Parallelisation of Surface-Related Multiple Elimination

G. M. van Waveren
High Performance Computing Centre, Groningen, The Netherlands
<waveren@cs.rug.nl>
and
I.M. Godfrey
Stern Computing Systems, Lyon, France
<igodfrey@acri.fr>

This paper presents the first parallelisation of the surface-related multiple elimination method. This method is used in the seismic industry to eliminate multiple data from recorded seismic data. Both data-parallel and control-parallel implementation schemes are investigated. A realisation of the data-parallel implementation on the SSP/Application, a symmetric multi-processor DEC-Alpha system is described.

1. Introduction

The efficient elimination of multiples from marine seismic data is one of the outstanding problems in geophysics. The marine seismic industry is a multi-million dollar market, and improvement of the accuracy and efficiency of the removal of multiples will lead to cost reduction and shorter turnaround times in this industry. The efficient elimination of multiples requires large amounts of computer time. This prompted us to investigate the applicability of using HPCN technology to increase the efficiency of this type of application.

Multiple data is seismic data which has reflected more than once in the sub-surface. In fig. 1, several types of multiples are shown. We distinguish between *surface-related multiples*, multiples which have at least one bounce against the free surface before being recorded, [(a)-(c) in the figure] and *internal multiples*, which have reflected at least three times before being recorded, without any reflection against the free surface being involved [(d) in the figure].

Many multiple elimination techniques have been developed, as described in Yilmaz (1987). Surface-Related Multiple Elimination (SRME), described by Berkhout (1982) and Verschuur et al. (1991, 1992, 1993), has many advantages over other multiple elimination methods, particularly in the number of types of multiples that are removed. SRME theoretically removes all surface-related multiples, which make up 90 % of the total multiple events. Only internal multiples should remain in the data after SRME. Figures 2 and 3 illustrate the use of SRME.

This article describes the parallelisation of SRME in order to make the method more attractive for routine data processing. It is, to our knowledge, the first parallelisation of this method. We investigated the application of data-parallel and control-parallel techniques to this method. A realisation and timings of a data-parallel implementation of this method on the SSP/Application, a symmetric multi-processor DEC-Alpha system, is described.

In paragraph 2, the theory of SRME is discussed. In paragraph 3, the data-parallel implementation and its realisation and the control-parallel implementation of SRME are described. We conclude with the discussion in paragraph 4.

2. Theory of SRME

Surface-Related Multiple Elimination is a technique used to remove all multiples that are generated through a reflection against the free surface. The free surface is the top-most surface in the seismic experiment, i.e the sea-surface for marine seismics and the earth surface for land seismics. No information about the sub-surface is needed for the elimination of multiples in this technique, only the free surface reflectivity and the source wave field are used to define a surface operator for the actual data. After removing directional source and receiver effects, the observed field at the free surface z_0, expressed in the frequency (ω) domain, is:

$$\overline{\mathbf{P}}_0(z_0) = \overline{\mathbf{P}}(z_0) - A(\omega)\{\mathbf{P}\overline{(z_0)}\}^2 + A^2(\omega)\{\mathbf{P}\overline{(z_0)}\}^3 - \ldots \qquad (1)$$

where $\overline{\mathbf{P}}(z_0)$ is the matrix containing all seismic shot records before multiple elimination. $\mathbf{P}_0(z_0)$ is the matrix containing all seismic shot records after multiple elimination. $A(\omega)$ is the surface operator. For the marine case, reflectivity is approximated by a coefficient -1. The sum of the second, third, etc. terms on the right-hand side of equation (1) constitute the surface-related multiples. These are subtracted from $\overline{\mathbf{P}}(z_0)$, the data with surface-related multiples, in order to obtain $\mathbf{P}_0(z_0)$, the data without surface-related multiples. Note that besides being a function of the depth z, the matrices are also a function of the frequency ω. The set of equations turns out to be independent with respect to the frequency. This characteristic will be exploited in the data-parallel implementation.

The multiple elimination process is converted to a recursive form for the actual computation:

$$\overline{\mathbf{P}}_{i+1}(z_0) = \overline{\mathbf{P}}(z_0)[\mathbf{I} - A(\omega)\mathbf{P}_i\overline{(z_0)}] \qquad (2)$$

for i = 1, 2, ... $\overline{\mathbf{P}_i}(z_0)$ is the i-th order term in equation (1). The recursion is started with $\mathbf{P}_0(z_0) = 0$.

The sequential pseudo-code for this algorithm is shown below.

```
procedure srme
begin
    for i:=1 to N_freq do begin                    (* i is the frequency counter *)
        copy frequency data from disk to central memory
        for j:=1 to N_mult do begin                (* j is the recursion counter *)
            if (j.eq.1) then
                set P_0(z_0) = 0
            else
                get last result P_{j-1}(z_0)
            endif
            calculate P_j(z_0) using recursion relation
            store P_j(z_0) in central memory
        end {j loop}
    end {i loop}
end
```

N_{freq} is the number of frequencies involved in equations (1) and (2). This number is typically in the order of a few thousand. N_{mult} is the number of recursions in equation (2). This number is typically 5 or 6. Within the inner *j*-loop in the pseudo-code, the recursive equation (2) is solved for one frequency. Within the outer *i*-loop of the pseudo-code, this equation is solved for all frequencies.

The above-mentioned surface-related multiple elimination technique is implemented in a seismic data processing sequence in the following way:

1. First the field data are preprocessed, e.g. the direct and surface waves are removed, and the data are interpolated for near-offset traces. The resulting data is stored in a disk file containing traces in the time domain.
2. Monochromatic data matrices are constructed using a Laplace transform. These matrices are stored in a disk file.
3. The source wavelet is estimated based on a selected number of shots.
4. Matrix multiplication and subtraction steps are recursively performed for each order of multiple to be eliminated, as specified in equation (2).
5. The resulting data are inverse-Laplace transformed to the time domain to give the multiple-free pre-stack traces.

3. Parallel Implementation

Step 4 in the above-mentioned sequence is the most time-consuming phase. The aim of this project was to reduce elapsed time by executing the multiple-elimination in parallel.

We considered two options for parallelisation, i.e. data parallelism and control parallelism. These options are illustrated in figure 4.

3.1. Data-Parallel Version

3.1.1 Implementation

Data parallelism can be attained by exploiting the fact that the set of equations (1) are independent with respect to frequency. Thus each frequency is executed independently. The resulting data-parallel implementation is shown in the following pseudo-code:

```
procedure srme
begin
    split the N_freq frequencies into N_proc parts
    for k:=1 to N_proc do begin
        (* N_proc processes are started simultaneously; k is the processor counter *)
        for i:= f_k,first to f_k,last do begin                (* i is the frequency counter *)
            copy frequency data from disk to central memory
            for j:=1 to N_mult do begin                       (* j is the recursion counter *)
                if (j.eq.1) then
                    set P_0(z_0) = 0
                else
                    get last result P_j-1(z_0)
                endif
                calculate P_j(z_0) using recursion relation
                store P_j(z_0) in central memory
            end {j loop}
        end {i loop}
    end {k loop}
end
```

N_{proc} is the number of processors available. N_{freq} and N_{mult} are defined as in the sequential pseudo-code. An extra *k*-loop is introduced in the data-parallel pseudo-code, compared to the sequential pseudo-code. In this loop, N_{proc} parallel tasks are started. Each of these tasks will handle part of the frequency spectrum, i.e. task k will handle the frequency range $f_{k,first}$ to $f_{k,last}$. Thus the *i*-loop will now cover the frequency range of task *k*, instead of the whole frequency range 1 to N_{freq} in the sequential pseudo-code.

3.1.2 Realisation

The above-mentioned data-parallel implementation was realised on the SSP/Application, a symmetric multi-processor DEC-Alpha system. The SRME program is written in standard Fortran 77 and was compiled using full optimisation. Parallelisation was

achieved using the features of the SSP/Soft Macrotasking Library. This library provides a set of routines to create parallel tasks, to synchronise between the tasks, and to lock critical sections - the calling interface is compatible with the Cray Macrotasking library.

In our design, parallel tasks were created just once at the beginning of the multiple-elimination phase, so startup overhead is small. Each task executed the same code and processed an equal fraction of the frequency range. For local arrays, such as the input data matrix array, memory allocation was multiplied by the number of tasks. Access to the data matrix file was locked by the tasks.

This study was performed on the above-mentioned SSP/Application. It was configured with 2 Alpha AXP™ processors (1) operating at 190MHz, sharing 128MB of memory through a 667MB/s bus. Disk space was limited to 2GB. Peak performance of this system is 380Mflops.

3.1.3 Timing Results

Synthetic test cases were used for testing and timing. To verify the parallel implementation we began with a small data case comprising 26 shots, each with 26 records of 1024 samples. The memory requirement of the program is dominated by the output trace array; 5.3MB for this small survey.

We then scaled up the problem to a more realistic size; 301 shots, 121 records, 2048 samples. To process the whole of this survey at one time would exceed the physical memory of the benchmark system. To avoid paging, therefore, we recorded times for subsets of the whole survey, based on two frequency ranges (6 and 12) and 3 groups of shots (31, 51, and 80). For each test, the elapsed time of the multiple elimination section was recorded for the original serial version and the new parallel version. The tests were run more than once, and the speedup results shown in figure 5 are the ratio of the minimum serial time to the minimum parallel time.

3.2. Control-Parallel Version

3.2.1 Implementation

Control parallelism can be attained by splitting the calculation of each multiple recursion in equation (2) over the different processors. The resulting control-parallel pseudo-code is shown below:

---

(1) The maximum number of processors in this system is 4.

```
procedure srme
begin
    for j:=1 to N_mult do begin          (* j is the recursion and the processor counter *)
        for i:=1 to N_freq do begin                    (* i is the frequency counter *)
        copy frequency data from disk to central memory
            if (j.eq.1) then
                set P_0(z_0) = 0
            else
                receive last result P_{j-1}(z_0) from processor (j-1)
            endif
            calculate P_j(z_0) using recursion relation
            send P_j(z_0) to processor (j+1)
        end {j loop}
    end {i loop}
end
```

In this control-parallel implementation, each recursion is calculated on a different processor. The j-loop variable is now both a recursion and a processor counter. Processor j will now receive recursion result $\mathbf{P}_{j-1}(z_0)$ from processor (j-1), calculate recursion result $\mathbf{P}_j(z_0)$ and send this result to processor (j+1).

3.2.2 Realisation

This control-parallel implementation is currently being realised on the 16-processor TMC CM-5 of the High Performance Computing Centre of the University of Groningen.

4. Discussion

This paper summarises the results of the first known parallel implementation of the SRME program from the Delphi Software Release. A data-parallel decomposition over the frequencies was used, and implemented using the Macrotasking Library in the SSP/Soft environment. Preliminary results demonstrate a maximum real speedup of 1.87 on the multiple elimination phase of the complete process.

Work is continuing to increase the timing sample and to compare different parallelisation techniques, such as the control parallelism mentioned earlier.

References

Berkhout, A.J., 1982, Seismic Migration: Imaging of acoustic energy by wave field extrapolation. A. Theoretical aspects, second edition: Elsevier Science Publ. Co., Inc., p.211-218.

Verschuur, D.J, Berkhout, A.J., and Wapenaar, C.P.A., 1992, Adaptive surface-related multiple elimination, Geophysics, Volume 57, p.1166-1177.

Verschuur, D.J., and Berkhout, A.J., 1993, Integrated Approach to Multiple Elimination and Wavelet Estimation, 55th EAEG meeting, exp. abstracts, abstract B025.

Verschuur, D.J., 1991, Surface-Related Multiple Elimination, an inversion approach, PhD Thesis, Delft University of Technology.

Yilmaz, Öz, 1987, Seismic Data Processing, Society of Exploration Geophysicists, Tulsa, OK, USA.

Acknowledgements

We wish to thank prof. dr. A. J. Berkhout and the Delphi group of the Delft University of Technology for the privilege of using the Delphi software release for the purpose of parallelisation. Dr. G. M. van Waveren wishes to thank Stern Computing Systems for the hospitality given to him.