Optimizing the performance of migration in two ways

Jan Thorbecke and A.J Berkhout, Silicon Graphics / Cray Research; Centre for Technical Geoscience, Delft University of Technology*

Summary

In this paper two approaches are used to reduce the computational work of a migration algorithm. The first approach uses a stepwise optimization procedure for the recursive extrapolation routines used in the migration program. And in the second approach the properties of CFP gathers are used to calculate an image of good quality by using less shot record migrations than in a full pre-stack shot record migration.

Introduction

Shot record migration consist of three steps; forward extrapolation of the source wave field, inverse extrapolation of the measured wave field and imaging at all depth levels for all lateral positions. The most computational intensive part for this type of migration is the extrapolation of the wave fields. Optimization of the extrapolation part of the algorithm is described in the first part of this paper. In the second part we will show how we can obtain a good image by using less shot record migrations.

Code optimization for recursive wave field extrapolation

Forward wave field extrapolation can be expressed by the well-known Rayleigh II integral:

$$P^+(\boldsymbol{x}) = \int_{\partial D} G^+(\boldsymbol{x}, \boldsymbol{x}')P^+(\boldsymbol{x}')\mathrm{d}^2\boldsymbol{x}', \tag{1}$$

where $P^+(\boldsymbol{x}')$ represents the measured wave field at the surface ∂D and $G^+(\boldsymbol{x}, \boldsymbol{x}')$ represents the Greens function from a point in the subsurface $\boldsymbol{x}$ to the surface ∂D (1; 7). For inverse wave field extrapolation the complex conjugate of the Greens function is chosen (matched filter approach).

In matrix notation the integral equation (1) can be written as

$$\tilde{\boldsymbol{P}}^+(z_k) = \mathbf{W}^+(z_k, z_0)\tilde{\boldsymbol{P}}^+(z_0), \tag{2}$$

where z_k is the depth level of $\boldsymbol{x}$ and z_0 the surface where the receivers are positioned. Writing this matrix equation in its different components gives:

$$\begin{bmatrix} \tilde{P}^+(0) \\ \vdots \\ \tilde{P}^+(N*\Delta x) \end{bmatrix}(z_k) = \Delta x \begin{bmatrix} W_{11} & \cdots & W_{1N} \\ \vdots & \ddots & \vdots \\ W_{N1} & \cdots & W_{NN} \end{bmatrix} \begin{bmatrix} P^+(\Delta x) \\ \vdots \\ \tilde{P}^+(N*\Delta x) \end{bmatrix}(z_0). \tag{3}$$

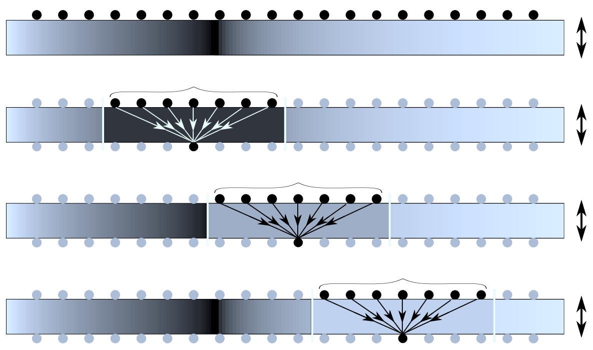
Fig. 1: The principle of recursive extrapolation with spatial convolution operators. If the medium properties change a new operator is read from the operator table. Note that within the length of the operator the medium is assumed to be homogeneous.

In figure 1 this matrix equation is applied in a recursive way; using a small extrapolation step of Δz the wave field is extrapolated from one depth level to another. The assumption that the medium is homogeneous within the operator length can be met if the extrapolation step Δz is chosen small. Note that a smaller extrapolation step also means more computational work.

Equation (3) shows that the most straightforward implementation of the extrapolation algorithm is a (band) matrix vector multiplication. Using the BLAS 2 routine `cgbmv` gives the following program structure for a migration based on recursive extrapolation of the wave fields;

```
for (iom = iomin; iom <= iomax; iom++) {
    get P(z_0) and S(z_0);
    for (d = 1; d <= ndepth; d++) {
        calculate W from operator table
        call cgbmv( P(z_d), W*, P(z_0) );
        call cgbmv( S(z_d), W , S(z_0) );
        calculate Image(z_d);
        copy P(z_d) to P(z_0);
        copy S(z_d) to S(z_0);
    }
}
```

The source (S) and measured (P) wave field are transformed to the frequency domain and for every frequency (iom) the extrapolation is done for all depth steps. In this algorithm the BLAS routines are used for the matrix-vector multiplication. After the image step a copy of the extrapolated wave fields is carried out to save memory since the data at the previous depth level is not needed anymore.

Running this algorithm on one CPU (the MIPS R10000, 195 MHz) of Silicon Graphics Origin 2000 gives for one chosen shot record migration a total wall clock-time of **78.8** seconds.

The performance analysis tools on the Origin 2000 show that 98 % of the code is spend in the extrapolation loop, 60 % is spend in `cgbmv` and 37 % is spend in constructing **W**. The amount of time in constructing the **W** matrix is due to the implementation of a band matrix in BLAS. This implementation requires a non stride-1 copy operation from the operator table to the band-matrix structure.

The performance analysis also showed that for this example 11.6 Gflop where used, resulting in a performance of 147 Mflop/s. Note that the MIPS R10000 is capable of doing an addition and a multiplication in one clock cycle, giving a peak performance of 390 Mflop/s.

Avoiding the BLAS routines means that we have to do the matrix vector multiplication ourselves. The frequency, depth loop, imaging step and copy operation are still the same, but now the `cgbmv` routines are replaced by the loop shown below;

```
index1 = ix + hopl2;
for (j = 0; j < hopl; j++) {
    i1 = index1+j;
    wa += locdat[i1].r*opx[j];
    da += locsrc[i1].r*opx[j];
    i2 = index1-j;
    wa += locdat[i2].r*opx[j];
    da += locsrc[i2].r*opx[j];
}
```

In this loop the floating point operations are always a combined multiplication and addition and can be computed in one clock-cycle. Note that the R10000 is a superscalar processor which makes it possible that the integer operations in the inner loop can be done in the same clock-cycle. The total wall-clock time for the same experiment as before is now reduced to **39.8** seconds.

Compiling with 'cc -S' gives information about software pipelining and is an indication of how the resources of the processor are used.

```
#<swps> Pipelined loop line 171 steady state
#<swps>
#<swps>    16 cycles per iteration
#<swps>    32 flops         (100% of peak)
#<swps>    16 madds         (100% of peak)
#<swps>    10 mem refs      ( 62% of peak)
#<swps>    10 integer ops   ( 31% of peak)
#<swps>    36 instructions  ( 56% of peak)
#<swps>     2 short trip threshold
#<swps>    15 integer registers used.
#<swps>    21 float registers used
```

This compiler option shows that within the inner loop the floating point instructions of the processor are fully used. The number of floating point operations for this implementation is 11.5 Gflop resulting in a performance of 288 Mflop/s

This implementation is bound by the floating point operations and gives a much higher number of Mflop/s compared to the BLAS implementation, but is it the fastest code? Note that the extrapolation operator is symmetrical and that this symmetry is not used very well in this implementation.

Using the symmetry of the operator explicitly gives the following convolution loop;

```
index1 = ix + hopl2;
for (j = 0; j < hopl; j++) {
    i1 = index1+j;
    i2 = index1-j;
    wa += (locdat[i1]+locdat[i2])*opx[j];
    da += (locsrc[i1]+locsrc[i2])*opx[j];
}
```

In this loop the number of multiplication is halved. The total wall clock time for this implementation is **31.6** seconds and 7.3 Gflop are used giving a performance of 222 Mflop/s. Note that the number of Mflop/s is less than the previous implementation, but the code also uses less floating point operations and is therefore faster. Note that cache-trashing (which could occur due to a combined stride 1 (`i1`) and -1 (`i2`) on the same array) is not a problem since `hopl` is usually small (typically 13).

Making this loop parallel on a shared memory computer like the Origin is very easy. Around the frequency loop the compiler directives are inserted and nothing else is changed in the code.

```
#pragma parallel
#pragma shared(image)
#pragma byvalue(hopl, velmod, lenx, iomin, iomax, taper, dom)
#pragma local(iom, tmp1, tmp2, cprev, d)
 { /* start of parallel region */

/* start extrapolation */
#pragma pfor iterate(iom=iomin;iomax;1)

 } /* end of parallel region */
```

The computation time by using more CPU's is shown in the following table;

CPU's	time (s)	parallel time (s)	speedup
1	31.6	30.3	1.0
2	17.0	15.6	1.9
4	9.3	8.0	3.8
8	5.7	4.3	7.0
16	4.3	2.9	10.4

Note that the results in the second column still contain a non-parallel part which takes about 1.3 seconds (compare with third column).

Migration of CFP gathers

Normally in pre-stack migration one uses the source wavelet and the measured response to calculate the migrated image. But in exactly the same way it is possible to use the focusing operator and the CFP gather (5; 2) to generate a migrated CFP gather. The CFP gather is regarded as the response of the focusing operator. Walter Rietveld has used this same approach in his areal shot record migration of plane waves (3; 4).

Different kinds of imaging experiments have been carried out to test and compare the image quality of migration of CFP gathers. The goal of all these experiments is to obtain an image as good as possible with the least amount of computational work. The used migration program is based on the optimized explicit recursive extrapolation algorithm of the previous section and is part of the DELPHI software release.

The migration examples shown in this paper are using the well-known Marmousi data set (6). In figure 2 the top image is calculated by using all the 240 shots available. The depth step in the velocity model is 5 meter and the lateral step is 25 meter. The whole migration of the Marmousi data set takes about one hour on a 8 CPU Origin 2000. Note that this can be made faster by taking larger depth steps and/or less frequencies. But in the examples shown in this paper we will show faster migration results by using less shot record migrations. The middle picture in figure 2 shows the image obtained after using every fourth shot record. In the bottom picture only 6 shot records are used to calculate the image and the used shot records show clearly their footprint on the final image.

The difference between the full migration result and the one using every fourth shot record is not very large but the computation time is 4 times smaller. This result indicates that the redundancy in the data is not used to give a better signal to noise ratio. This is obvious because there is no noise present in the (numerically calculated) data. If we did a similar experiment on data which includes noise, skipping of shot records will make a big difference for the quality of the image. This must be kept in mind

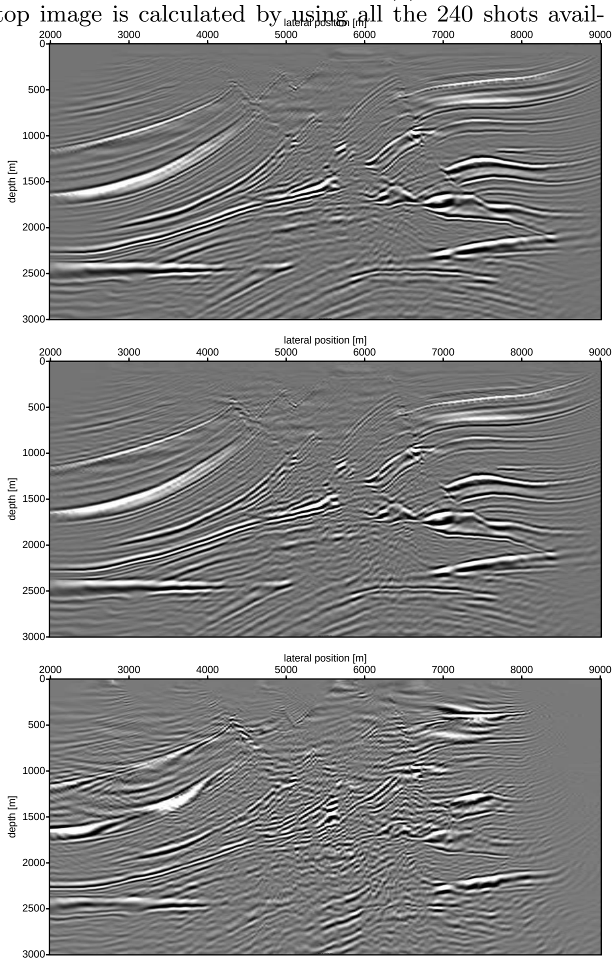

Fig. 2: The top picture shows an image of the Marmousi model using all shot records available, the middle picture shows the image obtained after using every fourth shot record and the bottom picture is obtained by using only 6 shot records.

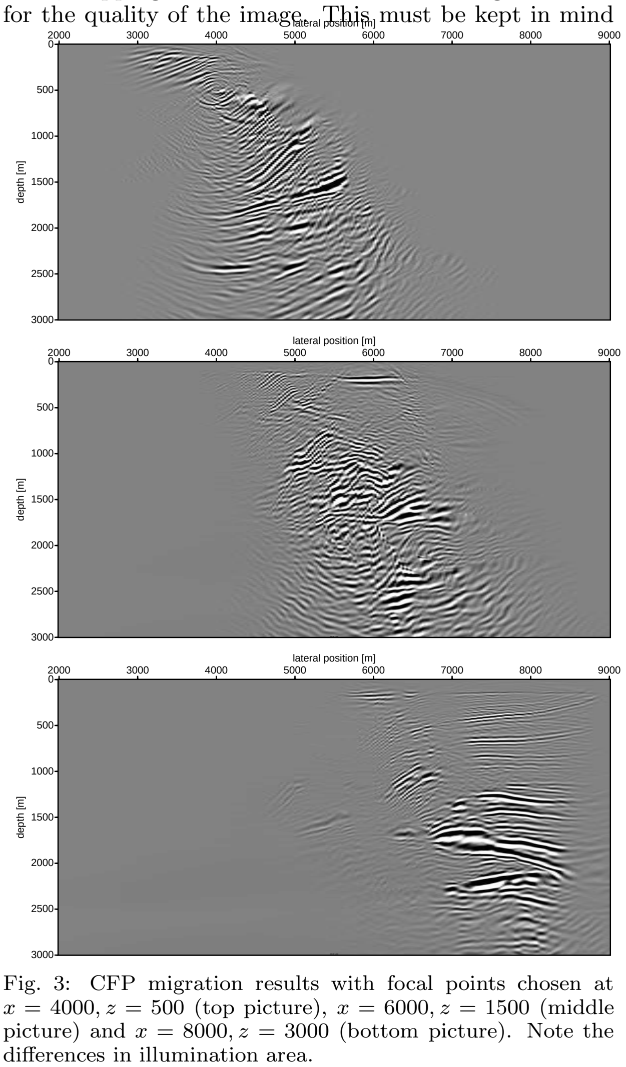

Fig. 3: CFP migration results with focal points chosen at $x = 4000, z = 500$ (top picture), $x = 6000, z = 1500$ (middle picture) and $x = 8000, z = 3000$ (bottom picture). Note the differences in illumination area.

in comparing the different images shown in this paper.

The image results of imaging different CFP gathers, where every CFP gather has a different focal point in the subsurface is shown in figure 3. The image of the CFP gather with its focal point close to the surface (top) shows a wide imaging area at larger depths. The image of the CFP gather with its focal point deep in the surface (bottom) shows a wide imaging area close to the surface and a focused image at the position of the focal point.

Choosing the focal points in a clever way, and adding the CFP migration image results for these focal points together, has the possibility of producing an image with a good quality at low computational cost.

In figure 4 the focal points are chosen at a depth of $z =$ 3000 m and with different lateral distances between the focal points at that depth. Note that with only 29 CFP gathers (the bottom picture) the image quality is already quite good.

Comparing the CFP migration result and the shot record migration result, with the same computational cost of the migration, in figure 5 shows that the CFP migration result has a better/sharper image. In the CFP image it is also observed that the focal points of the used CFP gathers are concentrated at the deeper part of the model, since the shallower parts is not illuminated completely. Note that noise is not included in this experiment and that it is expected that noisy data will favor the CFP results.

Conclusion and discussion

Optimizing the recursive extrapolation routine in the migration algorithm gives a large improvement in the performance and a good parallelization is obtained by using compiler directives. Migration of CFP gathers with their focal points sparsely distributed along a depth level is an efficient way of calculating an image. In the near future we will do a similar analysis for 3D recursive extrapolation and 3D migration and also look at the influence of noise.

References

[1] A. J. Berkhout. *Seismic resolution: resolving power of acoustic echo techniques.* Geophysical Press Ltd., 1984.

[2] A. J. Berkhout. Pushing the limits of seismic imaging: part I. *Geophysics*, 62:in print, 1997.

[3] W. E. A. Rietveld. *Controlled illumination in prestack seismic migration.* PhD thesis, Delft University of Technology, 1995.

[4] W. E. A. Rietveld and A. J. Berkhout. Prestack depth migration by means of controlled illumination. *Geophysics*, 59:801–809, 1994.

[5] J. W. Thorbecke. *Common Focus Point Technology.* PhD thesis, Delft University of Technology, 1997.

[6] R. Versteeg and G. Grau. The Marmousi experience . In *EAEG workshop on practical aspects of seismic data inversion.* Eur. Assoc. Expl. Geophys., 1991.

[7] C. P. A. Wapenaar and A. J. Berkhout. *Elastic wave field extrapolation.* Elsevier, Amsterdam, 1989.

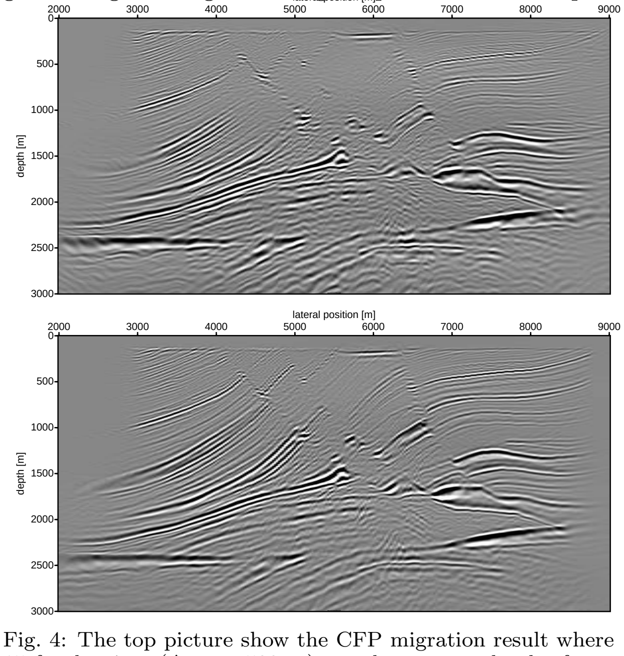

Fig. 4: The top picture show the CFP migration result where 15 focal points ($\Delta x_f = 500$ m) are chosen at a depth of $z =$ 3000 m and in the bottom picture 29 focal points are used ($\Delta x_f = 250$ m).

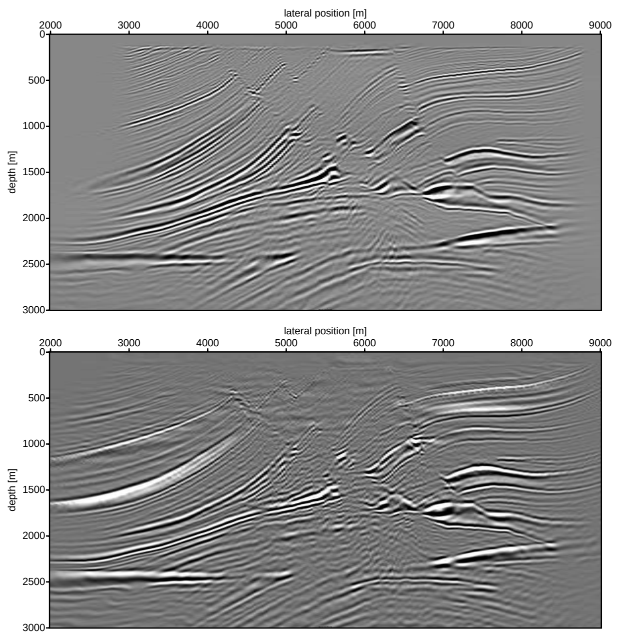

Fig. 5: Comparison of migrated images with the same computational cost. The top picture shows the CFP migration result and the bottom picture shows the shot record migration result.