The Efficiency of the Code Parallelization in Multi Core Environment on the Basis of Image Processing in 3D Space

Krzysztof Oleszko

Abstract

Grains parameters like volume, surface area or shape factor are important in geological issues connected with rocks and coal mining. For quick and efficient way to calculate that parameters are use some complicated algorithms which are based on source data written in flat 2D images. Almost always raw images need to be filtered and always need to be transformed into 3D images and process in 3D space to obtain reliable results. Even if algorithms which can do that operations are run on efficient computers, operations performed in 3D space consumes a lot of processing time. Authors made attempt of parallelization procedures performed in 3D space to improve efficiency on computers equipped in widely used, multi core processors and presents results.

Keywords

3d image analysis • Image processing • Grains reconstruction • Parallel processing

Introduction

Image processing performed in 2D space can be very time consuming, especially when object of interest is written in high resolution image. Almost always processing is composed of many separated algorithms which are applied one after another in exact way to the image, to obtain desired result. When processing is moved into 3D space [17], then whole process becomes more and more time consuming, especially when procedures are compound of algorithms based on mathematical morphology. In that kind algorithms, to process single pixel it is necessary to read and process neighborhood of it, few times more than in 2D algorithms. In geology, 3D image processing is present, but not commonly used [14]. Especially in topics connected with grains and its structure [1, 3]. That kind of measurement are important in rocks and coal mining. There are some algorithms, applied with success to compute volume, surface area or shape factor of grains [11]. Yet, if there are a lot of data to process, algorithms processed on single processor can last many long minutes or even hours. To speed up this process, algorithms with some small modifications can be run simultaneously to shorten time of image processing and analysis.

Many authors describes different parallel image processing algorithms [6] in very narrow application context, or describes general parallel concepts and problems [4, 5, 8, 12]. Also many of scientists performs popular parallel computing on GPUs [2, 13]. GPU processing is time saving process, but creating such algorithms is not as easy as parallel algorithms performed on CPUs. Most of researches are more theoretical, and it is difficult to find exact results of some parallelization proved by real tests and comparisons [16]. That is why author decided to prepared algorithms and compare them on different CPUs to present obtained results.

For the project, there was created new code, written in JAVA based on existing code used in previous research [11]. To 2D and 3D images, were applied algorithms based on mathematical morphology [15]. On the images used for the project were presented group of grains and each grain...
needed to be processed and analyzed. As a result of research, authors proposed simple way of code parallelization, and presents promising results.

**Code Parallelization**

In general, computer programs are written to be processed on single processor or single core, where problem is divided into some small pieces, where each piece is executed, one after another. Whole program can be rewrite in order to execute with success in parallel environment where can be achieved grate speedups. But not always there is available complex parallel code or parallel environment. However, nowadays, single processor or single core computers seem to be history. For some years, majority of personal computers are equipped with multi core processors. When user, in the same time is executing few programs, operating system can manage that situation, and provide best efficiency utilizing all of available cores. But in situation, when programmer creates some algorithms which are executed one after another on exact part of data, operating system can do nothing with that situation. Algorithms are being processed on single core, even if there are available 4 or even 8 cores. That situation is common in image processing and analyzing issues, where there is available single image, and there need to be applied some filters on that image or some objects presented on the image. When programmer writes his code in some high-level programming language like JAVA, there are provided some mechanisms which can speed up image processing and analysis executing same code on different cores processing different individual parts of data. What is more, those parallel mechanisms can be easily applied in standard serial code. A programmer only need to keep in mind to write parallel loops instead of standard ones. After all, code is executed in more efficient way and saves time.

**The Algorithm**

The algorithm in serial version was used in previous research, where is described with details [10, 11]. In this project code was rewrite in JAVA and some crucial parts of it were parallelized. Basically, algorithm applies transformations and filters like watershed, border cleaning or different morphological filtrations on 2D source image, next there is created 3D image based on 2D one using advanced mathematical transformations based on image symmetrical reflection [9]. After that, separately for each grain, there is applied spatial morphological filtration algorithm and there are computed grains parameters like volume, surface area or grain dimensions [7]. Operations performed in 3D space were parallelized to improve efficiency.

**Test Environments**

The program was tested in two different kind of environments. First one was PC personal computer with 16 GB RAM and Intel Core i7 2.93 GHz 64 bit processor. Second environment was PC laptop computer with 2 GB RAM and Intel Core 2 Duo 2.0 GHz 32 bit processor. On both machines, was installed Windows 7 Professional operating system. In first environment the tests were run on JRE (Java Runtime Environment) 1.6 32 and 64 bit, and JRE 1.7 32 and 64 bit. In second environment the tests were run on JRE 1.6 and 1.7 32 bit.

**Experiment**

For the tests there were used two images containing real data. Each image contained about 50 grains. After grains separation, each one was processed separately in different CPU process to improve efficiency. The program were run on different number of cores, depending on the environment. In the environment with two cores, program were run on single and on two cores for both images. In the environment with eight cores, firstly, program was run on single core, and then number of cores was successively increased by one to number of eight. Each test containing single image was repeated five times on given number of cores. From obtained five results was taken average value which was used for further deliberations. All the tests were run in both environments. In the Intel Core 2 Duo tests were run on two kinds of JRE’s, in the Intel Core i7 environment tests were run on four kinds of JRE’s. That situation was caused by particular CPU architecture. On 32bit CPU can be used only 32bit JRE, but on 64bit CPU can be used both, 32bit and 64bit version of JRE.

**Selected Results and Discussion**

In performed tests, due to similar grain sizes presented on processed images, there was not applied any algorithm dealing with data irregularity. Also, processed tests proved, that the data taken under consideration were quite regular and all the processes were executing equally.

In the following discussion there are presented results for one image for all test cases. Results for both images were more or less similar and that is why one of them will be presented.
Table 1  Results for Intel Core 2 Duo environment

<table>
<thead>
<tr>
<th>JRE</th>
<th>Number of processes</th>
<th>Read time (s)</th>
<th>Preparation time (s)</th>
<th>Processing time (s)</th>
<th>Program time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.6</td>
<td>1</td>
<td>5.56</td>
<td>5.66</td>
<td>432.64</td>
<td>443.80</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td></td>
<td></td>
<td>229.37</td>
<td>231.65</td>
</tr>
<tr>
<td>1.7</td>
<td>1</td>
<td>5.37</td>
<td>5.75</td>
<td>196.73</td>
<td>207.81</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td></td>
<td></td>
<td>105.61</td>
<td>116.77</td>
</tr>
</tbody>
</table>

**Intel Core 2 Duo Results**

In Table 1 are presented results for Intel Core 2 Duo environment. In first column is presented version of JRE, in second number of processes. In third column is presented time of image reading, in fourth time after image reading and before proper processing which mainly includes initial filtration in 2D space. Fifth column presents 3D processing time and sixth presents full program execution time. It is clearly noticeable that JRE 1.7 is more efficient and JRE 1.6. The majority of program time was generated by operations in 3D space. Speedup for whole program for JRE 1.6 is equal 1.92 and for JRE 1.7 is equal 1.78. But speedup JRE 1.7 in relation to JRE 1.6 for single process is equal 2.14. Therefore, in this scenario should be used only JRE 1.7 and parallel code which provides best efficiency.

**Intel Core i7 Results**

An Fig. 1 shows image read time for all the JRE’s configurations. For JRE x64 and x86 there is a big difference in those times and for JRE x64 efficiency is better. Also between JRE 1.6 and 1.7 there is a small difference in image read time, it can be noticed that JRE 1.7 achieved better execution times.

On Fig. 2 image preparation time is presented, which mainly contains initial filtration (preparation) time in 2D space. Same, like for image read time, there is quite big difference between JRE x64 and x86—the JRE x64 is more efficient in both versions, 1.6 and 1.7. However there is almost no difference between JRE 1.6x64 and 1.7x64.

Similar to Intel Core 2 Duo, in the Intel Core i7 environment most of the program execution time is generated by parallel processing of data in 3D space. Thus, there will be presented time of full program execution and its speedup, and there will not be separately considered 3D processing time.

Figure 3 shows full program execution time, on the x axis the number of cores is presented. On the y axis the execution time in seconds is presented. Each line represents different JRE version. It is noticeable, that JRE’s x86 is the slowest one of all of them. For example, on single core on JRE 1.6x86 program was executing 248.41 s but on JRE 1.7 only 27.37 s. It is caused by processor architecture, which is 64bit architecture. There is simple conclusion, that on 64bit processor should be only used JRE x64.

On Fig. 4 there are presented speedups for x64 JRE’s. For both, JRE 1.6 and JRE 1.7 speedups are similar and there is no significant difference between them. The interesting thing is, that speedups are increasing up to 4 cores and they are quite good, after that, for 5–8 cores speedups are worse than...
for 4 cores. In spite of Intel Core i7 provides eight logical cores, the real speedups can be achieved only for four physical cores. Therefore, to achieve best results for that kind of environment, parallel code should be used for maximum four cores which gives speedups about 2.45 comparing to single core.

Summary

The problem of program execution efficiency in multi-core environment was presented in this paper. The main task of the experiment was to improve efficiency of image processing algorithms applied in 3D space using commonly user processors. Obtained results proofs that speedup of program run in multi core environment can be easily achieved. However, not always good hardware specifications brings satisfactory results. Also, the programmer must be aware of traps which are connected with proper hardware—software configurations.

Acknowledgment This work was financed by the AGH University of Science and Technology, Faculty of Geology, Geophysics and Environmental Protection as a part of Dean’s grant number 15.11.140.213.

References