Efficient Parallel Implementation of Coherent Stacking Algorithms in Seismic Data Processing

Maxim Gorodnichev[1,2,3], Anton Duchkov[2,4], and Alexander Kupchishin[3,4]

[1] Institute of Computational Mathematics and
Mathematical Geophysics SB RAS, Novosibirsk, Russia,
[2] Novosibirsk State University, Russia,
[3] Novosibirsk State Technical University, Russia,
[4] Chinakal Institute of Mining SB RAS, Novosibirsk, Russia
maxim@ssd.sscc.ru, DuchkovAA@ipgg.sbras.ru, and vsegdatrezv@mail.ru

Abstract. We discuss efficient parallel implementation of coherent stacking algorithms which form basis for a class of seismic processing procedures. In detail we address the problem of processing data of microseismic monitoring for localizing seismic events in space and time. Continuous data recording by seismic array quickly generates terabytes of data to be processed in a timely manner, including real-time analysis in some cases. Thus processing requires efficient parallel implementation with a special attention to data partitioning between nodes, and using computations to mask data reading from disk. Efforts were taken to minimize cache misses and vectorize loops. Sequential version of the code demonstrates 8x speed up compared to a naive implementation of the algorithm; parallel code scales almost linearly.

Keywords: coherent stacking, microseismic monitoring, parallel computing, Xeon Phi

1 Introduction

Seismic data processing requires high performance computing due to extremely large volumes of acquired data [1]. In particular, coherent stacking (summation) operation is used in a number of processing procedures [2]. These include velocity analysis, stacking, Kirchhoff-type migration in time and depth, location of microseismic events, and many others.

Microseismic monitoring is widely used to monitor hydraulic fracturing, subsidence related to depletion, cap rock integrity, mapping fluid migration, detection of casing failure [3]. Such monitoring requires continuous recording of seismicity by a network of stations. This results in large data volumes which preferably need to be processed in a real time manner.

Microseismic data processing results in localisation of seismic events in space and time.

The result of the work is parallel implementation of the coherent stacking method for the analysis of large microseismic data volumes (terabytes) optimised for modern high performance computational platforms.

2 Coherent Summation Method

The method takes microseismic monitoring data and the parameters of the volume being studied as an input and solves the inverse problem discovering spatio-temporal location of a series of seismic events.

Microseismic monitoring data are represented as a set of traces, were a trace $G(t, D_k), t = 0, .., T-1$ — is a signal sampled by seismometer $D_k, k = 0..(Q-1)$. A surface location of the seismometer D_k is given as (x_k, y_k).

Build a mesh over the volume being studied. Denote the set of mesh nodes as U. Each node $u \in U$ is then tested as a possible event location.

For each $k = 0..(Q-1)$ and $u \in U$, we compute the time t_{uk} required for a wave front to reach the seismometer D_k from the location u. Let us denote $Tmin_u = \min\limits_k t_{uk}$, $Tmax_u = \max\limits_k t_{uk}$, and $Td_u = Tmax_u - Tmin_u$. The length of the signal will be denoted as L.

The coherent summation algorithm can be then defined as follows:

1: **for** $u \in U$ **do**
2: **for** $t \in 0..(T - Td_u - L)$ **do** $S_{u,t} = \sum\limits_{l=0}^{L-1} \sum\limits_{k=0}^{Q-1} G(t_{uk} - Tmin_u + t + l, D_k)$
3: **end for**
4: **end for**
5: find u_{max} and t_{max} such that $S_{u_{max},t_{max}} = \max\limits_{u,t} S_{u,t}$

If there is such a mesh node u_e that corresponds to the location of the seismic event, this u_e is expected to be found as u_e=u_{max}, and t_{max} will be the time of the event.

If we are interested to find a series of events, we will choose several S_{ut} that exceed a certain treshold value.

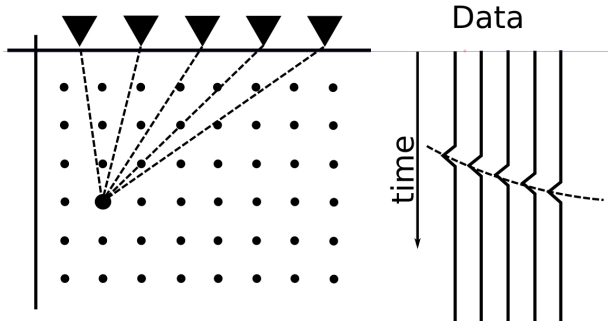

Fig. 1. Coherent summation. Seismic sensors are shown as triangles. The grid represents the set of probe points U. Summation of seismic data is done along the dotted line for the given point.

In practice, there can be variations to this method, but the general scheme remains and it is this simple. However, straightforward implementation of this scheme results in poor computational performance. The following sections describe the steps to efficient implementation of this method.

3 Mesh Refinement

The process starts [5] with a coarse mesh U^0 covering the whole volume V^0 being studied. The coherent summation method is then applied to this mesh and, in this way, we find a mesh node u^0_{max} that is close to the seismic event. Then we define a volume $V^1 \in V^0$ with a center in u^0_{max} and build a successive finer mesh U^1 in V^1. These steps are repeated until the volume V^j is small enough thus determing an event location with appropriate accuracy.

4 Hiding Disk Access Operations Behind Computation

Typical volumes of seismic data can be measured in hundreds of gigabytes or terabytes. Thus, it is necessary for the program to access low performance external memory (local disks, networked filesystems). In order to confront this issue, the double buffering approach is taken. The new portion of data is loaded from external memory asynchronously by a special thread while the previously loaded portion is being processed.

5 Elimination of Recomputing

One can notice that $S_{u,t+1} = S_{u,t} - \sum_{k=0}^{Q-1} G(t_{ku} - Tmin_u + t, D_k) + \sum_{k=0}^{Q-1} G(t_{ku} - Tmin_u + t + L, D_k)$. Thus, naive implementations of the the method will require multiple computation of the same values.

Elimination of recomputing made computing time independent of the signal length L (Fig. 2).

6 Loop Vectorization and Avoiding Cache Misses

Processor caches and hardware prefetching work efficiently when an application traverse data in memory in consequitive way [6]. In our case, the data is stored in memory in such a way that $G(t+1, D_k)$ follows $G(t, D_k)$ immediately. So, in order to force consequitive memory traversal, the loops should be organized in the following order:

1: **for** $k = 0..(Q-1)$ **do**
2: **for** $t = 0..(T_{end} - Td_u - L)$ **do**
3: $S_{u,t} = S_{u,t} + \sum_{l=0}^{L-1} G(t_{k,u} - Tmin_u + t + l, D_k)$
4: **end for**
5: **end for**

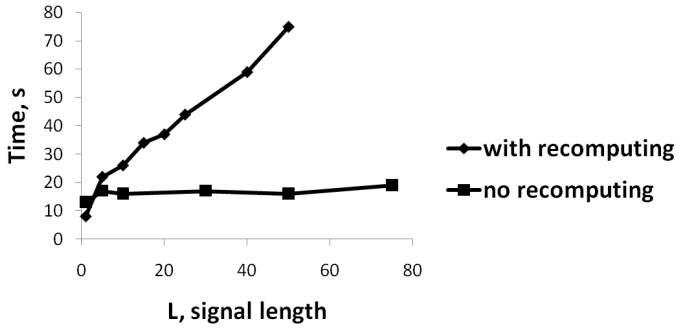

Fig. 2. Computation time, s, dependence on signal length. U: a grid of 151515 points, Q=100,T=2000.

It is important to vectorize the inner loop manually or write it in a form that would permit a compiler to do this job.

Changing the loop order to the correct one reduces the time of computation by 3.8 times on Intel® Xeon® E5-2690 and by 9.8 times on Intel® Xeon Phi™ 7110X. Vectorization of this loop speeded-up the sequential program by a factor of 2 for E5-2690 and by a factor of 2.8 for 7110X. The Intel® C++ Compiler ver. 14.0.1 20131008 with -O3 was used for all the tests.

7 Parallel Implementation

The method can be easily parallelized. Within a computing node and the shared memory processing paradigm, the computation can be divided among computing cores by partitioning the nodes of a mesh U. The nodes also can be processed independently of each other, thus the only problem with parallel implementation can be a memory bottleneck. This problem can be addressed by improving data access locality (see Section 6). The performance results of a shared memory parallel program obtained on Intel®Xeon Phi™7110 are presented in the Table 1.

Table 1. Computation time, s, of shared memory parallel program execution for different number of threads on Intel® Xeon® E5-2690 and on Intel® Xeon Phi™ 7110X.

Number of OMP threads	1	4	8	16	30	61	244
Xeon E5-2690	28	7	4	3			
Xeon Phi 7110X	779	193			27	21	12

In order to distribute the work between the nodes of a supercomputer we partition seismic data in chunks by time. Most of the processing can be done independently and the nodes will have to compare their maximal sums only

in the end of computation. The program was implemented with Intel® MPI Library. The performance results are presented in Fig. 3 and Fig. 4.

The problem characteristics for all the tests in this Sections are: U is a grid of 404010 points, Q=2000, and T=1200.

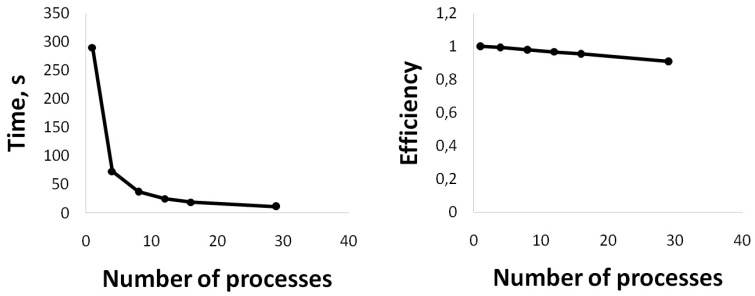

Fig. 3. Computation time, s, and efficiency of the program execution on the Intel®Xeon Phi™ 7110 co-processors depending on the number of co-processors used. Only one co-processor per computing node was used and 244 OMP threads were running on each co-processor.

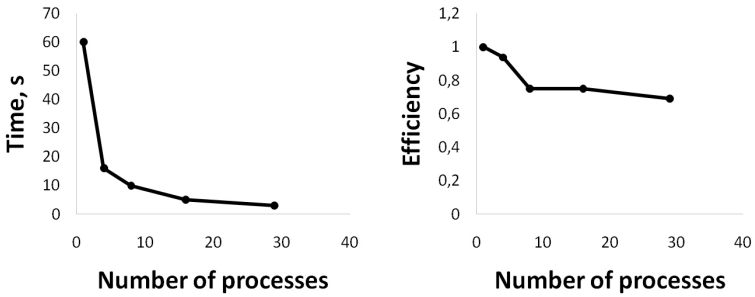

Fig. 4. Computation time, s, and efficiency of the program execution on the computing nodes with 2 Intel® Xeon® E5-2690 each depending on the number of computing nodes. Each node was running 16 OMP threads.

8 Conclusion

A number of optimization steps to naive implementation of coherent summation algorithm were taken in order to obtain efficient parallel program. The overal speed-up of the sequential computations is over 7x on Intel® Xeon® E5-2690

and over 26x on Intel® Xeon Phi™ 7110X. The efficiency characteristics measured on Intel(R) Core(TM) i5-3550 processor are as following: CPI (clocks per instruction) = 0.55, LLC (last-level cache) miss = 0.026 (a ratio of cycles with outstanding LLC misses to all cycles). A parallel program was implemented with OpenMP and MPI. The program is capable of processing large data sets on computational clusters and scales well over distributed memory computing nodes. Memory subsystem seems to be a bottleneck for a shared-memory program scalability and more efforts on memory access optimization are required.

The algorithm is to be included into different procedures for seismic data processing. It should be enhanced by introduction of workload balancing for heterogenous computing systems so that it could evenly distribute workload among all the available processing units, including CPU cores and co-processors. We also plan to integrate it into a popular open-source seismic processing package Madagascar [7].

Acknowledgements. Work was supported by the Russian Ministry of Education and Science (project # RFMEFI60414X0047).

References

1. Camp, W., Thierry, P.: Trends for high-performance scientific computing. The Leading Edge 29(1), 44–47 (2010)
2. Rückemann, C.: Comparison of stacking methods regarding processing and computing of geoscientific depth data. GEOProcessing 7(27), 35–40 (2012), http://dx.doi.org/10.12988/ces.2014.410187
3. Warpinski, N.: Microseismic monitoring: Inside and out. Journal of Petroleum Technology 61(11), 80–85 (2009)
4. Chambers, K., Kendall, J., Brandsberg-Dahl, S., Rueda, J.: Testing the ability of surface arrays to monitor microseismic activity. Journal of Petroleum Technology 58(5), 821–830 (2010)
5. Lemeshko, B.: Optimization Methods. NSTU Publishing, Novosibirsk, Russia (2009) (in Russian).
6. Cherkasov, A., Gorodnichev, M., Kireev, S., Markova, V., Artyom, M.: On optimization of numerical simulation programs. In: Science and Technology, 2005. KORUS 2005. Proceedings. The 9th Russian-Korean International Symposium on Science and Technology. pp. 584–589. IEEE (2005)
7. Fomel, S., Sava, P., Vlad, I., Liu, Y., Bashkardin, V.: Madagascar: open-source software project for multidimensional data analysis and reproducible computational experiments. Journal of Open Research Software 1(1), e8 (2013)