Paralleling of calculations and vectorization of processes in digital treatment of seismic signals by cubic spline

H N Zayniddinov and O U Mallayev

Tashkent university of Information technologies named after Muhammad al-Khwarizmi (TUIT), Amir Temur Street, 108, Tashkent, 100200, Uzbekistan

1 E-mail: info-oybek@rambler.ru

Abstract. In this work the possibility of parallelization of calculations when smoothing data using the cubic spline method using examples of seismic signal processing has been researched. The main indicators of efficiency and acceleration of parallel and sequential algorithms were calculated.

1. Introduction

Parallel computing is one of the most relevant areas of computational science in the last decade. The relevance of this field is associated primarily with the rapid development of numerical modeling. Numerical simulation is an intermediate element between analytical study methods and physical experiments. The increase in the number of tasks for which parallel computations are necessary is due to:

- The opportunity to study phenomena that are difficult, for some reason, to study experimentally.
- The necessity to manage complex industrial and technological processes in real time [1, 2].
- Increase in the number of tasks for the solution of which it is necessary to process huge amounts of information.

The computational algorithm for solving the wave equation at each node of the grid performs, using a given formula, the calculation of the new value \( y_{i}^{n+1} \). Programmatically this is implemented through a loop through the elements of the array. Such loops can run slowly in Python (and similar interpreted languages such as R and MATLAB). One of the methods to accelerate loops is to perform operations on whole arrays instead of working with one element of the array at the current time. This is called vectorization or vector computing. Operations on whole arrays are possible if the calculations affecting each element are independent of other elements. Vectorization not only speeds up the work of the program on serial computers, but also makes the program easier to use parallel computing.

In addition, the processors are approaching the limit of the clock frequency. Historically, Moore's law is fulfilled: every 18 months, the performance of computing systems doubles. Until the 21st century, the production of processes inclined because of a high increase of clock frequency. The main brake here was energy consumption: with an increase in the clock frequency by 20%, energy consumption increased by 73% [3]. Therefore, it was decided to go on the other way. Processor
manufacturers began to pay more attention to the implementation of parallelism, rather than the clock frequency of processors. This is achieved through the implementation of a multi-core architecture. Placing on one chip of the processor several simplified processors whose clock frequency was lowered and the functions of the ALD were simplified. Almost all modern devices have a multiprocessor architecture.

Due to the fact that multiprocessor systems, such as supercomputers, have become widespread, and there are many tasks that require large computing powers, the role of parallel programming methods has increased.

Splines have numerous applications both in mathematical theory and in applied mathematics (in particular, in various computational programs). In particular, the splines of two variables are intensively used to define surfaces in various computer simulation systems. The splines of the two arguments are called bi-splines (for example, a B cubic spline), which are two-dimensional splines that model surfaces. They are often confused with B-splines (basic splines), which are one-dimensional and, in a linear combination, make up curves - a framework for “stretching” surfaces. It is also possible to compose a three-dimensional construction from the basis splines for modeling volumetric bodies [7, 8, 9].

Any spline \( S_m(x) \) of degree \( m \) of defect 1 interpolating a given function \( f(x) \) can be uniquely represented by B-splines as a sum [2, 5, 10]:

\[
f(x) \equiv S_m(x) = \sum_{i=1}^{m+1} b_i \cdot B_i(x); \quad a \leq x \leq b,
\]

where \( b_i \) – coefficients, \( B(x) \) – basis spline.

In occasion if basis cubic splines are used, its value will be calculated by formula:

\[
B_3 = \begin{cases} 
0, & x \geq 2 \\
(2 - x)^3 / 6, & 1 \leq x \leq 2, \\
1/6(1 + 3(1-x) + 3(1-x)^2 - 3(1-x)^3), & 0 \leq x \leq 1.
\end{cases}
\]

Coefficients can be determined by formula:

\[
b_i(1/6)(-f_{i-1} + 8f_i - f_{i+1});
\]

According to formula (1), the value of the interpolating function at an arbitrary point of a given interval is determined by the values of only \( m + 1 \) terms - the pair products of the basis functions by constant coefficients. For example, cubic B-splines require four basic terms. The value of the function is calculated by the formula:

\[
f(x) \equiv S_3(x) = b_{-1}B_{-1}(x) + b_0B_0(x) + b_1B_1(x) + b_2B_2(x); \text{ at } x \in [0,1]
\]

The remaining basic splines on this subinterval are equal to zero and, therefore, do not participate in the sum formation. If we use one main basis spline and use the variable \( j \) to specify the addresses of different parts of the main spline, then equation (4) takes the form:

\[
S_3[i] = (b[i-1]B[j + 30]) + (b[i]B[j + 20]) + (b[i + 1]B[j + 10]) + (b[i + 2]B[j]);
\]
Figure 1. Programming mode for conventional scalar processors.

In many computational programs, the same operation is applied to a whole data set: to an array, or to some parts of an array. In this case, there is a repetitive data access model, as well as a repetitive operation that can be parallel performed. For example, the following code snippet shows an example of such a repeating model. This example gives the addition of two vectors.

A usual scalar processor will execute this program as follows: first, the numbers A (0) and B (0) are loaded into the register, then they are added up and the result is written to C (0), then A is increased by 1, elements A1 and B1 are loaded into the registers, and perform the operation of addition and so on. This is presented in figure 1.

Figure 2. Programming mode vector processors.

If the processor is a vector, then there are vector registers in it that allow you to store a vector of values, and vector instructions are implemented in the processor. This is presented in figure 2. If we use vector instructions, the processor will load several values of vector A and B into registers at once. Suppose 4 elements. Then in the registers will be the elements: A (0), A (1), A (2), A (3); and respectively for the vector B: B (0), B (1), B (2) and B (3). And for all 4 pairs, the addition operation will be executed simultaneously, and the result will be written in C (0), C (1), C (2) and C (3). The advantage of vectorization is that the execution time of the vector operation is the same as the scalar, but more useful work is performed. One vector command is recognized, decoded and executed faster than several scalar actions that perform the same.

The algorithm for vector computation can be divided into two parts:

1. In the first part, the elements of the arrays Lj, Kj, Pj i Tj from formula (6) are vectorized;
2. In the second part, the sum of the elements of the arrays Lj, Kj, Pj and Tj is vectorized.

As a result, we obtain the formula (6):
To develop a parallel algorithm, select the threads [6, 7]. To perform four parallel multiplications, we select \( m = 4 \) threads. As a result, we obtain the formula (6), where \( L_j, K_j, P_j \) and \( T_j \) are the multiplication flows of the matrix by the vector.

The calculation algorithm can be divided into two parts:

- Calculation of coefficients at interpolation nodes.
- Calculate spline values at a given point.

The first stage is not so laborious as compared to the second. The second stage takes a much larger part of the computational time, so the execution of the second stage of the algorithm is distributed between the processes.

The flowchart of the parallel computing algorithm with the allocation of threads using the OpenMP package is shown in figure 3. First of all, the user enters input node values. Then, using special MPI directives, a special function, the PFX and TPL library creates a parallel section that will be executed on several processes, the number of which is specified by the user when the application is started. After that, the data is distributed between processes. An example of the distribution of an array consisting of 100 elements is shown in figure 3.

After the data has been distributed, each process starts the calculation of values between the input interpolation nodes. For this, on each of the nodes a sequential function is invoked that implements the cubic spline method. After performing the calculations, each of the processes makes a barrier blocking so that all the processes finish the calculation, and after that the results of the calculations are compiled from all the processes into one array on one process.

After all the data are collected in one process, the algorithm is completed. The is algorithm presented in figure 4. The results of running programs on different data sets are shown in table 1.

![Figure 3](image-url)  
Figure 3. An example of the distribution of data between processes.

| Table 1. Result of execution on various data sets. |
|---------------|---------------|---------------|---------------|
| Dimension     | In series (1 process), sec | 2 processes, sec | 4 processes, sec |
| 1024          | 0.00128654    | 0.000809977    | 0.000403943    |
| 263000        | 0.0115795     | 0.00671721     | 0.00318956     |
| 33555000      | 0.98814       | 0.528883       | 0.261672       |
Acceleration when executing a program on 2 and 4 processes as compared to sequential execution is presented in table 2.

**Table 2. Program acceleration rates.**

| Dimension  | Acceleration compared to sequential execution |
|------------|-----------------------------------------------|
|            | 2 processes | 4 processes |
| 1024       | 1,588       | 3,1849      |
| 263000     | 1,723       | 3,6304      |
| 33555000   | 1,872       | 3,776       |
| 134218000  | 1,907       | 3,8654      |
Figure 4. A block diagram of a parallel computation vectorization algorithm with threading.

Table 3. The effectiveness of the program compared with the sequential execution.

| Dimension | Efficiency compared to sequential execution |
|-----------|--------------------------------------------|
|           | 2 processes | 4 processes |
| 1024      | 0.794       | 0.796225    |
| 263000    | 0.8615      | 0.9076      |
| 33555000  | 0.936       | 0.944       |
| 134218000 | 0.9535      | 0.96635     |
Table 4. Comparative data obtained as a result of seismic signal processing in a dual core processor.

| The number of samples of the input signal N | Vectorization (sec) | Parallelization (sec) | Acceleration coefficient |
|---------------------------------------------|---------------------|-----------------------|-------------------------|
| 1024                                        | 0.00001             | 0.00002               | 2                       |
| 2048                                        | 0.000011            | 0.00003               | 2.7                     |
| 4096                                        | 0.000073            | 0.000191              | 2.6                     |
| 16000                                       | 0.000023            | 0.000078              | 3.4                     |

In accordance with the data of table 4, for the vectorization of processes of parallel processing of a seismic signal with the number of samples N = 1024, it took 0.02x10^4 seconds in a conventional parallelized compiler, 0.01x10^4 seconds was required after its vectorization. With an increase in the number of input samples N, the parallel part of the processing increases and the acceleration coefficient increases. So, at N = 16000, for the vectorization of processes of parallel processing of the seismic signal, it took 0.078x10^-4 seconds in a conventional parallel compiler, 0.023x10^-4 seconds in vectorization.

2. Conclusion

Thus, the requirements of high performance computing systems used in the tasks of processing and recovering seismic data can be satisfied both by developing new methods, parallel algorithms and processing programs, and using multi-core parallel computing tools. Practical implementation of spline methods in the form of vectorization of parallel processing of seismic data using OpenMP, MPI technology and multi-core processor architecture in the prediction, interpolation, smoothing and identification, recovery and reduction of redundancy procedures allows for a general increase in system efficiency by increasing processing speed data at the established accuracy rates.

References
[1] Zavyalov Yu S, Leus V A and Skorospelov V A 1985 Splines in engineering geometry (Moscow: Machinebuilding)
[2] Voevodin V V and Voevodin VI V 2012 Parallel calculations (St. Petersburg: BHV-Petersburg)
[3] Bogachev K Yu 2013 Basics of parallel programming (Moscow: “Binom The laboratory of knowledge)
[4] Zayniddinov H N, Tozhboev G O and Mallaev O U 2018 Parallel seismic signal processing algorithms on multi-core processors Automation and software engineering 1(23) 89-95
[5] Gerbert R, Bik A, Smith K and Tian K 2010 Software Optimization. Collection of recipes (SPb: Piter)
[6] Mallayev O U 2018 Algorithm and software for speeding up computer memory using OpenMP technology. Descendants of Mohammed al-Khwarizmi 2(4) 71-5
[7] Zayniddinov X N and Mallaev O U 2018 The algorithm and the program to speed internal memory multi-core processors Scientific technical journal of FerPI 4 91-6
[8] Zayniddinov X N and Mallaev O U 2018 Vectorization of parallel processing of seismic signals using parallel studio TUIT Bulletin 2(46) 14-25
[9] Intel® 64 and IA-32 Architectures Optimization Reference Manual Available from http://www.intel.com/content/dam/doc/manual/64-ia-32-architectures-optimization-manual.pdf
[10] Intel Parallel Studio home page Available from http://software.intel.com/en-us/forums/intel-parallel-studio