# Seismic computations on the IBM 3090 Vector Multiprocessor

by A. Kamel
M. Kindelan
P. Sguazzero

***Computerized seismic prospecting is an echo-ranging technique usually targeted at accurate mapping of oil and gas reservoirs. In seismic surveys an impulsive source, often an explosive charge, located at the earth's surface generates elastic waves which propagate in the subsurface; these waves are scattered by the earth's geological discontinuities back to the surface, where an array of receivers registers the reflected signals. The data recorded are then processed in a complex sequence of steps. Among them, seismic migration and stacking velocity estimation represent two characteristic components of the process solving the inverse problem of recovering the structure and the physical parameters of the earth's geologic layers from echo measurements. A complementary tool in relating seismic data to the earth's inhomogeneities is provided by seismic numerical models, which assume a subsurface structure and compute the seismic data which would be collected in a field survey, by solving the direct problem of exploration geophysics. This paper describes a vectorized and parallelized implementation of a two-dimensional seismic elastic model on the IBM 3090 VF Vector Multiprocessor. An implementation of a parallel seismic migration algorithm is then described. The paper also reports performance data for a vector/parallel implementation on the IBM 3090 of some typical seismic velocity estimation algorithms. The three problems chosen are representative of a wide class of geophysical computations, and the results summarized in this paper show their suitability for efficient implementation on the IBM 3090 Vector Multiprocessor; combined vector/parallel speedups in the range 15–25 are in fact observed.***

Seismic prospecting for hydrocarbon detection seeks to determine the geologic structure of the earth from indirect measurements obtained at the earth's surface. In seismic prospecting, elastic wave fields (acoustic as a first approximation) are generated in a controlled fashion at the surface, penetrate the earth, and are backscattered by the earth's inhomogeneities to an array of receivers, where they are recorded as shown in the model of Figure 1A. The structure consists of four layers, with propagation velocities $V$ equal to 4, 3, 5, and 6 km/s, respectively. On the horizontal axis (at the earth's surface), the source–receiver distance is measured in km; on the vertical axis (pointing down into the earth), the depth is measured in km from the surface. The seismic interpreters correlate what is seen in the seismic data (Figure 1B) with the earth's structure through the use of a complex sequence of data transformations, among which the processes of seismic migration, seismic velocity estimation, and seismic forward modeling play a major role. The three hyperbolic curves are the echoes of the three layer interfaces of Figure 1A. On the horizontal axis (at the earth's surface), the source–receiver distance is measured in km; on the vertical axis, the arrival time of the echoes is measured in seconds.

The purpose of *seismic migration*[1] is to reconstruct the reflectivity map of the earth from the seismic data recorded at the surface. The seismic signal recorded by the receivers is a superposition of up-

Figure 1 (A) Computerized seismic prospection over a multilayer

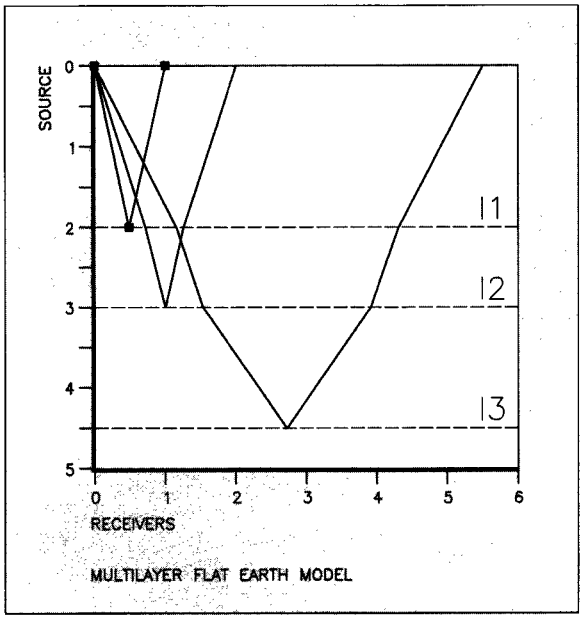
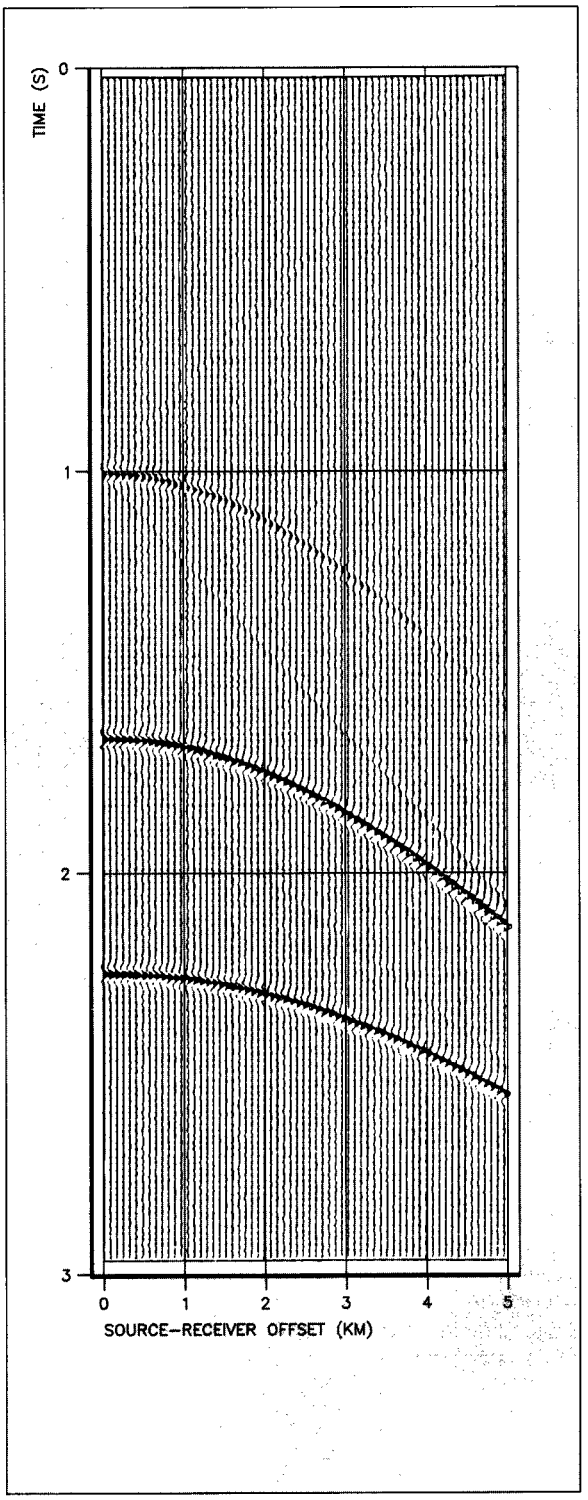



Figure 1 (B) Seismic gather corresponding to the multilayer of Figure 1(A)




ward-directed seismic waves originating from all of the discontinuity surfaces of the subterrain. In the migration process these recorded waves are used either as initial conditions or as boundary conditions for a wave field governed by the wave equation. As a result, these waves are propagated *backward* and in *reverse time* from the surface to the reflector locations, thus allowing the accurate mapping of subsurface geological "anomalies" such as oil and gas reservoirs.

The time series associated with a single shot and receiver is known as a trace. Seismic processing techniques have been developed for groups of traces, known as *gathers*. One such processing step preceding migration is termed *stacking*.[2] It consists of the summation of the traces sharing the same source–receiver midpoint after correcting them to compensate for the offset between source and receiver. The stacking process is based upon the accurate estimation of the average propagation velocities of seismic waves in the subsurface as a function of depth—the *stacking velocities*. Seismic migration and seismic velocity estimation represent two essential steps in the *inverse process* of recovering the structure and the physical parameters of the earth's subsurface from measurements obtained at the surface.

Additional insight is often obtained by *forward-modeling* several hypothesized geologic structures with the elastic or acoustic wave equation; in this way, field surveys can be simulated numerically, and computed results can be compared to field data.[3]

The first section of this paper presents a vector multiprocessor implementation of a pseudospectral 2D elastic model and compares it, from the viewpoint of performance, with a conventional 2D finite-difference model, fourth-order accurate in space and second-order accurate in time. Both implementations exploit the IBM 3090 architecture together with its associated software. A finite-difference scheme has a lower operation count per grid point than a pseudospectral (Fourier) method, but the latter, for a specified accuracy, requires fewer grid points than the finite-difference method.[4]

In the second section we deal with the problem of estimating the stacking velocity efficiently. Basic algorithms for the estimation of stacking velocity date back to the late 1960s.[5,6] The conventional stacking velocity algorithms in the hyperbolic time-offset domain can be generalized to the case of analytic[7] seismic traces. Complex-valued coherency functionals may then be introduced.[8] This section reports performance data of a vector/parallel implementation on the IBM 3090 of some characteristic seismic velocity estimation algorithms.

The last section of this paper deals with seismic migration. Efficient and accurate algorithms have been developed for downward extrapolation, based on Fourier transform methods. These frequency-domain approaches have proven to be more accurate than conventional finite-difference methods in the space–time coordinate frame. The seismic migration section summarizes the results obtained implementing *phase-shift plus interpolation (PSPI)*, a frequency-domain, pseudospectral migration method, on the IBM 3090 Vector Multiprocessor.

## The forward-modeling problem: Elastic modeling

**Elastic wave modeling.** The basic equations governing wave propagation in a 2D continuous elastic medium are the momentum-conservation and the stress–strain relations.[9] Following Reference 10, we make use of a derived set of wave equations which contain *stresses* as variables instead of displacements; i.e.,

$$\ddot{\sigma} = A\sigma + Bf, \tag{1}$$

where

$$\sigma = \begin{pmatrix} \sigma_{xx} \\ \sigma_{zz} \\ \sigma_{xz} \end{pmatrix},$$

$$A = LD^T \frac{1}{\rho} D,$$

$$B = LD^T,$$

$$L = \begin{bmatrix} \lambda + 2\mu & \lambda & 0 \\ \lambda & \lambda + 2\mu & 0 \\ 0 & 0 & \mu \end{bmatrix},$$

$$D = \begin{bmatrix} \partial_x & 0 & \partial_z \\ 0 & \partial_z & \partial_x \end{bmatrix},$$

$$\text{and } f = \begin{pmatrix} f_x \\ f_z \end{pmatrix}, \tag{2}$$

where $x$ and $z$ are, respectively, horizontal and vertical Cartesian coordinates; $\sigma_{xx}$, $\sigma_{zz}$, and $\sigma_{xz}$ are the three stress components; $f_x$ and $f_z$ represent the body forces; $\rho(x, z)$ represents the density; and $\lambda = \lambda(x, z)$ and $\mu = \mu(x, z)$ represent Lame's elastic parameters. Here a dot above a variable represents a time derivative and $T$ denotes matrix transposition. In the direct (modeling) problem, the stresses serve as the main unknowns, whereas the rock parameters $\rho$, $\lambda$, and $\mu$, as well as the body forces $f_x(x, z, t)$ and $f_z(x, z, t)$, are known quantities. Appropriate initial and boundary conditions are to be prescribed: In this work we consider exclusively absorbing boundary conditions.[11]

In order to solve Equation (1) numerically, a *discretization in time* is first performed, using a leapfrog technique:

$$\sigma(t + \Delta t) - 2\sigma(t) + \sigma(t - \Delta t) = (\Delta t)^2[A\sigma(t) + Bf(t)]. \tag{3}$$

Then a *discretization in space* is operated, using either a pseudospectral approach similar to that described in Reference 10, or a conventional fourth-order finite-difference scheme.

In the pseudospectral approach, the spatial derivatives appearing in Equation (2) are accurately computed in the Fourier (spectral) domain. In other words, the differential operators $\partial_x$ and $\partial_z$ appearing in (2) are represented with $\mathbf{D}_x$ and $\mathbf{D}_z$ respectively, where

$$\mathbf{D}_\gamma = \mathbf{F}_\gamma^{-1} i K_\gamma \mathbf{F}_\gamma \qquad (\gamma = x, z), \tag{4}$$

with $\mathbf{F}_\gamma$ and $\mathbf{F}_\gamma^{-1}$ representing, respectively, the direct and inverse Fourier transform operators [efficiently

implemented by the use of the fast Fourier transform (FFT)].[12] $K_\gamma$ is the operator multiplying each Fourier coefficient by its wave number, and $i = \sqrt{-1}$. The *computational cost* per time step of the pseudospectral modeling algorithm is

$$\frac{\mathbf{C}}{N_t} = 20N_xN_z\log_2(N_xN_z) + 40N_xN_z + 12N_xNSTRIP_z + 12N_zNSTRIP_x, \quad (5)$$

where $NSTRIP_x$ and $NSTRIP_z$ are absorption strip widths in the $x$ and $z$ directions, respectively, introduced to implement absorbing boundary conditions. The first term in Equation (5) accounts for Fourier transformations, the second term for time advancing with Equation (3), the last two for absorption. For a medium-size 2D numerical mesh of $N_x = N_z = 256$, $N_t = 1024$, the number of floating-point operations performed in a simulation run is of the order of $10^{10}$.

In the finite-difference approach, the spatial derivatives are computed by approximating the differential operators $\partial_x$ and $\partial_z$ appearing in (2) with $\mathbf{d}_x$ and $\mathbf{d}_z$, where

$$\mathbf{d}_\gamma = \frac{1}{2h_\gamma}\left(\frac{1}{6}\mathbf{E}_\gamma^{-2} - \frac{4}{3}\mathbf{E}_\gamma^{-1} + \frac{4}{3}\mathbf{E}_\gamma^{1} - \frac{1}{6}\mathbf{E}_\gamma^{2}\right) = \mathbf{D}_\gamma + O(h_\gamma^4) \quad (\gamma = x, z), \quad (6)$$

with $\mathbf{E}_\gamma$ representing the translation operator and $h_\gamma$ the grid spacing in the $\gamma$ direction. The *computational cost* per time step of the finite-difference modeling algorithm is

$$\frac{\mathbf{C}}{N_t} = 80N_xN_z + 12N_xNSTRIP_z + 12N_zNSTRIP_x. \quad (7)$$

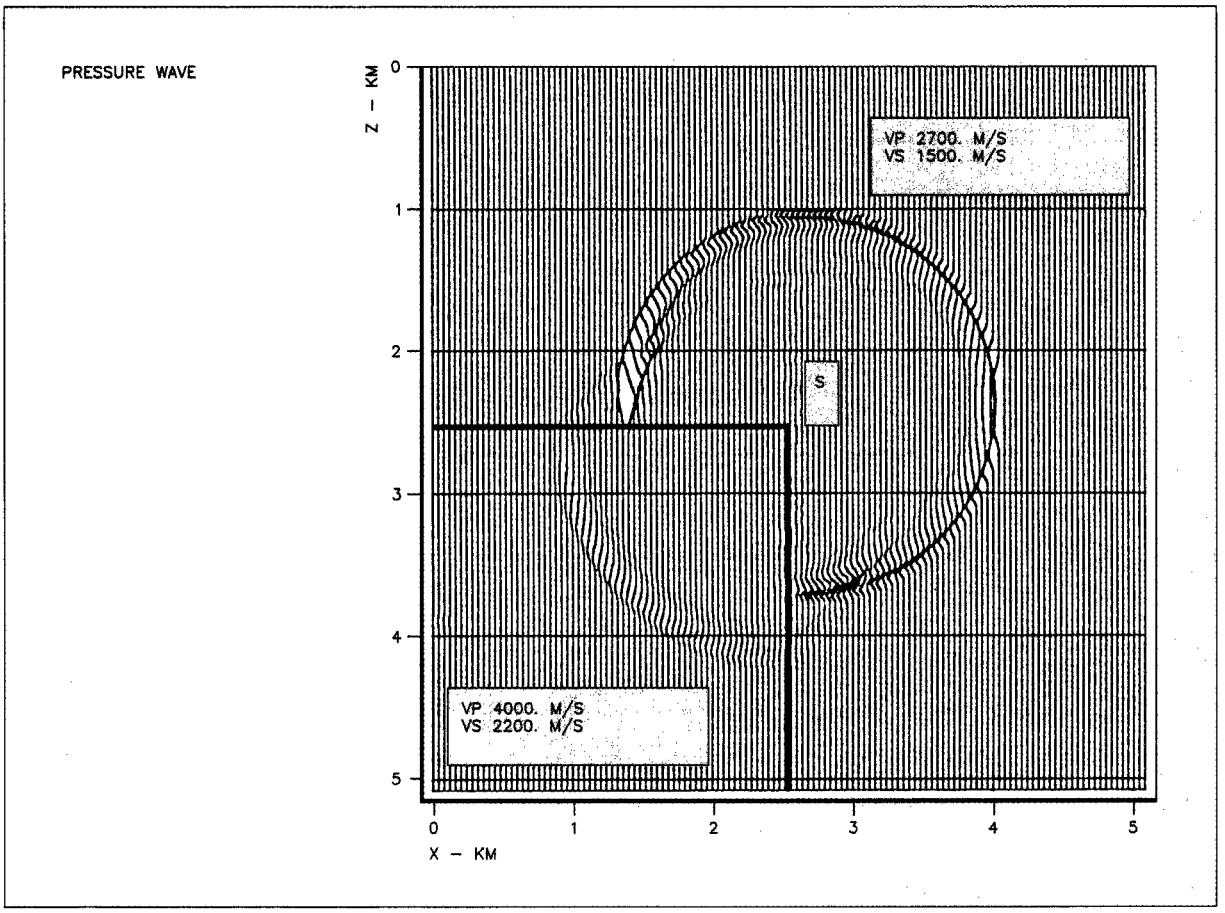
With large numerical meshes or 3D modeling, it becomes essential to look for computing techniques that speed up the execution of such codes.

**Vector/parallel implementation.** The development work discussed in this paper was primarily carried out on the IBM 3090 200 Vector Facility—later upgraded to a 3090 400E—at the IBM European Center for Scientific and Engineering Computing (ECSEC). The IBM 3090 400E computer[13,14] at ECSEC has four independent serial processors which share 128 MB of central memory and 256 MB of expanded storage. Each processor has a 64-KB memory cache, a Vector Facility with sixteen 32-bit vector registers 128 elements long, and operates at a cycle of 17.2 ns, with a theoretical peak speed of 116 million floating-point operations per second. Additional runs were performed on the machine of the Systems Evaluation Laboratory at the IBM Washington Systems Center, a six-way 3090 600E with 256 MB of central memory and 1024 MB of expanded storage.

The 3090 VF architecture is well suited to numerical schemes solving evolutionary PDE such as (1), and particularly to the pseudospectral algorithm [(3), (4)], since it permits very efficient computation of the FFTs using the vector hardware, and at the same time a parallel decomposition of the computation using the 3090 multiple processors; similar considerations hold for the finite-difference scheme [(3), (6)]. In order to utilize the Vector Facility efficiently, we have used routines from the ESSL library[15,16] to compute the direct and inverse Fourier transforms. In addition, we have used the VS FORTRAN Vectorizing Compiler to vectorize the remaining computations.[17,18]

With respect to the parallel decomposition of the algorithms, we have used a simplified *domain decomposition* technique. This strategy makes use of the fact that with computational arrays oriented along the Cartesian coordinates $x$ and $z$, space derivatives operate on individual columns/rows independently of one another. Thus, the problem domain can be decomposed into strips, and each processor handles the operation for its strip. The orientation of the parallel strips must be switched from row to column or vice versa when the direction of the partial derivative is switched. Equivalently, the data arrays can be transposed and the parallel strips kept always oriented columnwise, as recommended for faster execution in FORTRAN. This constitutes an important difference between pseudospectral and finite-difference schemes. The pseudospectral schemes are nonlocal in the sense that in order to compute derivatives in the $x$ and $z$ directions at a given point, they must access all the data corresponding to the row and the column of the point. Conversely, finite-difference schemes are local, and for each point, only data in a relatively small neighborhood need be considered. This difference has an important implication when one is trying to parallelize the algorithm. For finite-difference schemes, the parallel processors can be permanently assigned a data subdomain, and need only exchange or share the data at the boundaries of the subdomains. In Fourier methods, on the contrary, the parallel processors must access (for example, in the $x$ differentiation), data previously assigned to other processors (in the $z$ sweep), thus leading to a greater volume of communication.

Figure 2 Pressure wave snapshot for the corner problem (high velocity inside the corner) as computed by the elastic pseudospectral model




With respect to the software support for parallelization, we have examined two alternatives for the pseudospectral code. The first divides the work among the available processors with the help of the VS FORTRAN Multitasking Facility (MTF).[18] MTF is based on DSPTCH/SYNCRO primitives, to allow the asynchronous execution of subroutines on multiple processors sharing a common memory. For more details on this implementation, see Reference 19. The second alternative makes use of Parallel FORTRAN (PF)[20] constructs, which allow a finer level of parallelism, down to the grain of a DO loop. To implement the finite-difference scheme, we have simply vectorized and parallelized it with PF.

As an example of the numerical results obtained with the elastic model, Figure 2 shows the response of a geological model corresponding to a corner structure. In Figure 2, the terms VP and VS refer to pressure and shear velocities, respectively, and S indicates the source location. Calculations were made using a pressure-point source with a high-cut frequency of 40 Hz for a numerical mesh of 256 × 256 nodes with a grid spacing of 20 m in both $x$ and $z$ directions and a 0.001-s time step. For a quantitative assessment of the geophysical significance of the pseudospectral elastic model, refer to Reference 21.

**Performance analysis.** The sustained performance of the pseudospectral algorithm is presented in Table 1, which includes the CPU time per step in seconds, the percentage of time used in computing FFTs, the sustained performance in millions of floating-point operations per second (Mflop/s), and the speedup, for a resolution of 128 × 128 grid points: Here and consistently through the rest of the paper, all timings

Figure 3 Time distribution of machine instructions in the execution of finite-difference elastic modeling code on the IBM 3090 running in scalar mode

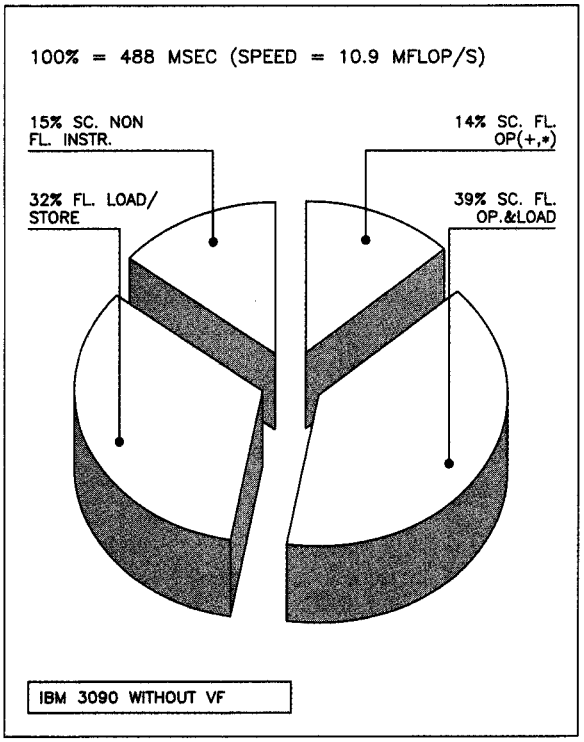



Figure 4 Time distribution of machine instructions in the execution of finite-difference elastic modeling code on the IBM 3090 running in vector mode

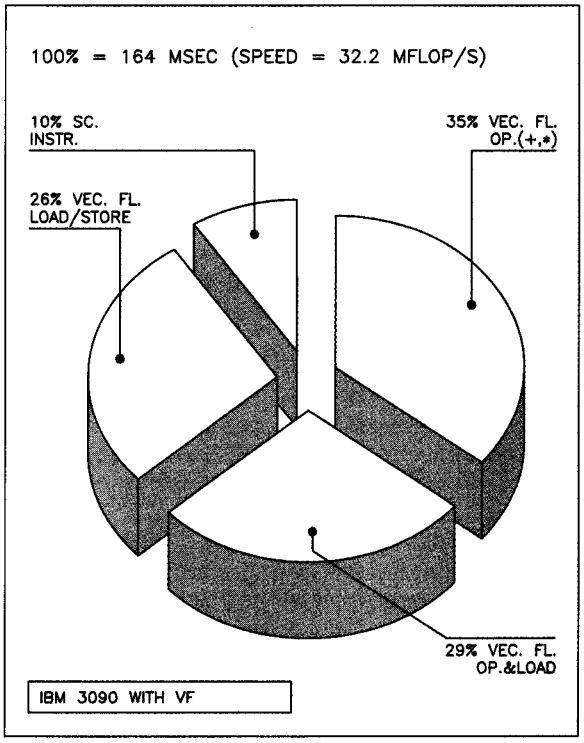



Table 1 Performance of different implementations of the pseudospectral elastic modeling algorithm for the 128 × 128 grid on the IBM 3090 200 VF (main storage = 64 MB; expanded storage = 128 MB) using MTF for parallel execution and ESSL (Release 1) for Fourier transformations.[22] Computations are done in single (32-bit) precision.

| Configuration | Processing | CPU Time per Step (s) | FFT % | Mflop/s | Speedup |
|---|---|---|---|---|---|
| A 3090 Uniprocessor | Scalar–Serial | 0.99 | 87.1 | 5.31 | |
| B 3090 200 | Scalar–Parallel, MTF | 0.53 | 83.5 | 9.96 | 1.87 (B/A) |
| C 3090 VF Uniprocessor | Vectorized | 0.19 | 66.5 | 28.3 | 5.21 (C/A) |
| D 3090 200 VF | Vector–Parallel, MTF | 0.12 | 59.8 | 45.4 | 1.58 (D/C) |

are taken in standalone mode, with computations performed in single (32-bit) precision.

The Mflop/s rates are obtained using Equation (5) to calculate the number of floating-point operations per time step. Equation (5) for the pseudospectral method and the companion equation (7) for the finite-difference method consider only floating-point additions and multiplications, and therefore neglect integer arithmetic, transpositions, and loads/stores, although it is well known that they play a nonnegligible role in the balance: Evidence is provided in Figures 3 and 4 showing the time distribution of IBM System/370 machine instructions in the execution of the finite-difference elastic modeling code [(3), (6)].

As is shown in Case C in Table 1, running the vectorized algorithm on the 3090 VF produces very good performance results, with a 3090 vector/scalar ratio of 5.21. This ratio is 7 for the computation of FFTs with the ESSL routines, and ranges from 2.2 to 2.4 for the remaining computations.[22]

In the *scalar* multitasking implementation of the decomposition technique (Table 1, Case B), the parallelized parts of the code correspond to 96 percent of the execution time (essentially, Fourier transforms and time stepping), while matrix transpositions and computation of the absorption are performed in serial mode. The speedup (parallel/serial) ranges from 1.6 to 1.94 for the individual sections of the algorithm, leading to an overall two-way speedup of 1.87 on the two-way multiprocessor 3090.

In the *vectorized* and multitasking case (Table 1, Case D), the speedup is only 1.58, because the parallelized portion of the code is also vectorizable, and the use of the Vector Facility shrinks the parallelized fraction to 87 percent of the total execution time; speedups for individual sections range from 1.5 to 1.9. However, as shown in Table 2, with increased grid sizes one obtains at the same time overall two-way speedups greater than 1.75 and higher vector/scalar speedups. This is due to the fact that the percentage of floating-point operations associated with FFT computations increases with grid size, and these computations are very efficiently handled. The key factor in performance improvement is the increased number of floating-point operations per memory reference.

It is worth noting that, for the cases in which the problem size does not fit in real storage (last column of Table 2), the data are allocated to the expanded storage of the 3090[13] (the specific configuration used in this experiment had 64 MB of real storage and 128 MB of expanded storage). The operating system then pages the data in and out of real storage at a rate of approximately 75 $\mu$s per page fault. This leads to a degradation in performance, as can be observed from the results shown in Table 2 for the 1024 × 1024 problem size. Nevertheless, this performance is still very satisfactory; moreover, since the operating system takes care of paging, it is possible to run very large models without explicit memory management.

The upper portion of Table 3 shows performance data on the six-way 3090 600E for a grid size of 512 × 512, in the vectorized and parallelized case. Also shown is the fraction of the code which runs in parallel, which in this case corresponds to 92 percent of the serial time. It is observed that although this fraction is rather high, the relative weight of the parallelized fraction decreases rapidly with the number of processors, so that with six processors it is only 66 percent. To fully exploit the parallel potential of the code, therefore, it was necessary also to parallelize the transpositions and the absorption. This was achieved quite easily by using the PF constructs shown in Figure 5. In this case the subroutines parallelized are SCRFT and SRCFT, performing the FFT algorithm, and the subroutine WAVE, performing the multiplication by wavenumbers; loops 20 and 70 are parallelized. Figure 6 shows, for the elementary kernel of matrix transposition, the synergism between parallel and vector execution.

**VS FORTRAN Interactive Debug (IAD) is a very useful tool.**

Results are given in the lower part of Table 3 and in Figure 7, which shows the sustained speed, relative to that of the scalar 3090 uniprocessor, as a function of the number of processors. Results are presented for both scalar and vector implementations on the IBM 3090 600E processor complex, showing a very significant combined vector/parallel speedup. Dotted lines show the theoretical linear speedup curves. In the scalar MTF case, where the parallel fraction is 96 percent, the speedup is nearly linear. In the vector MTF case, where the parallel fraction is only 92 percent, it is clear that to improve the performance when using four to six processors, it is necessary to increase the fraction of the code which runs in parallel. This has been done by parallelizing the transposition and absorption using PF.

The sustained performance of the different versions of the finite-difference code for different grid sizes is given in Table 4. Vector/scalar speedups of the order of 3 are observed, together with an almost linear speedup when the code is parallelized with PF.

It is worth mentioning that the VS FORTRAN Interactive Debug (IAD)[23] is a very useful tool that identifies computation-intensive portions of application programs, providing statistical analyses and summary information at subroutine and statement levels. Moreover, the activity can be graphically displayed, as shown in Figure 8. Asterisks and the associated values indicate CPU percentage spent at the corresponding statement.

**Table 2** Single-precision performance of the pseudospectral elastic code for different grid sizes on the IBM 3090 200 VF (main storage = 64 MB; expanded storage = 128 MB) using MTF for parallel execution and ESSL (Release 1) for FFT computations. The program data fit in the main memory except in the $1024^2$ case (last column) which requires approximately 100 MB.

| Configuration | Grid Size | | | | |
|---|---|---|---|---|---|
| | $64^2$ | $128^2$ | $256^2$ | $512^2$ | $1024^2$ |
| | Speed (Mflop/s) | | | | |
| 3090 Uniprocessor | 4.6 | 5.31 | 5.30 | 5.52 | 4.54 |
| 3090 200 | 8.1 | 9.96 | 10.13 | 10.38 | 8.02 |
| 3090 VF Uniprocessor | 21.5 | 28.3 | 31.1 | 34.3 | 26.9 |
| 3090 200 VF | 28.5 | 45.4 | 54.2 | 61.3 | 43.2 |

**Table 3** Single-precision performance on the IBM 3090 600E VF and on the IBM 3090 400E of the pseudospectral elastic modeling code for a resolution of 512 × 512, as a function of the number of processors, using MTF and PF for parallel execution. Program data (25 MB) fit in main memory.

| Configuration | Processors | Time (s) | Parallel % | Mflop/s | Speedup |
|---|---|---|---|---|---|
| 3090 600E, MTF | 1 (1 VF) | 2.64 | 92.0 | 39.7 | 1.00 |
| 3090 600E, MTF | 2 (2 VF) | 1.45 | 84.8 | 72.6 | 1.82 |
| 3090 600E, MTF | 3 (3 VF) | 1.04 | 79.0 | 100.7 | 2.54 |
| 3090 600E, MTF | 4 (4 VF) | 0.853 | 73.3 | 123.1 | 3.09 |
| 3090 600E, MTF | 5 (5 VF) | 0.737 | 68.8 | 142.5 | 3.58 |
| 3090 600E, MTF | 6 (6 VF) | 0.665 | 66.3 | 158.0 | 3.97 |
| 3090 400E, PF | 1 (1 VF) | 2.71 | | 38.6 | 1.00 |
| 3090 400E, PF | 2 (2 VF) | 1.38 | | 75.8 | 1.96 |
| 3090 400E, PF | 3 (3 VF) | 0.93 | | 112.8 | 2.89 |
| 3090 400E, PF | 4 (4 VF) | 0.73 | | 144.2 | 3.73 |
| 3090 600E, PF | 5 (5 VF) | 0.61 | | 173.7 | 4.50 |
| 3090 600E, PF | 6 (6 VF) | 0.54 | | 195.6 | 5.05 |

**Table 4** Single-precision performance of the finite-difference elastic code for different grid sizes using PF for parallel execution on the IBM 3090 400E VF (main storage = 128 MB; expanded storage = 256 MB) and on the IBM 3090 600E VF (main storage = 256 MB; expanded storage = 1024 MB).

| Configuration | Grid Size ‖ Memory (MB) | | | |
|---|---|---|---|---|
| | $127^2$ ‖ 1.28 | $255^2$ ‖ 5.12 | $511^2$ ‖ 20.48 | $1023^2$ ‖ 81.52 |
| | Time (ms) \| Speed (Mflop/s) | | | |
| 3090 400E (1 scalar processor) | 115 \| 11.4 | 488 \| 10.9 | | |
| 3090 400E (1 VF) | 40.0 \| 32.7 | 164 \| 32.2 | 682 \| 31.0 | 2767 \| 30.6 |
| 3090 400E (2 VF) | 22.1 \| 59.4 | 84.7 \| 62.4 | 351 \| 60.3 | 1454 \| 58.2 |
| 3090 400E (3 VF) | | 59.7 \| 88.5 | 237 \| 89.2 | 970 \| 87.2 |
| 3090 400E (4 VF) | | 46.6 \| 113.2 | 182 \| 116.2 | 746 \| 113.5 |
| 3090 600E (6 VF) | | | 132 \| 159.4 | 537.6 \| 157.5 |

**Figure 5** Using the PF constructs SCHEDULE and WAIT and directives to parallelize the execution of subroutine calls and transposition loops in the pseudospectral elastic code

```
C              Differentiate F with respect to x and save in FX
C
C      (1)     Transpose F, store in FT
C
C$PARA IGNORE RECRDEPS
             DO 20 K=1,NPROC
C$PARA PREFER VECTOR
             DO 10 J=1,NX
                DO 10 I=(K-1)*CHUN+1,K*CHUN
10                 FT(J,I)=F(I,J)
20           CONTINUE
C
C       (2)     Fourier transform
C
             DO 30 I=2,NPROC
          SCHEDULE ANY TASK ITASK(I),
     *    CALLING SRCFT(0,FT(1,YINIT(I)),MX,DFX(1,1,YINIT(I)),
     *                  STR,NX,NYH,1,1.,
     *                  ADX1(1,I),NDX1,ADX2(1,I),NDX2,ADX3(1,I),NDX3)
30         CONTINUE
          CALL SRCFT(0,FT(1,1),MX,DFX(1,1,1),STR,NX,NYH,1,1.,
     *             ADX1(1,1),NDX1,ADX2(1,1),NDX2,ADX3(1,1),NDX3)
          WAIT FOR ALL TASKS
C
C       (3)     Multiply by wave numbers
C
              DO 40 I=2,NPROC
            SCHEDULE ANY TASK ITASK(I),
     *      CALLING WAVE(MXH1,MY,NXH1,YINIT(I),YFIN(I),DFX,CSKX)
40             CONTINUE
            CALL WAVE(MXH1,MY,NXH1,YINIT(1),YFIN(1),DFX,CSKX)
            WAIT FOR ALL TASKS
C
C        (4)    Inverse Fourier transform
C
             DO 50 I=2,NPROC
          SCHEDULE ANY TASK ITASK(I),
     *    CALLING SCRFT(0,DFX(1,1,YINIT(I)),STR,FT(1,YINIT(I)),MX,
     *                  NX,NYH,-1,INX,
     *                  AIX1(1,I),NIX1,AIX2(1,I),NIX2,AIX3(1,I),NIX3)
50           CONTINUE
          CALL SCRFT(0,DFX(1,1,1),STR,FT(1,1),MX,NX,NYH,-1,INX,
     *             AIX1(1,1),NIX1,AIX2(1,1),NIX2,AIX3(1,1),NIX3)
          WAIT FOR ALL TASKS
C
C          (5)   Transpose the result and store in FX
C
C$PARA IGNORE RECRDEPS
             DO 70 K=1,NPROC
C$PARA PREFER VECTOR
             DO 60 J=1,NX
               DO 60 I=(K-1)*CHUN+1,K*CHUN
60                  FX(J,I)=FT(I,J)
70           CONTINUE
      RETURN
      END
```

**Figure 6** Synergism between parallel and vector execution in the matrix transposition kernel compiled with PF

```
              C$DIR IGNORE RECRDEPS
PARA +-------       DO 10 IPROC=1,NPROC
     |        C$DIR PREFER VECTOR
VECT |+------          DO 20 J=1,NY
ELIG ||+-----           DO 20 I=(IPROC-1)*NCHUNK+1,IPROC*NCHUNK
     |||____               FT(J,I)=F(I,J)
     ||_____
     |______
```

**Figure 7** Sustained speed on the IBM 3090 600E multiprocessor, for vector and scalar processing, relative to the scalar uniprocessor for the elastic pseudospectral code [22]

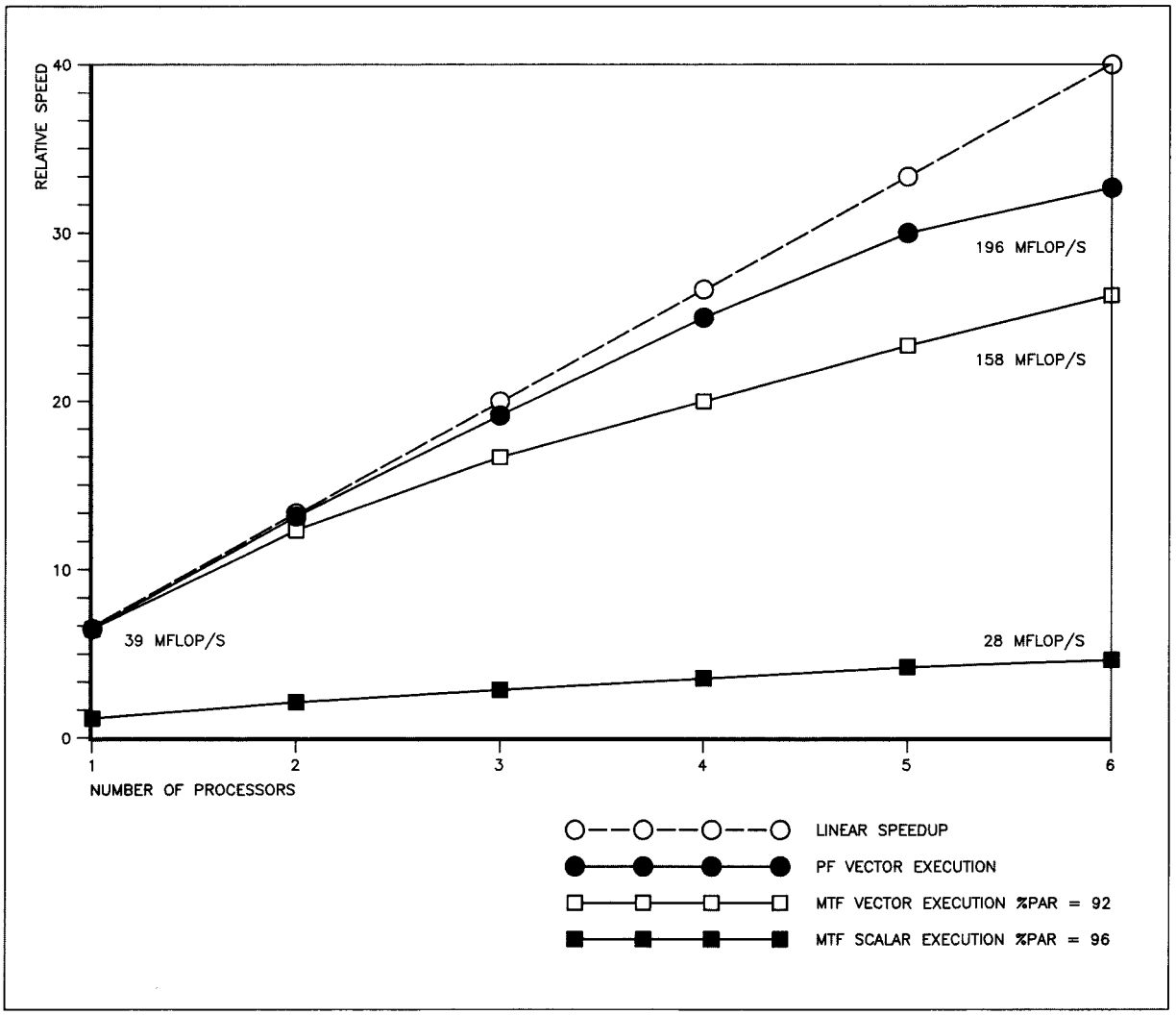


Figure 8 IAD annotated listing (with sampling option) of a part of the 2-D finite difference elastic code

```
      DO 1 J=3,NX-2
         DO 1 I=3,NY-2
            FY(I,J)=(CIY*(SYY(I-2,J)-SYY(I+2,J))                    14.73 ********
     *+C2Y*(SYY(I+1,J)-SYY(I-1,J))
     *+C1X*(SXY(I,J-2)-SXY(I,J+2))
     *+C2X*(SXY(I,J+1)-SXY(I,J-1)))*FIDEN(I,J)
            FX(I,J)=(CIX*(SXX(I,J-2)-SXX(I,J+2))                    14.45 ********
     *+C2X*(SXX(I,J+1)-SXX(I,J-1))
     *+C1Y*(SXY(I-2,J)-SXY(I+2,J))
     *+C2Y*(SXY(I+1,J)-SXY(I-1,J)))*FIDEN(I,J)
    1 CONTINUE                                                      0.28
```

## The inverse problem: Seismic velocity estimation

**Problem formulation.** Conventionally, the stacking velocity estimation is performed directly in the offset-time domain $(x, t)$ on one or more gathers,

$$p = p(x, t), \tag{8}$$

where $x$ is the source–receiver offset, $t$ is the two-way travel time, and $p$ is the observed wavefield; basically, the physical domain $(x, t)$ is scanned with the two-parameter family of curves,

$$T^2(x; t_0, v) = t_0^2 + \frac{x^2}{v^2}, \tag{9}$$

looking for hyperbolic alignments of the data $p(x, t)$. From the algorithmic viewpoint, the velocity estimation procedure consists of two steps, to be repeated for every stacking velocity $v$:

1. The nonzero-offset data $p$ are transformed into zero-offset data $\bar{p}$ by a hyperbolic coordinate transformation consisting of a time-variable shift, the *Normal Move-Out* (NMO) correction

   $$\bar{p} = \bar{p}(x, t_0; v) = p[x, T(x; t_0, v)], \tag{10}$$

   where $T(x)$ is given by Equation (9). In this way the data $p$, originally characterized by hyperbolic alignments (9), are transformed into data $\bar{p}$ with straight-line patterns $t_0 = \text{const}$.
2. For each travel time $t_0$, the uniformity of the data $\bar{p}(x, t_0; v)$, NMO-corrected with the stacking velocity $v$, is quantitatively evaluated by means of some *coherency* functional $F$ and assigned to the *velocity spectrum* $F(v, t_0)$.

**Coherency functionals.** Numerous functionals have been proposed to evaluate quantitatively the goodness of fit obtained on a given gather with a certain stacking velocity $v$. The most common functionals measure the similarity of the traces of the NMO-corrected gather, and are based on either the summation of the traces or the correlation of the traces with various choices of normalization. A widely used coherency measure[6] is the *semblance*

$$\tilde{S}(v, t_0) = \frac{\sum_{t=t_0-\delta/2}^{t_0+\delta/2} \left[\frac{1}{N_x} \sum_{x=x_0}^{X} \bar{p}(x, t; v)\right]^2}{\sum_{t=t_0-\delta/2}^{t_0+\delta/2} \frac{1}{N_x} \sum_{x=x_0}^{X} [\bar{p}(x, t; v)]^2}. \tag{11}$$

All the coherency functionals can immediately be generalized, following Reference 7, to the case of *complex-valued* gathers

$$\psi(x, t) = p(x, t) + iq(x, t), \tag{12}$$

where $p(x, t)$ is an ordinary real-valued gather and $q(x, t)$ is obtained from $p(x, t)$ by the application of the Hilbert transform with respect to time $t$. To this new class belongs, by example, the *statistically normalized complex-correlation* functional

$$\tilde{\Gamma}(v, t_0) = \frac{2}{N_x(N_x - 1)} \sum_{x=x_0}^{X} \sum_{z>x}^{X} \frac{\sum_{t=t_0-\delta/2}^{t_0+\delta/2} \bar{\psi}^*(x, t; v)\bar{\psi}(z, t; v)}{\left[\sum_{t=t_0-\delta/2}^{t_0+\delta/2} |\bar{\psi}(x, t; v)|^2\right]^{1/2} \left[\sum_{t=t_0-\delta/2}^{t_0+\delta/2} |\bar{\psi}(z, t; v)|^2\right]^{1/2}}, \tag{13}$$

where * denotes complex conjugation.

Figure 9 shows a seismic gather of marine data. The corresponding velocity spectrum, computed with the help of the estimator (13), is shown in Figure 10. The coherency functional uses the statistical cross-correlation (13) and is enhanced with the postprocessing technique described in Reference 8. The velocity is sampled with an interval $\Delta v = 15$ m/s.

**Parallel algorithms for stacking-velocity estimation.** For a medium-size problem ($N_v = 100$, $N_x = 50$, $N_t = 1000$), the cost of a velocity estimation is of the order of $10^8$ flops. If a large number of gathers must be processed, or interactivity is needed, it becomes essential to look for computing techniques that speed up the execution. As indicated in the previous dis-

**Figure 9** CMP gather of marine data (Black Sea): the sampling interval is $\Delta t = 4$ ms; the trace spacing is 50 m

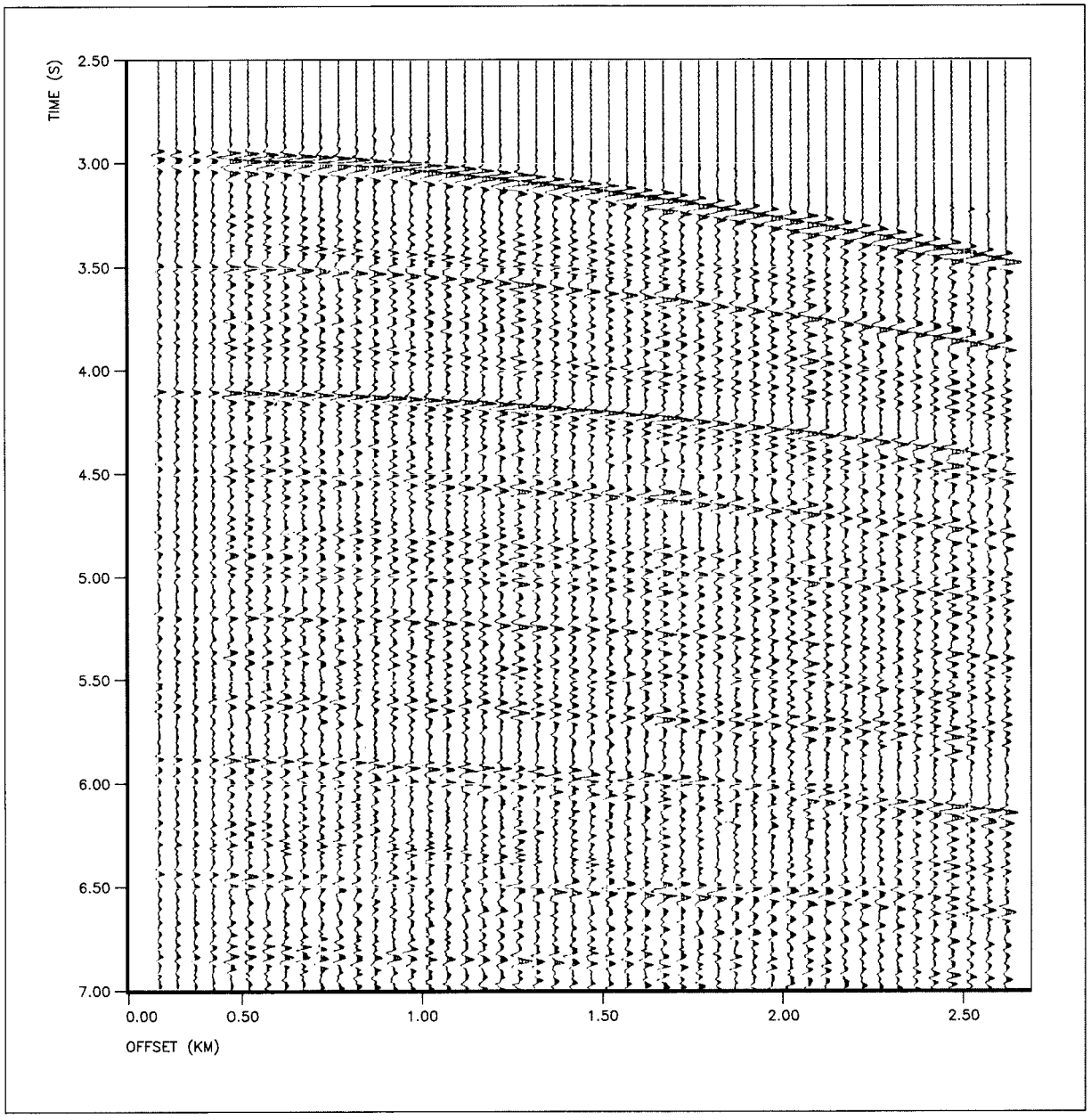
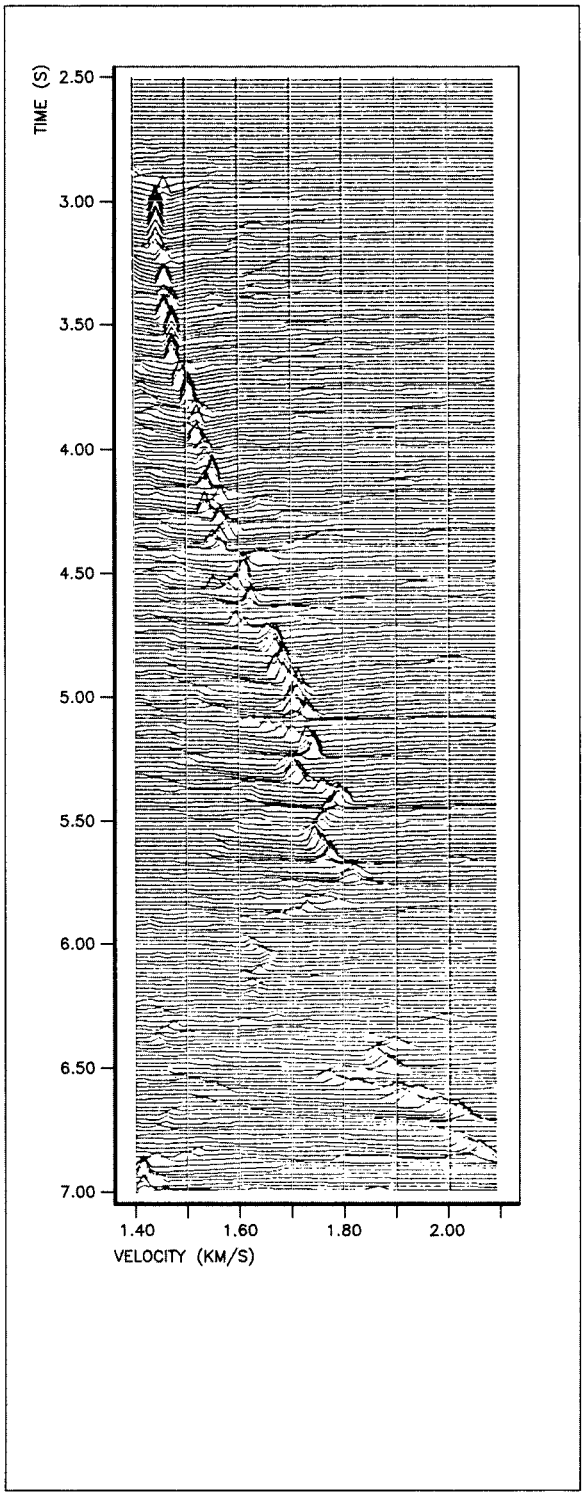


Figure 10 Velocity spectrum of the marine CMP gather of Figure 9




cussions, the stacking velocity estimation algorithms consist essentially of nested loops, running over the velocities $v$, the offsets $x$, and the time samples $t$. The NMO transformation accounts for more than 50 percent of the cost (see Figure 11) and consists of an offset-dependent stretching of the time axis of the traces. The rest of the computations comprise sample-by-sample additions or multiplications of the traces, in particular as required by the normalization of formulae (11) and (13). Therefore, the velocity estimation algorithms are ideal candidates for implementation on a vector computer architecture where the trace operations are mapped into vector instructions. Furthermore, all vector operations appear in an outer loop running over the trial stacking velocities. Since the computations associated with different velocities $v$ are totally independent, these algorithms are also good candidates for a parallel computer implementation. An optimal performance can be achieved on a vector multiprocessor by splitting the outer parallel loop on several processors which execute the innermost loops over the trace samples in vector mode. Table 5 summarizes the timing results of a FORTRAN-coded implementation of the velocity algorithms (11) and (13) on the IBM 3090 VF Vector Multiprocessor, with the parallelization and vectorization criteria emerging from the previous discussion; for more details, see Reference 8.

It is easily observed that a $q$-fold speedup is effectively measured when executing on $q$ processors of the IBM 3090 vector multiprocessor $q$ identical tasks generating $1/q$ of the spectrum. The $q$ tasks are completely independent and must synchronize only at the end of the computation; therefore, the interprocessor communication and the parallelization overhead are virtually zero. This linear speedup is attributed to the large granularity of the problem and is achieved by modifying the serial code using MTF.

## The inverse problem: Seismic migration

**Wave-equation migration.** Migration can be formulated as a numerical solution to the partial differential equations which govern the back-propagation of the recorded signals from the surface to the reflector locations in reverse time. This approach, generally referred to as wave-equation migration, consists of two steps: *wave extrapolation* and *imaging*. The theory of wave extrapolation is based on the assumption that the zero-offset pressure data, defined in the ($x$, $t$) domain, satisfy the scalar wave equation

$$\frac{\partial^2 p}{\partial z^2} = \frac{4}{v^2}\frac{\partial^2 p}{\partial t^2} - \frac{\partial^2 p}{\partial x^2}, \tag{14}$$

**Figure 11** Sampling analysis of the velocity estimation code using the estimator (13)

```
=AFF010I VS FORTRAN VERSION 2 RELEASE 3 INTERACTIVE DEBUG
=AFF011I 5668-806 (C) COPYRIGHT IBM CORP. 1985, 1988
=AFF013I LICENSED MATERIALS-PROPERTY OF IBM
=AFF933I THE AFFON FILE WAS PROCESSED WITH 0 ERRORS
=AFF995I WHERE: VELANT.15
=* ENDDEBUG SAMPLE(4) CALLED
=AFF306I PROGRAM HAS TERMINATED; RC ( 0)
=* ANNOTATE (COPRO,ANRZK,ITP2C1) SAMPLING   ALL
=AFF561I ANNOTATING LISTING FOR PROGRAM UNIT "COPRO"
=AFF561I ANNOTATING LISTING FOR PROGRAM UNIT "ANRZK"
=AFF561I ANNOTATING LISTING FOR PROGRAM UNIT "ITP2C1"
=AFF224I 481 LINES OF OUTPUT WRITTEN TO AFFPRINT
=* LISTSAMP *  SUMMARY ALL
=AFF550I PROGRAM SAMPLING INTERVAL WAS 4 MS; TOTAL NUMBER OF SAMPLES WAS 3465.
=AFF552I SUM OF DIRECT AND CALLED SAMPLES:
=AFF556I PROGRAM UNIT                SAMPLES  %TOTAL
=AFF558I VELANT                         3465  100.00 ********************
=AFF558I QUADR   (get analytic signal)    27    0.78
=AFF558I COPRO   (estimator  (13)    )  3399   98.10 ********************
=AFF558I ITP2C1  (trace interp. in NMO) 1469   42.40 ********
=AFF558I ANRZK   (normalization)         778   22.45 ****
=AFF558I V#SQRT  (square root)           579   16.71 ***
```

**Table 5** Single-precision performance of two stacking velocity algorithms on the IBM 3090 Vector Multiprocessor operating on a gather consisting of 100 traces (1024 samples per trace) and generating a spectrum consisting of 60 trial velocities and 256 travel times.

| Algorithm *F* | Scalar Uniprocessor Time (s) | Vector Uniprocessor Time (s) | Two-processor Time (s) | Six-processor Time (s) |
|---|---|---|---|---|
| $\tilde{S}$ (11) | 16.6 | 6.0 | 3.0 | 1.0 |
| $\tilde{\Gamma}$ (13) | 47.0 | 19.2 | 9.6 | 3.2 |

with $p = p(x, t, z)$, where $x$ is the horizontal variable, $z$ is depth, $t$ is two-way travel time, and $v = v(x, z)$ is the velocity. The corresponding one-way wave equation in terms of the variable $P(x, \omega, z)$ obtained from $p(x, t, z)$ with a Fourier transform with respect to time, reads

$$\frac{\partial P}{\partial z} = \frac{2i\omega}{v} \mathbf{F}_x^{-1}\left[1 - \left(\frac{k_x v}{2\omega}\right)^2\right]^{1/2} \mathbf{F}_x P, \qquad (15)$$

where $k_x$ is the wavenumber with respect to $x$, $\omega$ is the temporal frequency, and $\mathbf{F}_x$ and $\mathbf{F}_x^{-1}$ represent the direct and inverse complex Fourier transform operators with respect to the direction $x$. Equation (15) is the fundamental equation for downward extrapolation of zero-offset data. It is expressed in the wavenumber-frequency domain $(k_x, \omega)$, and does not have an explicit representation in the physical domain $(x, t)$. In the absence of horizontal velocity dependence, Equation (15), which governs the extrapolation of the zero-offset seismic data, has a simple analytic solution (essentially a *phase shift*).[24] For velocity fields with lateral variations, the square-root pseudodifferential operator appearing in Equation (15) must be approximated in some form, for instance with the approach described in Reference 24, the *phase-shift plus interpolation* (PSPI) method.

Table 6 Performance of the PSPI code on the IBM 3090 600E Vector Multiprocessor (single precision) with ESSL Release 1.

| Scalar Code (uniprocessor) [CPU (s)] | (FFT %) | Vector Code (uniprocessor) [CPU (s)] | (FFT %) | Vector–Parallel (2 processors) [CPU (s)] | Vector–Parallel (6 processors) [CPU (s)] |
|---|---|---|---|---|---|
| 2115.0 | 61.8 | 457.3 | 39.1 | 236.0 | 87.2 |

**Parallel decomposition of seismic migration algorithms.** All frequency-domain methods begin with the Fourier transform of the seismic data in time, thus replacing the independent variable $t$ with $\omega$. Due to the linearity of the migration problem, the calculations carried out for different $\omega$ are simply superimposed to obtain the complete solution. Furthermore, these calculations are completely independent, with virtually no interprocess communication. When all the harmonic components $P(x, \omega, z)$ have been processed, the migrated section is completed. As a particular case, the PSPI migration algorithm consists essentially of three nested loops, and the proposed decomposition partitions the outermost loop (in $\omega$); the vectorization of the innermost loop (in $x$ or $k_x$) is ensured by its structure.[26] The intermediate loop (in $z$) is sequential, representing a downward continuation process, and cannot be parallelized except for the particular case of a horizontally stratified medium. All considerations that hold for the two-dimensional case are also valid for algorithms designed for the migration of three- and four-dimensional data, e.g., migration before stack[27] and 3D migration.[28]

**Performance measurements of the PSPI code.** The vectorizability and speedup characteristics of the PSPI code have been studied by running it and measuring execution times on the 3090 in scalar and vector mode. The problem under consideration[29] consists of the migration of a synthetic zero-offset section of size $512^2$. Table 6 summarizes the results of our measurements by showing the CPU time, in seconds, for the main part of the code. A speedup of 7.3 in performing FFTs combined with a 2.9 speedup in performing add/multiply operations leads to the observed 4.6 vector/scalar speedup.

As mentioned earlier, the data and the computations associated with different frequencies $\omega$ are independent. We take advantage of this fact to run the loop over the frequencies in parallel on the six CPUs of the 3090 VF. The software tool used is the VS FORTRAN Version 2 Multitasking Facility. The same effect could have been achieved had we used PF. Figure 12 shows a piece of the code with MTF, while Figure 13 is a summary of the most computation-intensive sections of the code as given by IAD. When we dispatch six identical tasks that process one sixth of the frequencies each, using the vector structure and the parallelism inherent in the problem, we obtain an effective speedup of above 20 compared to the scalar uniprocessor version.

## Concluding remarks

The first section of this paper has presented vector multiprocessor implementations of elastic modeling algorithms based on pseudospectral (Fourier) as well as finite-difference methods. The algorithms are very well suited to *vector multiprocessors*, on which significant performance improvements are obtained by simultaneously vectorizing the innermost loops and parallelizing the outer loops. In the last section, an analysis of the suitability for parallel processing of frequency-domain seismic migration methods has been presented. The frequency-domain methods are easily decomposable into parallel tasks having large granularity, and require very limited interprocessor communication. Among the frequency-domain methods, optimal results (in terms of accuracy) can be obtained by the PSPI method. This algorithm can be adapted conveniently to the IBM 3090 Vector Multiprocessor architecture. Frequency-domain methods lend themselves most conveniently to parallel formulation, since there is no interdependence among data associated with different temporal frequencies. Similar considerations hold for the problem dealt with in the second section of this paper, namely, seismic velocity estimation.

The seismic computations presented in this paper share some common characteristics: large amounts of data to be handled, high floating-point content, and heavy use of arrays. Furthermore, each problem can be naturally decomposed into a set of independent tasks which require minimum synchronization.

These three characteristic problems indicate that in seismic computations there is a very large potential for *coarse-grain* parallelism at the *algorithmic* level,

**Figure 12 Part of the migration code using the MTF primitives DSPTCH and SYNCRO**

```
      SUBROUTINE KPSPIV
C     read input data (time section)
            CALL TSIN(P,NT,NX,IUUMIG,TSTCSE)
C     Fourier transform in time the input data and transpose
            CALL TRFTRP(P,POM,NT,NX,NT2,W,L2NT)
C     read velocity field and transpose
            CALL CIN(C,NZ,NX,IUC,TSTCSE)
            CALL RTRAP(CT,NX,NZ,C)
C     transform velocity field into normalized refraction index
            CALL CTRAN(CT,NX,NZ,REMN,REMX,CREF)
C     compute the negative of the squares of wavenumbers
            CALL NGSQA(ALFA2,NX,DX)
C     zero output matrix
            CALL VZMV$$(QT,NX*NZ,1)
C============= begin parallel migration in the frequency domain
C           number of frequencies per process
            NOMBLK=NOMEFF/NPROC
C ---------------- loop over blocks of layers (sequential)
      DO 3000 IZ=1,NZEFF,NZBLK
C ---------------- loop over frequency bands (parallel)-----
        DO 1024 IBLKS=1,NPROC-1
         IAMIN(IBLKS)=1+(IBLKS-1)*NOMBLK
         JAM=IAMIN(IBLKS)+NOMBLK-1
C============= fork=======================================
         CALL DSPTCH ('FDMGAV',
     *               NT,NX,NZ,DT,DX,DZ,NOMEFF,NZEFF,IDZDZC,
     $                   L2NX,W(1,1,IBLKS),CREF,REMN(IZ),REMX(IZ),
     $                   POM(1,IAMIN(IBLKS)),CT(1,IZ),
     $                   QTO(1,1,IBLKS),NOM,IAMIN(IBLKS),
     $                   NOMBLK,NZBLK,PZ(1,IBLKS),PZDZ(1,IBLKS),
     $                   ALFA2,IZ,FRCSTR,LNEAR,INTRIG,EPSRFI,
     $                   CLOA,NZBLM)
1024    CONTINUE
C
         IAMIN(NPROC)=1+(NPROC-1)*NOMBLK
         NOMBL2=NOMEFF-IAMIN(NPROC)+1
         CALL FDMGAV(
     *               NT,NX,NZ,DT,DX,DZ,NOMEFF,NZEFF,IDZDZC,
     $                   L2NX,W(1,1,NPROC),CREF,REMN(IZ),REMX(IZ),
     $                   POM(1,IAMIN(NPROC)),CT(1,IZ),
     $                   QTO(1,1,NPROC),NOM,IAMIN(NPROC),
     $                   NOMBL2,NZBLK,PZ(1,NPROC),PZDZ(1,NPROC),
     $                   ALFA2,IZ,FRCSTR,LNEAR,INTRIG,EPSRFI,
     $                   CLOB,NZBLM)
C============= join=======================================
          CALL SYNCRO
C -------------- add harmonic images ----------------------
         DO 1025 IBLKS=1,NPROC
1025     CALL F$ADD(LOCK,QT(1,IZ),QTO(1,IZ,IBLKS),NX*NZBLK,1,1)
C ------------------------------------------------------------
         JAM=IAM+NOMBLK-1
         JBM=IBM+NOMBLK-1
 3000 CONTINUE
C============= end of parallel migration in the frequency domain
C     transpose migrated section
      CALL RTRAP(Q,NZ,NX,QT)
C     write output data (depth migrated section)
      CALL ZSOUT(Q,NZ,NX,IUMIG,TSTCSE)
      RETURN
      END
```

**Figure 13 Sampling analysis of the seismic migration code**

```
=AFF010I VS FORTRAN VERSION 2 RELEASE 3 INTERACTIVE DEBUG
=AFF011I 5668-806 (C) COPYRIGHT IBM CORP. 1985, 1988
=AFF013I LICENSED MATERIALS-PROPERTY OF IBM
=AFF933I THE AFFON FILE WAS PROCESSED WITH 0 ERRORS
=AFF995I WHERE: PSPIT.15
=* ENDDEBUG SAMPLE(4) CALLED
=AFF112E INTERVAL TIMER WAS RESET BY USER PROGRAM, THUS CANCELLING SAMPLING.
=AFF306I PROGRAM HAS TERMINATED; RC ( 0)
=* ANNOTATE (PSPIT,KPSPIV,FDMGAV,PSPIVT,CITP,CMPTF)  SAMPLING   ALL
=AFF561I ANNOTATING LISTING FOR PROGRAM UNIT "PSPIT"
=AFF561I ANNOTATING LISTING FOR PROGRAM UNIT "KPSPIV"
=AFF561I ANNOTATING LISTING FOR PROGRAM UNIT "CITP"
=AFF224I 746 LINES OF OUTPUT WRITTEN TO AFFPRINT
=* LISTSAMP *  SUMMARY ALL
=AFF550I PROGRAM SAMPLING INTERVAL WAS 4 MS; TOTAL NUMBER OF SAMPLES WAS 23704

=AFF552I SUM OF DIRECT AND CALLED SAMPLES:
=AFF556I PROGRAM UNIT                SAMPLES  %TOTAL
=AFF558I PSPIT                         23704  100.00 ********************
=AFF558I KPSPIV  (program kernel)      21969   92.68 ******************
=AFF558I CITP    (complex interpol.)   10477   44.20 *********
=AFF558I SCFT    (Fourier transform)    8573   36.17 *******
=AFF558I VC#ABS  (complex modulus)      5916   24.96 *****
=* QUIT
```

which can be enhanced by one or more suitable *integral transformations* (for example, Fourier transforms). Parallelism appears at multiple levels which can be exploited simultaneously, rendering possible and useful the application of parallel decomposition techniques such as *domain* decomposition and (at the lowest level of parallelism) *vectorization*. Moreover, the combination of a tuned library (i.e., ESSL), vector (VS FORTRAN) and parallel (PF) compilers, and expanded storage (rendering memory management transparent to the application developer) simplifies the task of tuning scalar seismic codes to the IBM 3090 Vector Multiprocessor, thus rendering possible combined vector/parallel speedups in the range 15–25.

## Acknowledgments

The authors wish to thank Kevin Goin and Alan Karp of the IBM Washington Systems Center for providing the 600E performance data reported in this paper.

## Cited references and note

1. J. Gazdag and P. Sguazzero, "Migration of seismic data," *Proceedings of the IEEE* **72,** 1302–1315 (1984).
2. W. A. Schneider, "The common depth point stack," *Proceedings of the IEEE* **72,** 1238–1254 (1984).
3. C. B. Wason, J. J. Black, and G. A. King, "Seismic modeling and inversion," *Proceedings of the IEEE* **72,** 1385–1393 (1984).
4. B. Fornberg, "The pseudospectral method: Comparisons with finite difference for the elastic wave equation," *Geophysics* **52,** 483–501 (1987).
5. T. M. Taner and F. Koehler, "Velocity spectra—digital computer derivation and application of velocity functions," *Geophysics* **34,** 859–881 (1969).
6. N. S. Neidell and M. T. Taner, "Semblance and other coherency measures for multichannel data," *Geophysics* **36,** 482–497 (1971).
7. M. T. Taner, F. Koehler, and R. E. Sheriff, "Complex seismic trace analysis," *Geophysics* **44,** 1041–1063 (1979).
8. P. Sguazzero and A. Vesnaver, "A comparative analysis of algorithms for stacking velocity estimation," *Deconvolution and Inversion,* M. Bernabini et al., Editors, Blackwell, London (1987), pp. 267–286.
9. J. D. Achenbach, *Wave Propagation in Elastic Solids,* North-Holland Publishing Company, Amsterdam (1975).

10. D. Kosloff, M. Reshef, and D. Loewenthal, "Elastic wave calculations by the Fourier method," *Bulletin of the Seismological Society of America* **74,** 875–899 (1984).
11. R. Clayton and B. Enquist, "Absorbing boundary conditions for acoustic and elastic wave equations," *Bulletin of the Seismological Society of America* **67,** 1529–1540 (1977).
12. J. W. Cooley and J. W. Tukey, "An algorithm for the machine calculation of complex Fourier series," *Mathematics of Computation* **19,** 297–301 (1965).
13. S. G. Tucker, "The IBM 3090 system: An overview," *IBM Systems Journal* **25,** 4–19 (1986).
14. W. Buchholz, "The IBM System/370 vector architecture," *IBM Systems Journal* **25,** 51–62 (1986).
15. R. C. Agarwal and J. W. Cooley, "Fourier transform and convolution subroutines for the IBM 3090 Vector Facility," *IBM Journal of Research and Development* **30,** 145–162 (1986).
16. *Engineering and Scientific Subroutine Library, General Information,* GC23-0182, IBM Corporation; available through IBM branch offices.
17. R. G. Scarborough and H. G. Kolsky, "A vectorizing Fortran compiler," *IBM Journal of Research and Development* **30,** 163–171 (1986).
18. *VS FORTRAN Version 2, General Description,* GC26-4219, IBM Corporation; available through IBM branch offices.
19. M. Kindelan, P. Sguazzero, and A. Kamel, "Elastic modeling with Fourier methods on the IBM 3090 vector multiprocessor," *Scientific Computing on IBM Vector Multiprocessors,* R. Benzi and P. Sguazzero, Editors, IBM European Center for Scientific and Engineering Computing, Rome (1987), pp. 635–674.
20. *Parallel FORTRAN Language and Library Reference,* SC23-0431, IBM Corporation, available through IBM branch offices.
21. M. Kindelan, G. Seriani, and P. Sguazzero, "Pseudo-spectral modeling and its application to amplitude versus angle interpretation," to appear in *Geophysical Prospecting.*
22. Scalar timings reported in Tables 1, 2, and 6 have been obtained using ESSL Release 1 for the scalar FFT computations. In the current Release 2 of ESSL, such scalar computations are performed with a speed increase of about 30 percent.
23. *VS FORTRAN Version 2, Interactive Debug Guide and Reference,* SC26-4223-1, IBM Corporation; available through IBM branch offices.
24. J. Gazdag, "Wave equation migration with the phase shift method," *Geophysics* **43,** 1342–1351 (1978).
25. J. Gazdag and P. Sguazzero, "Migration of seismic data by phase shift plus interpolation," *Geophysics* **49,** 124–131 (1984).
26. A. Dubrulle and J. Gazdag, "Migration by phase shift—An algorithmic description for array processors," *Geophysics* **44,** 1661–1666 (1979).
27. P. S. Schultz and J. W. C. Sherwood, "Depth migration before stack," *Geophysics* **45,** 376–393 (1980).
28. C. C. Hsiung and W. Butscher, "A new numerical seismic 3-D migration model on the Cray X-MP," presented at the *SIAM Conference on Parallel Processing and Scientific Computing,* Norfolk, VA (November 1983).
29. J. Gazdag, G. Radicati, P. Sguazzero, and H. H. Wang, "Seismic migration on the IBM 3090 Vector Facility," *IBM Journal of Research and Development* **30,** 172–183 (1986).

**Aladin Kamel** *IBM Bergen Scientific Center, Allegaten 36, 5007 Bergen, Norway.* Dr. Kamel received his Ph.D. in electrical engineering from the Polytechnic Institute of New York in 1981, and from 1981 to 1983 was an assistant professor of electrical engineering there. He joined IBM in 1983 at the Kuwait Scientific Center, of which he became manager in 1986; he is currently with the Bergen Scientific Center. Dr. Kamel's current research interest is in the area of seismic exploration.

**Manuel Kindelan** *European Center for Scientific and Engineering Computing, Via Giorgione 159, 00147 Rome, Italy.* Dr. Kindelan received his Ph.D. in applied mathematics from the University of California at San Diego in 1975. From 1975 to 1977 he worked with INTA (the Spanish Institute of Aerospace Technique) in Madrid, doing research in combustion modeling. In 1978 he joined the IBM Madrid Scientific Center, working in thermal modeling and image processing. Since 1986 Dr. Kindelan has been on assignment at ECSEC in Rome, where he manages the scientific applications group. His current interests are in the areas of seismic data processing and vector and parallel computing.

**Piero Sguazzero** *European Center for Scientific and Engineering Computing, Via Giorgione 159, 00147 Rome, Italy.* Dr. Sguazzero received his doctorate in mathematics from the University of Trieste in 1969. Since 1970 he has been with the IBM Scientific Centers, first in Venice, then in Palo Alto and in Rome, where he worked in the areas of numerical hydrodynamics and seismic data processing. His current research interest is the development of algorithms and software for vector and parallel machines, with special attention to seismic applications. Dr. Sguazzero is a member of the Society of Exploration Geophysicists.

Reprint Order No. G321-5340.