similarl , since e need to paralleli e HZETA on dimension J, the H arra is distributed on dimension J as follo s:

```
!HPF$ DISTRIBUTE H(*,BLOCK,*,*) ONTO PROCS
```

Since different phases require different data distributions, it ould be nice to d namicall distribute an arra during e ecution. HPF pro ides such a mechanism the DYNAMIC and REDISTRIBUTE directi es. Although arious redistribution methods ha e been studied and a lot of impro ements ha e been made, the redistribution of large arra s still takes a huge amount of time in general [20]. In our research, e ha e also tried to use these t o directi es to distribute arra s d namicall . Since redistribution of large arra s such as U0 and H takes a lot of time, the o erhead incurred is larger than the sa ing in time. Hence, e did not use this redistribution scheme in our nal implementation. Another distribution method is CYCLIC. This distribution distributes an arra of elements c clicall to different processors. We found this method as not ef cient for the MAX3D code.

3.2. Loop parallelization

The major part of the paralleli ation process for the MAX3D code is loop paralleli ation since the codes contain se eral big loops. The parallelism model used focuses on the Fortran DO loop. The compiler e ecutes different iterations of the DO loop in parallel on multiple processors.

The most e ible and ef cient data partition of a computational domain ould be the three-dimensional domain decomposition in each coordinate of the transformed space (ξ, η, ζ) since this ill cause the least amount of data communication. The global domain has IL, JL and KL cells in the ξ, η and ζ coordinates respecti el . After careful stud , ho e er, e found that it is hard to paralleli e all three dimensions at the same time due to data dependenc . Hence, e decided to paralleli e one loop at a time and different loops are paralleli ed in different subroutines (hiche er arra dimension leads to least data dependencies). E periments indicate that the parallel code using this approach runs quite ef cientl as sho n later.

In HPF, e ma use a FORALL statement to paralleli e a nested loop [21]. Ho e er, a FORALL statement automaticall paralleli es all the loops ithin a nested loop and users ha e no control o er the a in hich loops are paralleli ed [22]. Hence, users cannot specif hich loop to paralleli e ithin a nested loop. Due to data dependence, this sometimes results in poor performance. In some cases, a DO loop cannot be simpl transformed to a FORALL statement, because there are different semantics bet een DO and FORALL statements [19]. The other compiler directi e for multiprocessing in HPF is `!HPF$ INDEPENDENT`. This directi e directs the compiler to generate a special code to run iterations of a DO loop in parallel. The loop is also called a shared loop. In a shared loop, the iterations are di ided among the a ailable tasks. A shared loop can be speci ed b the INDEPENDENT directi e. A pri ate loop, b contrast, is e ecuted onl b the task that in okes it; no ork is shared among tasks. Pri ate loops are not preceded b an INDEPENDENT directi e. For e ample, the follo ing directi e speci es that loop K in subroutine FXI should be paralleli ed and I, IM, etc., are pri ate ariables.

```
!HPF$ INDEPENDENT, NEW(I, IM, IP, J, SSXI, RSSXI, ....)
      DO 1 K=1,KLM
      DO 1 J=1,JLM
      DO 2 I=1,ILM
C     Initial value set up
  2   CONTINUE
      DO 3 I=2,ILM
      IM=I-1
      IP=I+1
C     RECONSTRUCT THE DATA AT THE CELL INTERFACE, KAPA
      UP1(I)=U1(I,J,K,1)+0.25*RP*((1.0-RK)*(U1(I,J,K,1)-U1(IM,J,K,1))
     1                          +(1.0+RK)*(U1(IP,J,K,1)-U1(I,J,K,1)))
      ......
```

Shared loops specif the beha ior of all tasks collecti el but the onl implicitl de ne the beha ior of indi idual tasks. Shared loops do not guarantee the order in hich iterations ill be e ecuted. The lack of a de ned order allo s the s stem to e ecute iterations concurrentl .

Inside a shared loop, each task e ecutes its assigned loop iterations as if each task ere its o n serial region. The assignment of loop iterations is accomplished b aligning the iterations according to the distribution of the arra named in the ON clause of the INDEPENDENT directi e.

There is an implicit barrier s nchroni ation at the end of a shared loop, at hich all tasks ait until the last one has completed. The shared loop ends after the DO loop that immediatel follo s the INDEPENDENT directi e. After the shared loop has nished e ecuting, the tasks continue to e ecute in parallel.

For multiprocessing to ork correctl , the iterations of the loop must not depend on each other; each iteration must stand alone and produce the same ans er regardless of hen an other iteration of the

This is a scanned copy of the original tunt(f)-01(asp7)ar 368-5.2annotns356ehe

Some data dependencies cannot be eliminated due to the nature of the computations in ol ed in the code. Our strateg is to paralleli e the loops hich ha e no data dependencies. For e ample, loop K in subroutine HZETA has data dependencies. Ho e er, loop J does not. Hence, e paralleli ed loop J instead of loop K in subroutine HZETA. The code segment of the subroutine HZETA is as follo s. The data dependencies e ist in the statements `S1`, `S2`, and `S3` for loop K:

```
      SUBROUTINE HZETA
      PARAMETER(IL=73,JL=61,KL=97)
      ......
!HPF$ INDEPENDENT,  NEW(K, KM, KP, I, SSZT, RSSZT, ....)
      DO 2 J=1,JLM
      DO 2 K=1,KLM
      KM=K-1
      KP=K+1
      DO 2 I=1,ILM
C S1
      UP1(I,K)=U1(I,J,K,1)+0.25*RP*((1.0-RK)*(U1(I,J,K,1)-U1(I,J,KM,1))
     1                            +(1.0+RK)*(U1(I,J,KP,1)-U1(I,J,K,1)))
      ......
C S2
      UC1=UI1(I,KM)*UP1(I,K)+UI2(I,KM)*UP2(I,K)+UI3(I,KM)*UP3(I,K)
      ......
C S3
      H(I,J,K-1,1)=SSZT*(HP(I,K-1,1)+HM(I,K,1))
      ......
 2    CONTINUE
      ......
      RETURN
      END
```

Similarl , e paralleli ed loop K in subroutines FXI and GETA. This is also re ected in their data distributions: arra F, G and H are distributed on different dimensions as sho n pre iousl .

4. PERFORMANCE RESULTS

During this research, e used a CRAY T3E computer s stem at the Ohio Supercomputer Center, hich is a po erful scalable parallel s stem ith 128 processing elements. Its peak performance can reach 76.8 GFLOPS. Each processor is a DECchip 64-bit super-scalar RISC processor. It has four- a instruction issue ith t o oating-point operations per clock. Each processor has on-chip 8 kb te direct-mapped L1 instruction cache and an on-chip 8 kb te direct-mapped L1 data cache. It also has an on-chip 96 kb te three- a -set-associati e L2 uni ed cache. Each processor has a local memor of 16 M ords (or 128 Mb te). The clock speed of the processor is 300 MH and its peak performance is 600 MFLOPS. Although the speed is quite fast compared ith other parallel s stems such as the SGI Origin 2000, the cache and local memor si es of the T3E are much smaller (each processor in the SGI Origin 2000 contains a 4 Mb te secondar cache). This limits the po er of the T3E for programs using a lot of memor space such as the MAX3D code. Actuall , the T3E's smaller cache si e effects sequential programs as ell as parallel programs.

Cop right © 2003 John Wile & Sons, Ltd.

Table I. E ecution times of the major subroutines.

No. of processors	FXI	GETA	HZETA	SUM
4	0.592 36	0.895 35	0.8809	2.8894
8	0.314 00	0.456 46	0.443 89	1.4939
16	0.169 63	0.252 90	0.226 04	0.836 92
32	0.099 733	0.150 42	0.118 62	0.507 16
48	0.074 417	0.108 90	0.115 01	0.376 80
64	0.053 583	0.076 501	0.064 894	0.256 80
90	0.053 734	0.076 40	0.064 764	0.255 11
98	0.029 48	0.041 19	0.064 06	0.1443
128	0.029 48	0.041 08	0.063 84	0.1430

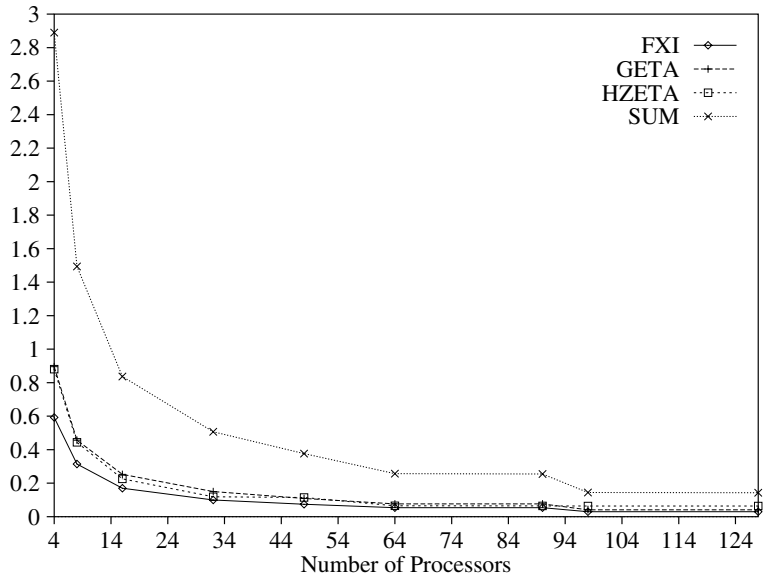


Figure 1. E ecution times of the major subroutines.

Se eral e periments ere carried out to tune our parallel code and to adjust our strateg as to ho to distribute the arious arra s and hich loops and subroutines to paralleli e. The HPF compiler used is PGHPF [16] ith optimi ation le el 3. In this section, some e perimental results are reported for the nal parallel code produced. All the times in the follo ing presentation are in seconds.

Table I lists the the timing information ith hand paralleli ation for the major subroutines in the code ith arra si e ($73 \times 61 \times 97$). The corresponding times are also presented in Figure 1. Since these subroutines are repeated man times, the time spent there contributes hea il to the total time of the

the parallel program could be run on a T3E ith one or t o processors. Hence, Table II onl sho s the e ecution times using at least four processors. Due to the requirement of the HPF compiler e used, man huge arra s had to be initiali ed before the ere used and, hence, a lot of o erhead is introduced. Therefore, the e ecution time on the T3E ith four processors is longer than that for the Origin 2000 ith four processors. Ho e er, the HPF code on the T3E is more scalable than the PA code on the Origin 2000 [23]. From Figure 2, e can also see that the e ecution time reaches its minimum hen 98 processors are used. Clearl , the parallel code not onl reduces the total e ecution time drasticall but it is also scalable up to 98 processors on the T3E s stem.

We can also see that the time is not reduced an further hen e increase the number of processors from 64 to 71 or 90. This is natural since e paralleli e the code on dimension K hich has a si e of 97; 71 or 90 processors are not enough to partition the arra on dimension K. Due to arra padding, the actual si e on dimension K is 98 and hence has a perfect partition o er 98 processors. Further increase in the number of processors ill not be effecti e unless e can distribute o er more than one arra dimension.

Because the timing information is not a ailable for the T3E using onl one processor (due to memor space limitation, e cannot run the code on one processor), e cannot calculate the speedup e actl . In order to estimate the speedup, e assume that the program is e ecuted optimall on the T3E ith four processors. This assumption is quite reasonable since e periments indicate that the MAX3D code runs ef cientl ith a small number of processors. With this assumption, e can estimate the e ecution time on the T3E ith one processor b multipl ing the e ecution time on the T3E ith four processors b four. Table III sho s the speedups of the HPF code after loop transformations and hand paralleli ation using this assumption. The corresponding data are also sho n in Figure 3. It is clear that the code is still quite scalable hen the number of processors used is 98.

For comparison purposes, e also include the speedups of the same code running on an IBM SP2 machine using MPI in Figure 3. The MPI code uses a three-dimensional decomposition scheme [13] and hence the amount of data e change is smaller than ith the HPF code hich basicall uses one-dimensional decomposition (paralleli ing one loop onl). Because of the different architectures and memor si es, it is hard to compare the e ecution times of the T3E and IBM SP2. Ho e er, e can still use the speedup performance to compare the scalabilit of the t o implementations. It is clear that the speedups are similar hen the number of processors used is less than 64. As the number of processors increases, the MPI code sho s better scalabilit than the HPF code. This phenomenon can be e plained easil . Here, e are distributing onl one arra dimension of si e 98. Therefore, for an number of processors bet een 49 and 97, one or more processors stores t o elements in the distributed dimension, resulting in load imbalancing. In contrast, the MPI code uses a three-dimensional decomposition, so the number of elements per processor, and hence the computational load per processor, can ar more smoothl ith the number of processors (it is better 'load balanced'), gi ing a smoother speedup for MPI.

The e periments conducted in this research are limited to a ed problem si e ($73 \times 61 \times 97$). When the problem si e changes, the scalabilit and ef cienc ma also change. Theoreticall , hen the problem si e is bigger, the ratio of computation o er communication also increases. This implies that the communication o erhead becomes relati el smaller. It is our conjecture that the parallel code ill be more scalable and ef cient hen computing larger problems. Ho e er, more e periments are needed to con rm our conjecture.

Cop right © 2003 John Wile & Sons, Ltd.

Table III. Speedups for problem si e (73 × 61 × 97).

No. of processors	Speedup
1	1
2	2
4	4
8	7.49
16	13.46
32	22.38
64	39.38
71	38.88
90	38.70
98	57.96
128	56.90

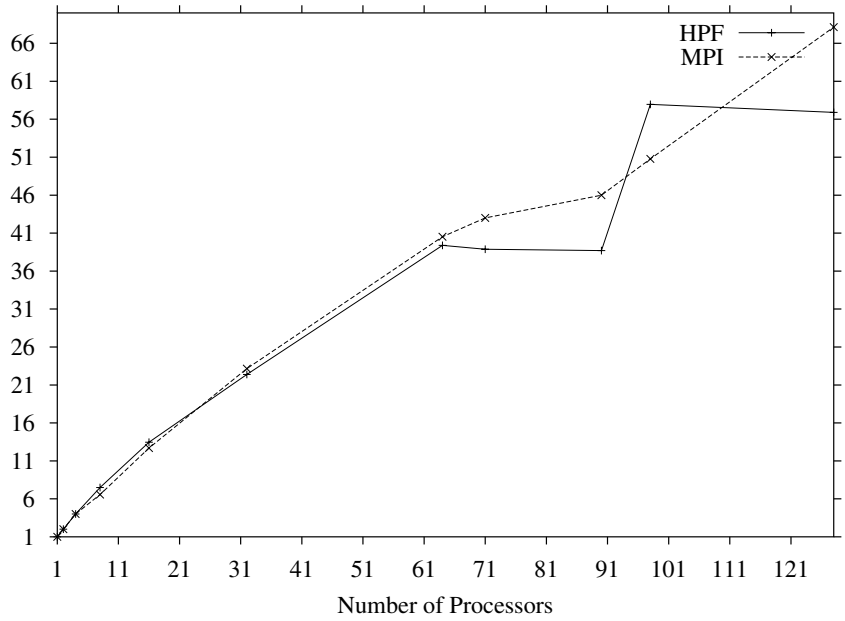


Figure 3. Comparison of speedups using HPF on T3E and MPI on SP2.

Cop right © 2003 John Wile & Sons, Ltd.

5. CONCLUSIONS

As e all kno , a lo -le el message-passing programming language such as MPI has se eral disad antages compared ith a high-le el parallel programming language: the cost of producing a message-passing code is usuall much higher, the length of the code gro s signi cantl and it is much less readable and maintainable than the one produced using a high-le el parallel programming language such as HPF. For these reasons, it is idel agreed that a higher le el programming paradigm is essential if parallel s stems are to be idel adopted. HPF has reached a critical stage in its histor . Ha ing struggled hile the compiler technolog e ol ed into a usable state, the parallel computing communit has no found it possible to rite portable, high-performance implementations for selected applications in HPF. HPF is becoming a standard for high-le el parallel programming and is a ailable on almost e er major parallel computer s stem. This research indicates that large-scale data-parallel applications such as CEM simulations can use HPF to achie e a reasonable performance.

This research addresses the portabilit and scalabilit problems of the MAX3D code through using HPF. Our results achie ed during the research ha e demonstrated that the sequential MAX3D code can be paralleli ed quickl b simple directi es insertion and et the code produced this a can still be e ecuted ef cientl . Furthermore, the code is still quite scalable. While the MPI code is more scalable than the HPF code due to different data distribution schemes (3D ersus 1D) hen using a larger number of processors, the labor cost of producing the MPI code is much larger than the HPF code. As ne ersions of commercial HPF compilers emerge in the market, and the HPF technolog becomes more mature, e e pect that the performance of the HPF MAX3D code ill be impro ed substantiall in the near future ith little change in the code. We belie e that high-le el parallel programming languages such as HPF is the future for fast implementation of parallel codes.

ACKNOWLEDGEMENTS

This research as supported b the Air Force Of ce of Scienti c Research under grant F49620-93-C-0063. Additional support has been pro ided b the National Science Foundation under grants CCR-9211621, OSR-9350540, CCR-9503882, and ECS-0010047, the Air Force A ionics Laborator , Wright Laborator , under grant F33615-C-2218, and the Japan Societ for the Promotion of Science Grant-in-Aid Basic Research (C) No. 14580386. We ould also like to thank Mark Young of Portland Group for his help in using PGI's High Performance Fortran compiler pghpf. Computational resources for the ork presented here ere pro ided b the Ohio Supercomputing Center, Columbus, Ohio. Man thanks also go to the three re ie ers and Professor Mark Baker for their constructi e comments and suggestions hich impro ed our paper greatl .

REFERENCES

1. Shang JS. A fractional-step method for sol ing 3-D time-domain Ma ell equations. *31th Aerospace Sciences Meeting and Exhibit*. American Institute of Aeronautics and Astronautics: Ne York, 1993.
2. Shang JS. Characteristics based methods for the time-domain Ma ell equations. *29th Aerospace Sciences Meeting*. American Institute of Aeronautics and Astronautics: Ne York, 1991.
3. Shang JS, Calahan DA, Vikstrom B. Performance of a nite olume CEM code on multicomputers. *Computing Systems in Engineering* 1995; **6**(3):241 250.
4. Shang JS, Fithen RM. A comparati e stud of characteristic-based algorithms for the Ma ell equations. *Journal of Computational Physics* 1996; **125**:378 394.
5. Shang JS, Gaitonde D. Scattered electromagnetic eld of a re-entr ehicle. *Journal of Spacecraft and Rockets* 1995; **32**(2):294 301.

6. Shang JS, Gaitonde D. Characteristic-based, time-dependent Ma ell equation sol ers on a general cur ilinear frame. *American Institute of Aeronautics and Astronautics Journal* 1995; **33**(3):491 498.
7. Shang JS, Gaitonde D, Wurt ler K. Scattering simulations of computational electromagnetics. *27th AIAA Plasmadynamics and Lasers Conference*. American Institute of Aeronautics and Astronautics: Ne York, 1996.
8. Shang JS, Gaitonde D. High-order nite- olume schemes in a e propagation phenomena. *27th AIAA Plasmadynamics and Lasers Conference*. American Institute of Aeronautics and Astronautics: Ne York, 1996.
9. Shang JS, Gaitonde D. On high resolution schemes for time-dependent Ma ell equations. *34th Aerospace Sciences Meeting and Exhibit*. American Institute of Aeronautics and Astronautics: Ne York, 1996.
10. Shang JS, Scherr SJ. Time-domain electromagnetic scattering simulations on multicomputers. *26th AIAA Plasmadynamics and Lasers Conference*. American Institute of Aeronautics and Astronautics: Ne York, 1995.
11. Snir M *et al.*. *MPI: The Complete Reference*. MIT Press: Cambridge, MA, 1996.
12. Gropp W, Lusk E, Skjellum A. *Using MPI: Portable Parallel Programming with the Message-passing Interface*. MIT Press: Cambridge, MA, 1994.
13. Shang JS, Wagner M, Pan Y, Blake DC. Strategies for adopting FVTD on multicomputers. *IEEE Computing in Science and Engineering (formerly known as IEEE Computational Science and Engineering)* 2000; **2**(1):10 21.
14. Hogue C. *MIPSpro (TM) Power Fortran 77 Programmer's Guide*. Silicon Graphics. Inc., 1996.
15. Pan Y. Impro ement of cache utili ation and parallel ef cienc of a time-dependent Ma ell equation sol er on the SGI Origin 2000. *Final Report for AFOSR Summer Faculty Research Program, Air Force Office of Scientific Research*, Bolling Air Force Base, DC, August 1997.
16. Miles D, Schuster V, Young M. The PGHPF high performance Fortran compiler: Status and future directions. *2nd Annual HPF User Group Meeting*. Springer: Berlin, 1998; 25 26.
17. Shang JS, Gaitonde D. Characteristic-based, time-dependent Ma ell equations sol ers on a general cur ilinear frame. *American Institute of Aeronautics and Astronautics Journal* 1995; **33**(3):491 498.
18. Anderson WK, Thomas JL, Van Leer B. A comparison of nite olume u splittings for the Euler equations. *American Institute of Aeronautics and Astronautics Journal* 1986; **24**(9):1453 1460.
19. Koelbel CH. *The High Performance Fortran Handbook*. MIT Press: Cambridge, MA, 1994.
20. Guo M, Yamashita Y, Nakata I. An ef cient data distribution technique for HPF compliers on distributed memor parallel computers. *Transactions of Information Processing Society of Japan* 1998; **39**(6):1718 1728.
21. Wolfe M. *High Performance Compilers for Parallel Computing*. Addison-Wesle : Reading, MA, 1996.
22. Guo M. A denotational semantic model of an HPF-like language. *Proceedings of The First International Conference on Parallel and Distributed Computing, Applications and Technologies*, Uni ersit of Hong Kong, Ma 2000; 1 8.
23. Pan Y. Parallel implementation of computational electromagnetics simulation using high performance Fortran. *Final Report for AFOSR Summer Research Extension Program*, Air Force Of ce of Scienti c Research, Bolling Air Force Base, DC, No ember 1998.

Cop right © 2003 John Wile & Sons, Ltd.