Stanford Exploration Project, Report 72, December 18, 1997, pages 1–137

Kirchhoff 3-D prestack time migration on the Connection Machine

David Lumley and Biondo Biondi[1]

ABSTRACT

We develop and implement a computer algorithm to image process (migrate) 3-D prestack seismic reflection data on the massively parallel Connection Machine (CM) architecture. Our Kirchhoff 3-D wave-equation time migration achieves our initial goal of 400 Mflop/s on an 8k processor CM, and is scalable to over 3 Gflop/s on a full 64k processor Connection Machine. This is accomplished by parallelizing the migration problem in surface coordinates (x, y), and by serial looping over the pseudodepth coordinate τ, thus optimizing the amount of in-processor parallel computation. Data I/O rates between the front-end and the CM are enhanced by first transposing the data and then calling parallel processor data access subroutines. Internal data movement is enhanced by calling indirect memory addressing and circular data shifting subroutines.

INTRODUCTION

Three-dimensional seismic reflection data are being acquired on an increasingly routine basis. These data offer the potential for extremely high-resolution 3-D images of subsurface geology and subsequent estimation of earth physical properties (Lumley and Beydoun, 1991). Data sets range in size from a few Gigabytes (Gb) to hundreds of Gb. Wave-equation imaging (migration) of these 3-D prestack data requires on the order of hundreds to tens of thousands of Tflop ($10^{14} - 10^{16}$ floating point operations). Evidently the computational challenge is both daunting and exciting.

We approach this computational problem by utilizing a Connection Machine. The CM is a massively parallel computer which can perform billions of floating point operations per second (Hillis, 1985). Since January 1991, Thinking Machines Corporation has provided the Stanford Exploration Project with two CM sequencers, each sequencer consisting of 4096 processors and 32 Mb core memory. Our goal was to develop a 3-D prestack time migration algorithm using the CM resources at SEP, which meets or exceeds some minimum requirements for 3-D prestack image processing.

[1]Thinking Machines Corporation, Menlo Park, CA, and Department of Geophysics, Stanford University.

The algorithm we present is efficient for migrating data sets in which the dimensions of the image space are smaller than, or comparable in size to, the prestack migration aperture. In this case, each input trace needs to be summed into every output trace. A more general algorithm would required a different, more complicated strategy for data communication. As we discuss in the next section, the first field data sets we plan to migrate meet the former requirements and can thus be efficiently migrated with our algorithm.

The content of this paper proceeds as follows. First, we review the minimum computational requirements for 3-D prestack time migration in relation to SEP's computing environment. Next, we discuss the geometry of our 3-D Kirchhoff prestack time migration algorithm and give some examples illuminating algorithm structure using the Fortran 90 parallel language. Finally, we present some timings using a 3-D point-diffractor synthetic seismogram example.

THE COMPUTATIONAL CHALLENGE

The computational problem we are attacking is the prestack time migration of a 3-D seismic reflection data set to produce 3-D images of subsurface geology. We will use the dimensions of a Unocal 3-D survey as an example of the minimum computational requirements. One of us (David) hopes to get a suitable 3-D data set at SEP for a 3-D seismic reservoir characterization project (Lumley, 1991). Whether we obtain the Unocal data or not, using it as a real example to compute minimum requirements is a worthwhile exercise.

The Unocal data consists of 390 shot gathers, 480 traces per shot, of which we are interested in about 3000 samples per trace. This represents a data volume of about 2.25 Gb per 3-D data set. The 3-D volume we wish to image needs to be a reasonable minimum of about 64 by 64 surface positions, by about 1024 pseudodepth (vertical traveltime) samples. This image volume represents about 4 million voxels, or 16 Mb. Using an estimate of 65 floating point operations (flop) per image voxel per trace (the amount required in our algorithm to calculate the migration traveltimes, amplitudes, linearly interpolate trace values, weight and sum into the migration image), a single 3-D prestack time migration of the Unocal data requires a minimum of about 50 Tflop (50×10^{12} floating point operations). This would require a minimum of 60 cpu days on a 10 Mflop/s workstation such as an IBM RS/6000, 30 cpu days on a 20 Mflop/s mini-supercomputer such as a Convex C-2, or 6 cpu days minimum on a 100 Mflop/s supercomputer. Our CM code runs at about 400 Mflop/s on our 8k cpu machine, suggesting a minimum time of about 1.5 cpu days to do a single 3-D prestack migration at SEP using both sequencers. Even more exciting is the potential scalability of our code to a full 64k cpu Connection Machine, which would run at over 3 Gflop/s, suggesting a full 3-D prestack time migration of a real, although admittedly small, 3-D data set in just over 4 cpu hours.

THE CONNECTION MACHINE ALGORITHM

Algorithm geometry

Kirchhoff prestack migration is based on the Kirchhoff boundary integral solution to the scalar wave equation. It is a mathematical statement of Huygens' principle that the wavefield at any interior point in a volume can be reconstructed by a surface integration of the boundary values of the wavefield, weighted by the Green's function impulse response to the wave operator. The original Kirchhoff migration theory was discussed by French (1975) and Schneider (1978) with respect to post-stack data, but is easily generalized to prestack wavefield continuation and imaging (see Lumley, 1989, for example).

As an integral summation, the migration sum can be expressed as a trace-sequential operation; each seismic trace can be migrated independently of the other traces. Because of the Kirchhoff trace-sequential property, each processor in a massively parallel supercomputer, such as the Connection Machine, can migrate a seismic trace in parallel with the other processors. This suggests loading as many traces as possible into the available processors, and migrating them simultaneously by parallel computation. We refer the reader to Biondi (1991) for alternate approaches to wave-equation algorithms on massively parallel computers.

We implement our parallel philosophy as follows. Please refer to Figure ?? for the following discussion. We define an input data array `DATA(nt,nx*ny)`, where `nt` is the number of time samples in the trace, and `nx, ny` are the number of traces in the x- and y-directions respectively. We use the `cmf$ layout` directive to declare `DATA(:serial,:news)`, which instructs the CM compiler that `DATA` will be a 2-D array of dimensions `nt` by `nx*ny`, serial in the first dimension and parallel in the second dimension. Hence, a given seismic trace will reside completely within a single processor, denoted by a column of the `DATA` matrix. Note that `DATA` can be filled with traces from any arbitrary gather or location, since we will also keep the respective trace header coordinates in parallel vectors of the form `Xr(nx*ny)`, laid out as `Xr(:news)`. This frees us to load shot gathers, 3-D CMP binned data, or arbitrarily located traces, into `DATA` until each processor is completely saturated. The better we make simultaneous use of all processors, the more efficient the algorithm.

We define the output image array as `IMG(ntau,nx*ny)`, where `ntau` is the number of pseudodepth samples desired in the migrated output. The `cmf$ layout` instruction declares `IMG(:serial,:news)`, which instructs the CM compiler that `IMG` will be serial in the first dimension and parallel in the second dimension. We achieve a great efficiency by having the parallel dimensions of both `DATA` and `IMG` to be of the same length. This results in a perfect one-to-one overlay of the input and output traces within each processor. This alleviates processor-to-processor communication, which can drastically reduce computational speed, by having all migration calculations done within a single processor, but done in parallel.

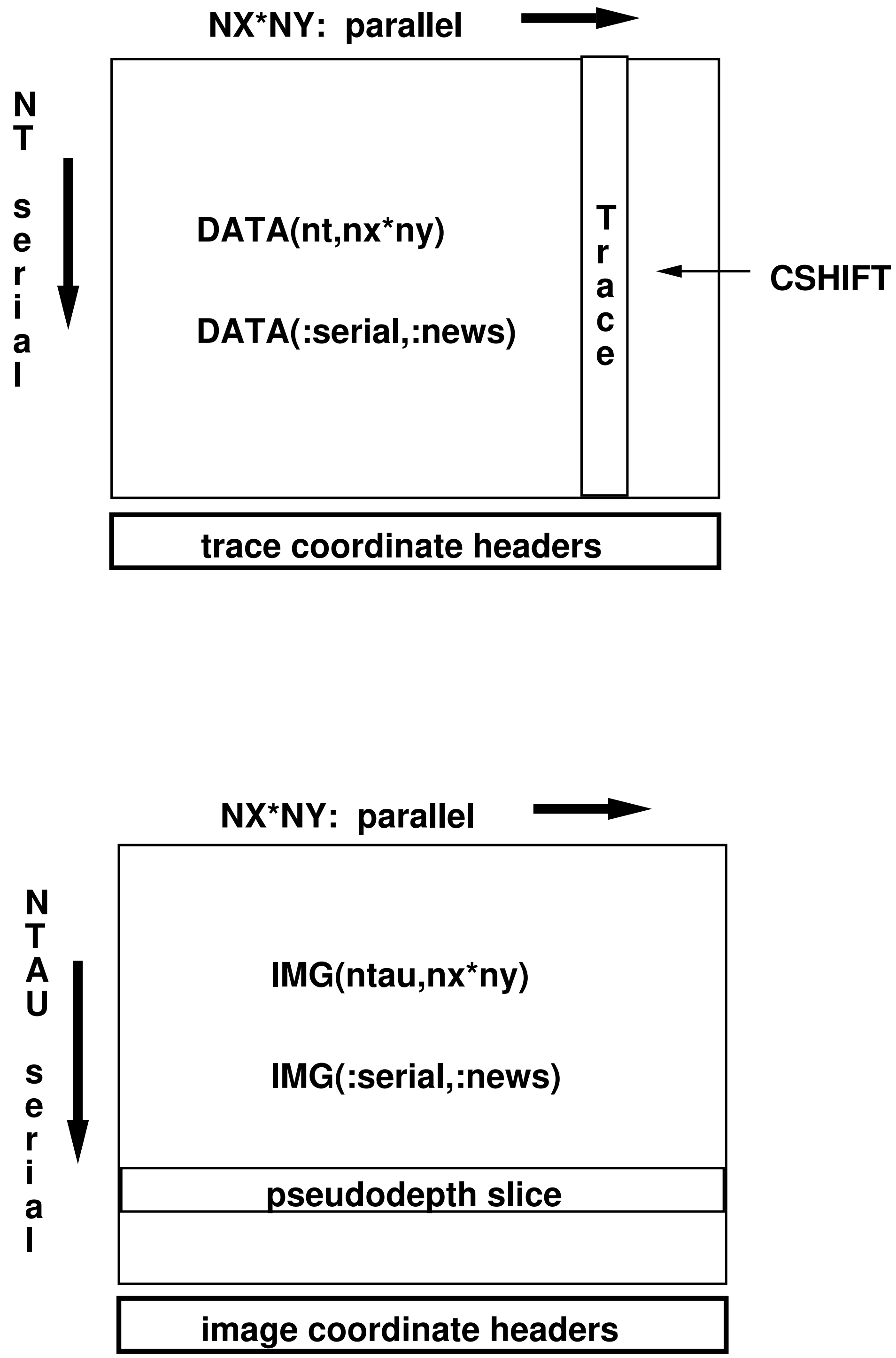

Figure 1: The geometry of the CM Kirchhoff 3-D prestack time migration algorithm. david-geometry [NR]

The CM Fortran 90 code

The basic outline of our CM migration algorithm follows. First, a large swath of the input trace data is loaded into the array `DATA(nt,nx*ny)`, and the trace coordinate headers are loaded into their respective vectors of the form `Xr(nx*ny)`. To enhance I/O speed, we transpose the trace data on the front-end using the SEPLIB routine `Transp`. Then we load the trace data in parallel from the front-end to the CM using the block data transfer subroutine `cmf_fe_array_to_cm` and the temporary front-end vector `fe_temp(nx*ny)`:

```
c       LOAD TRACES INTO DATA...
        do 300 it=1,nt
          irec= (ishot-1)*nt+it
          read(11,rec=irec) (fe_temp(ixy),ixy=1,nxny)
          call cmf_fe_array_to_cm(data(it,:),fe_temp)
 300    continue
```

Next, the required migration surface geometry distances are calculated in parallel, given the source and receiver surface coordinates `(xs,ys)`, `(xr,yr)`, and the image volume surface coordinates `(xi,yi)`:

```
c         CALC DISTANCES...
          ps2(:)= (xs(:)-xi(:))**2+(ys(:)-yi(:))**2
          pr2(:)= (xr(:)-xi(:))**2+(yr(:)-yi(:))**2
```

The heart of the migration operation appears in the next serial loop over `ntau`:

```
c       MAIN LOOP...

        do 2000 itau=1,ntau

c         Calculate migration imaging time...
          tau= 0.5*(itau-1)*dt+t0
          tau2= tau*tau
          timg(:)= sqrt(ps2(:)/vs2(:)+tau2)+sqrt(pr2(:)/vr2(:)+tau2)

c         Calculate cos(theta) and amplitude correction weights...
          cosr(:)= tau/(timg(:)+1.0e-06)
          amp(:)= sqrt(ps2(:)/vs2+tau2)/(sqrt(pr2(:)/vr2+tau2)+1.0e-06)

c         Calculate image time pointers...
          itimg1(:)= int((timg(:)-t0)/dt)+1
          where(itimg1(:).ge.nt)  itimg1(:)= nt-1
          timg1(:)= (itimg1(:)-1)*dt+t0
          itimg2(:)= itimg1(:)+1
```

```
c         Use indirect memory addressing to linearly interpolate trace amps...
          call cmf_aref_1d(value1(:),data(:,:),itimg1(:),.true.)
          call cmf_aref_1d(value2(:),data(:,:),itimg2(:),.true.)
          trcamp(:)= (value1(:)+((value2(:)-value1(:))/dt)*(timg(:)-timg1(:)))

c         Sum weighted trace value into migrated image...
          img(itau,:)= cosr(:)*amp(:)*trcamp(:) + img(itau,:)

 2000   continue
```

This main loop is extremely compact due to the high degree of parallel efficiency. We have left the explicit notation `data(:,:)` to demonstrate that all the calculations are being done in parallel over the `nx*ny` dimension. First, the vertical traveltimes `tau` and migration imaging times `timg(:)` are computed using heterogeneous source and receiver time migration (squared) velocities `vs2(:)` and `vr2(:)`. Next, the obliquity and divergence compensation amplitudes, `cosr(:)` and `amp(:)`, are calculated followed by the imaging time pointers `itimg1(:)` and `itimg2(:)`. Next, the trace values are linearly interpolated in parallel using the fast in-processor addressing subroutine `cmf_aref_1d`. Finally, all migrated values are summed in parallel for one pseudodepth step `tau` into the migrated image `img(:,:)`.

At this point, we have migrated each input trace into a single processor (i.e., for a fixed offset per processor). We need to repeat this operation by migrating the traces into other processors (offsets) until each trace has been migrated into each processor. This is efficiently accomplished by the CM `cshift` subroutine, which shifts each of the traces and its surface coordinate vector to its adjacent processor in parallel:

```
        data(:,:)=      cshift(data(:,:), 2, -1)
        xs(:)=          cshift(xs(:),1,-1)
        ys(:)=          cshift(ys(:),1,-1)
        xr(:)=          cshift(xr(:),1,-1)
        yr(:)=          cshift(yr(:),1,-1)
```

The number of `cshift`s required is equal to the size of the parallel dimension `nx*ny`. Depending on the relative sizes of `data(:,:)` and `img(:,:)`, it may be more efficient to `cshift img(:,:)` and the coordinate vectors `xi(:)` and `yi(:)`, rather than `data, xs, ys, xr` and `yr` as we have done.

The entire process described above is repeated for the number of generic trace gathers which need to be reloaded into core memory until all front-end trace data have been migrated. Finally, the migrated image is written in parallel from the CM to the front-end using the block data transfer subroutine `cmf_fe_array_from_cm` as follows:

```
        do 3000 itau=1,ntau
          call cmf_fe_array_from_cm(fe_temp,data(itau,:))
          write(21,rec=itau) (fe_temp(ixy),ixy=1,nxny)
 3000   continue
```

For efficiency, we have written the data out in transposed format, to be followed by `Transp` to map it back to trace sequential format.

A 3-D SYNTHETIC EXAMPLE

The point diffractor model

Now we present a brief example of the migration algorithm at work. We constructed a simple model of three point diffractors in a constant velocity (2000 m/s) volume to test our algorithm. The dimensions of the model space are $960 \times 960 \times 960$ m^3, or about 1 km^3, sampled at 15m intervals. The three point diffractors exist along the diagonal of the cube at (x, y, z) coordinates (225,225,225)m, (465,465,465)m, and (705,705,705)m. 3-D synthetic seismograms were generated with the Kirchhoff modeling algorithm `Land3d` that one of us (Biondo) has written for the CM. A zero-phase bandpass wavelet was applied with upper and lower frequency cutoffs at 10Hz and 50Hz respectively.

The synthetic seismograms

Figure **??** displays the synthetic data which represent a single 3-D shot gather volume "sliced" vertically within the face of the cube at $(x, y = 465, t)$. If you have the CD–ROM version of this report, you can click your mouse on the figure caption button `movie1`. This action, after waiting for the movie to load, will pop up Dave Nichols' `Animator` window, which will start running a movie of y-slices through the shot record volume. With the cursor in the `Animator` window, you can type `q` to quit, `n` to stop the movie at the next frame, `m` to get the previous frame, and `a` to restart the frame loop if stopped. Also, you can click in the sidebar to reposition the frame within the volume or resize the window in the usual manner.

The shot is located on the surface in the center of the survey, directly above the second diffractor, at (465,465,0)m. There are $64 \times 64 = 4096$ receiver (trace) positions along the surface of the cube, spaced at `dx = dy = 15`m. Each trace represents 1 sec of data sampled at 4 msec, resulting in a total shot gather volume of 1M voxels requiring 4 Mb memory. Each hyperbolic event corresponds to 3-D scattering from one of the point diffractors.

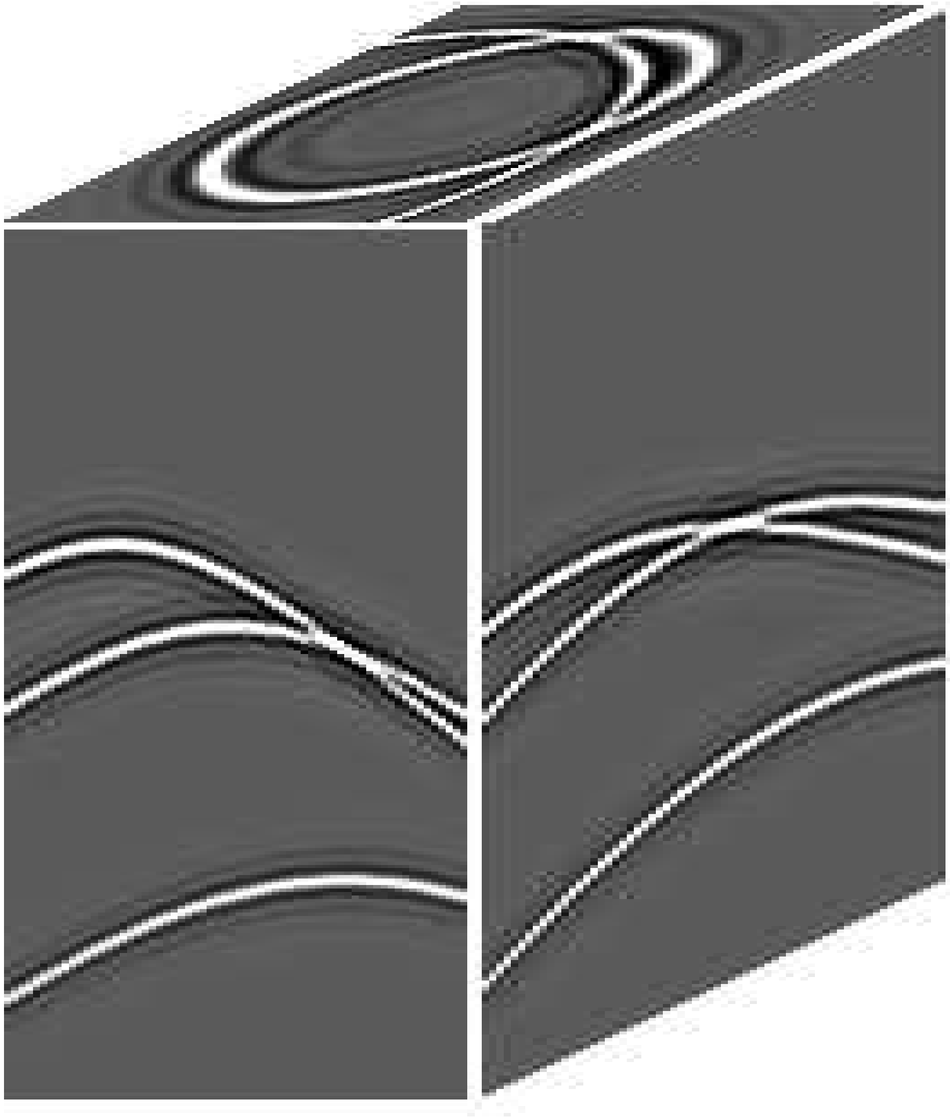

Figure 2: 3-D synthetic point diffractor data. If you have the CD–ROM version of this paper, you can click your mouse on the figure caption button `movie1` to run a movie of y-slices through the shot gather volume. Refer to the text for more details. david-movie1 [NR]

The 3-D prestack migrated image volume

Figure **??** displays the prestack time-migrated data of the single shot gather of Figure **??**. It represents a 3-D image volume sliced vertically within the face of the cube at $(x, y = 465, \tau)$. If you have the CD–ROM version of this report, you can click your mouse on the figure caption button `movie2`. This action, after waiting for the movie to load, will pop up an `Animator` window which will start running a movie of y-slices through the 3-D prestack time-migrated image volume.

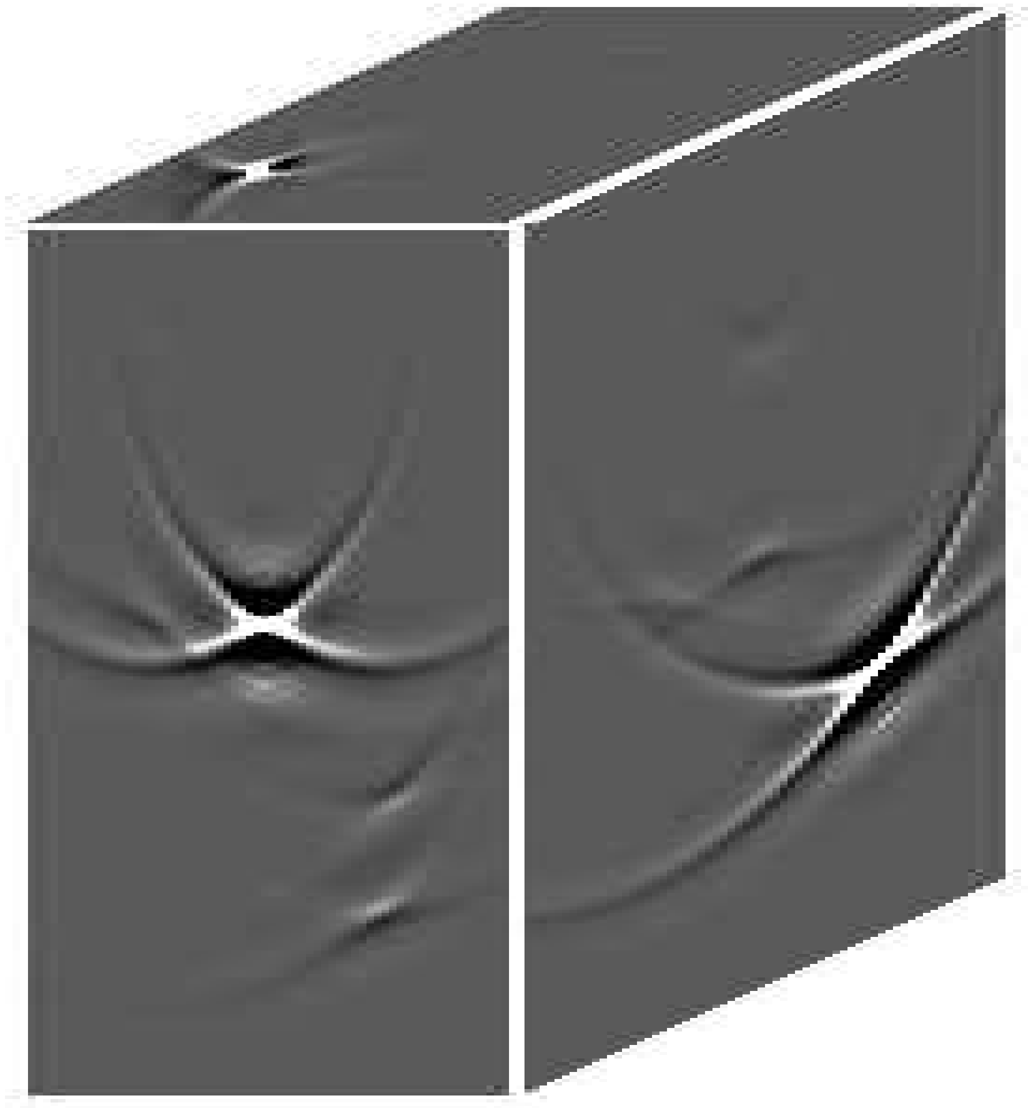

Figure 3: 3-D prestack time-migrated point diffractor data. If you have the CD–ROM version of this paper, you can click your mouse on the figure caption button `movie2` to run a movie of y-slices through the 3-D image volume. Refer to the text for more details. david-movie2 [NR]

Figure **??** shows the focused image of the point diffractor at (465,465,465)m. The image volume is the same size as the input shot gather volume, although the vertical (t, τ) dimensions need not be the same length in general. The image is not a perfect bandpassed spike due to the spatial truncation (finite recording geometry aperture) of the synthetic data, as seen in Figure **??**. The spatial truncations in data space contaminate the image space with "smile" artifacts. In the migration movie, the three spikes fade in and out of focus along the diagonal of the image volume, depending on the y-coordinate of the y-slice (movie frame). This movie is suggestive of the potential for interactive 3-D prestack migration velocity focusing analysis.

CM TIMINGS

We now give some CM timing results of our Kirchhoff 3-D prestack time migration algorithm.

```
CM TIMING:     1 3D shot gather ---> 64x64x256 migration voxels

518 oas> cake cmig
cmattach -w -S0-1 /r1/david/bin/SUN4/cmig.x
Attached to Connection Machine System MAUNALOA
cold booting ... done.
Attached to 8192 processors on sequencers 0-1, microcode version 6002
Paris safety is off.
CM clock speed: 7.00 MHz

Migration complete
Timings from timer # 0
Total Mflops =279345.9 in elapsed time= 654.662178 and CM time= 654.106250
Total elapsed Mflps/s:   8192 proc= 426.702349 --> 64k proc= 3413.618792
Total CM Mflps/s:        8192 proc= 427.065006 --> 64k proc= 3416.520047
MIN migration value -2787.723
MAX migration value  5700.036
 FORTRAN STOP
Detaching... done.
620.4u 32.1s 11:38 93% 0+1832k 615+16io 720pf+0w
```

The above timing is for a single 3-D prestack shot migration described in the point diffractor example. The migration was run using both sequencers for a full 8192 processors with 64 Mb memory. The shot migration required 279 Gflop total, which was performed by the CM in 654 sec (10.9 min), for a total CM computational rate of 427 Mflop/s. The flop count was measured as 65 flops per image voxel per trace, which included `sqrt` operations as 8.4 flop each (Biondi, 1991, p. 69–70). This timing scales to 3.4 Gflop/s on a full 64k processor CM, provided that eight shot gathers (or 8 times as many total trace samples) are available to saturate CM core memory, which is obviously no problem at all with 3-D seismic data! This timing includes all migration calculations, `cshift`ing, internal memory data access and processor-to-processor data movement. However, it does not include external I/O to and from the front-end, which tends to be variable depending on unrelated front-end loads, and would change dramatically if one used CM's data vault or a high-speed disk array. For this 11 cpu minute shot migration, typical elapsed times including front-end I/O average about an extra 20-30%, or 13-14 minutes total elapsed time per 3-D prestack shot migration.

CONCLUSIONS

We have developed and implemented a computer algorithm to perform Kirchhoff 3-D prestack time migration of seismic reflection data on the massively parallel Connection Machine architecture. Our algorithm runs at 427 Mflop/s on an 8k processor CM with 64 Mb core memory, and is scalable to over 3.4 Gflop/s on a full 64k processor Connection Machine. Our migration code allows heterogeneous time migration velocities and arbitrary trace binning, both of which are advantageous in 3-D processing. The present geometry of our algorithm is well-suited to situations in which the image volume dimensions are similar to or smaller than the acquisition aperture. If the image volume is much larger than the survey aperture, the code may require modification to incorporate migration aperture more efficiently. Finally, at these rapid computational speeds, it becomes feasible to contemplate real applications in 3-D prestack migration velocity analysis, migration impedance inversion, and seismic reservoir characterization.

ACKNOWLEDGMENTS

We would like to thank Martin Karrenbach for helpful advice on constructing `cake` files to include the movies into this document, and Dave Nichols for his fine work on the `Animator` software with which the movies are displayed. We also thank Jon Claerbout and the SEP sponsors, especially Thinking Machines Corporation, for supporting massively parallel supercomputing research on the Connection Machine at SEP, without which this paper would not have been feasible.

REFERENCES

Biondi, B., 1991, Wave-equation algorithms on massively parallel computers: Stanford Exploration Project Report, **70**, Stanford University, 59–71.

French, W. S., 1975, Computer migration of oblique seismic reflection profiles: Geophysics, **40**, 961–980.

Hillis, W. D., 1985, The Connection Machine: The MIT Press series in artificial intelligence, Cambridge, Massachusetts.

Lumley, D. E., 1989, Kirchhoff prestack depth migration: imaging conditions and amplitude recovery: 59th Annual International Meeting, Society of Exploration Geophysicists, Expanded Abstracts, 1336–1339.

Lumley, D. E., 1991, Some aspirations in seismic reservoir characterization: Stanford Exploration Project Report, **70**, Stanford University, 161–164.

Lumley, D. E., and Beydoun, W. B., 1991, Elastic parameter estimation by Kirchhoff prestack depth migration/inversion: Stanford Exploration Project Report, **70**, Stanford University, 165–192.

Schneider, W., 1978, Integral formulation for migration in two and three dimensions: Geophysics, **43**, 49–76.