Research Article

Parallelized Dilate Algorithm for Remote Sensing Image

Suli Zhang, 1 Haoran Hu, 2 and Xin Pan 1

1 School of Computer Project & Technology, Changchun Institute of Technology, Changchun 130012, China
2 School of Computer & Information, Anqing Teachers College, Anqing 246011, China

Correspondence should be addressed to Haoran Hu; haoranhu123@126.com

Received 10 February 2014; Accepted 27 March 2014; Published 11 May 2014

Academic Editors: Z. Zhou and X. Zhu

Copyright © 2014 Suli Zhang et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

As an important algorithm, dilate algorithm can give us more connective view of a remote sensing image which has broken lines or objects. However, with the technological progress of satellite sensor, the resolution of remote sensing image has been increasing and its data quantities become very large. This would lead to the decrease of algorithm running speed or cannot obtain a result in limited memory or time. To solve this problem, our research proposed a parallelized dilate algorithm for remote sensing Image based on MPI and MP. Experiments show that our method runs faster than traditional single-process algorithm.

1. Introduction

Land use/cover information has been identified as one of the crucial data components for many aspects of global change studies and environmental applications. The development of remote sensing technology has increasingly facilitated the acquisition of such information [1]. As an important algorithm, dilate algorithm can give us more connective view of a remote sensing image which has broken lines or objects. However, with the technological progress of satellite sensor, the resolution of remote sensing image has been increasing and its data quantities become very big. This would lead to the decrease of algorithm running speed or cannot obtain a result in limited memory or time.

Parallel program can split a big computing task into subcomputing tasks and make full use of the advantage of multicore and multicomputer to improve the computing speed [2]. To accelerate the process speed of remote sensing images algorithm, many methods had been proposed: parallel k-means or EM cluster method for remote sensing image [3, 4]. Wang utilized loud computing to a rapid processing of remote sensing image [5]. Parallel classification method has been proposed to archive a faster remote sensing images training speed [6, 7]. Parallel program can be further divided into multiprocesses parallel and multithreads parallel. Message passing interface (MPI) is a library specification for message passing, proposed as a standard by a broadly based committee of vendors, implementers, and users [8]; we can realize multiprocesses. Multiprocessing (MP) is the use of two or more central processing units (CPUs) within a single computer system [9].

In this research, we introduce MPICH2 and OpenMP technology and propose a parallelized dilate algorithm for remote sensing image (PDARSI); through PDARSI a big dilate task can be split into a lot of subtasks; each subtask can run on corresponding computer or core. Experiments show that our method runs obviously faster than traditional single-process algorithm.

2. Preliminary Knowledge

2.1. Dilate Algorithms. There are two sets $A$ and $B$ in $Z$; a complement set of $A$ is as follows:

$$A^{C} = \{\omega \mid \omega \notin A\}.$$  \hspace{1cm} (1)

Based on this formula the difference of $A$ and $B$ represented by $A - B$ can be defined as

$$A - B = \{\omega \mid \omega \in A, \omega \notin B\} = A \cap B^{C}.$$  \hspace{1cm} (2)

The reflection of $B$ represented as $\bar{B}$ can be defined as

$$\bar{B} = \{\omega \mid \omega = -b, b \in B\}.$$  \hspace{1cm} (3)
Figure 1: Five stages of algorithms.

3. Parallelized Dilate Algorithm for Remote Sensing Image

The generic process of parallelized dilate algorithm for remote sensing image (PDARSI) is shown in Figure 1.

Firstly, in main function, MPI interface start and initial all the processes by:

```c
MPI::Init(argc, argv); // Initial all the process;
```

```c
size = MPI::COMM_WORLD.Get_size(); // Get number of processes;
```

```c
rank = MPI::COMM_WORLD.Get_rank(); // Get current process's rank number.
```

And then the algorithm is divided into five steps: (1) Rank0 read the entire remote sensing data, and data in accordance with the number of processes is divided into multiple subdata; (2) Rank0 sends data, and the data is distributed to each process; (3) each rank processes its own data to obtain the corresponding results; (4) Rank0 collect all the data, and data integration as a result; (5) Rank0 write the result to disk. Finally, in main function, call

```c
MPI::Finalize(); // stop all the process.
```

All the process was destroyed and the algorithm was ended.

Stage 1. Reading stage: in order to solve the problem of image data read, task assignment, and data transmission, PDARSI adopt a Plines class to store the remote sensing data; Plines class has the following functions: (1) storing remote sensing image in units of row; (2) the storage part of the data; (3) redundant storage of data boundary information; (4) supporting serialization; and (5) supporting the reconstruction of the data; the process of Rank0’s reading and splitting can be described in Algorithm 1.

Through this procedure, Rank0 can split all the data into subdata corresponding to each Rand, and the PDARSI proceed to Step 2. In Step 2, Rank0 send all the subdata to Rank0 to Rankn by
Input: $S$ (Remote Sensing image), size (the number of processes), window (the window size of dilate algorithm)
Output: A array of Plines objects

Begin
PLines object pall = read all the data of $S$;
$n = \text{rows of } S$;
rown = floor ($n$/size);
PLines[] array = new Plines[size];
for $i = 0$ to size − 1 loop
    array[$i$].DataStartPosition = $i$ * rown;
    array[$i$].DataEndPosition = ($i + 1$) * rown;
    array[$i$].UpperBuffer = window/2;
    array[$i$].BottomBuffer = window/2;
    if $i == 0$ then array[$i$].UpperBuffer == 0; endif;
    if $i == $size − 1 then
        array[$i$].BottomBuffer = 0;
        array[$i$].BottomBuffer = $n - i$ * rown
    endif
end loop
return array;
End

Algorithm 1

<table>
<thead>
<tr>
<th>The number of processes</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>The number of threads</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>79.03</td>
<td>41.36</td>
<td>29.16</td>
<td>22.35</td>
<td>18.27</td>
<td>15.38</td>
<td>13.63</td>
<td>11.84</td>
<td>13</td>
<td>15.8</td>
</tr>
<tr>
<td>2</td>
<td>57.02</td>
<td>37.27</td>
<td>31.63</td>
<td>28.19</td>
<td>26.1</td>
<td>24.73</td>
<td>23.87</td>
<td>23.24</td>
<td>23.5</td>
<td>23.29</td>
</tr>
<tr>
<td>4</td>
<td>37.46</td>
<td>28.08</td>
<td>24.64</td>
<td>22.85</td>
<td>21.78</td>
<td>21.08</td>
<td>20.55</td>
<td>20.81</td>
<td>20.8</td>
<td>20.68</td>
</tr>
<tr>
<td>5</td>
<td>30.88</td>
<td>22.7</td>
<td>20.36</td>
<td>18.47</td>
<td>17.58</td>
<td>17.1</td>
<td>17.29</td>
<td>20.42</td>
<td>17.2</td>
<td>17.14</td>
</tr>
<tr>
<td>7</td>
<td>27.67</td>
<td>21.49</td>
<td>19.71</td>
<td>20.92</td>
<td>17.71</td>
<td>18.42</td>
<td>20.71</td>
<td>17.75</td>
<td>17.62</td>
<td>19.74</td>
</tr>
<tr>
<td>9</td>
<td>28.59</td>
<td>23.7</td>
<td>21.98</td>
<td>18.31</td>
<td>18.24</td>
<td>18.62</td>
<td>20.69</td>
<td>18.02</td>
<td>18.02</td>
<td>18.56</td>
</tr>
<tr>
<td>10</td>
<td>26.43</td>
<td>22.35</td>
<td>21.08</td>
<td>20.01</td>
<td>20.25</td>
<td>22.67</td>
<td>19.67</td>
<td>19.55</td>
<td>19.41</td>
<td>22.15</td>
</tr>
</tbody>
</table>

**Table 1:** Run speed in multiprocess and multithread.

$\text{MPI::COMM\_WORLD.Send}(\&\text{linehead}, \text{sizeof(Line Head)}, \text{MPI::BYTE}, 0);$ //send Plines head

$\text{MPI::COMM\_WORLD.Send}(\text{lines[i]}, \text{GetSendBuffer Head}(), \text{sizeof(Line head)}, \text{MPI::BYTE}, 0);$ //send Plines data.

4. Experiments

This research chooses Landsat-5 TM image and extracts a band for test image; the image size is $5230 \times 4736$ and 23.6 M (see Figure 3).

To test the efficiency of PDARSI algorithm, we adopt a HPC cluster which contains Intel i5 2300 computer as head node; it controls two compute nodes which have AMD FX8350 8-core CPU. Each computer of cluster utilizes Fedora 16 64-bit Linux operating system and MPICH2 as MPI management interface and OpenMP as multithread library. In order to test the effectiveness of parallel algorithms, we adopt the number of processes from 1 to 10 and the number of threads from 1 to 10, totally 100 times test. The dilate operator use $15 \times 15$, in each pixel of image algorithm would compute $15 \times 15 = 225$ times, Table 1 is test result.

It can be seen from Table 1, when there are one processes and one threads, the algorithm is equivalent to the traditional...
single-process algorithm, the running time is slowest 79.03 seconds, with the number of processes or the number of threads increase the running speed become faster. When 10 processes and one thread the algorithm running time is 22.67 seconds, only 0.27 times of the single-process run time. When 10 threads and one process the running time is 15.8 seconds, only 0.2 times of the single-process run time. Both MP and MPI can play an important role in accelerating the program running speed.

Figure 4 is algorithm speed and its relation to the number of processes and 1–5 threads. As can be seen from Figure 4(a), the elapsed time of algorithm declined along with the increasing number of processes, but the trend became slower when the processes number exceeds 4. Figures 4(b) and 4(c) show that the algorithm’s speed increases more slowly when the thread process number is bigger than 1; this means that threads can bring more increase than the processes. From Figures 4(d) and 4(e), the number of threads in the initial stage more than 4, number of processes’ increase may actually reduce the operating speed, which is due to the improvement of the process of the speed has less influence than the speed of data.
transmission between processes. Figure 5 is algorithm speed and its relation to the number of processes and 6–10 threads.

In Figure 5, there are less differences among the five figures, due to HPC cluster computer hardware limitations (the compute nodes containing a total of 16 cores) and the time-consuming communication among processes. The algorithm's speed is not linear with the number of processes and threads, and when the speed limitation is reached, the increase of the number of processes or threads will not increase running speed or even decrease the speed. The algorithm will archive better result when more powerful HPC cluster is utilized.

The relation between threads and processes can be seen from Figure 6. Multithread method does not require data transmission so is can bring more obvious increase in algorithm speed, since algorithms require the transmission of data between two computers in multiprocess stage, when process number is even algorithm need transmitted half of
the data from Rank0 node to another node, so the speed is more faster in processed number is odd.

The results of the PDARSI algorithm and the traditional single-process algorithm can be seen in Figure 7.

PDARSI algorithm splits a remote sensing image into subdata, and each subdata has UpperBuffer and BottomBuffer to ensure a pixel which at subdata border can access neighbor pixels on the other subdata; this mechanism guarantees that the dilation algorithm can obtain right result even the whole calculate task are partitioned into processes. When the dilation algorithm calculation in each process is completed, Rank0 collect the results from processes and integrate all the results into a result image. As can be seen from Figure 7, PDARSI does not change the results of the original algorithm and result images are exactly the same; this proves that our proposed algorithm can accelerate running speed and does not alter the results of the original algorithm.
5. Conclusions

This research uses MPICH2 and OpenMP to design a parallelized dilate algorithm; it can take full advantage of HPC cluster computing resources and achieve the purpose of rapid processing of remote sensing image. Through PDARSI a big dilate task can be split into a lot of subtasks; each subtask can run on corresponding computer or core. Experiments show that our method runs obviously faster than traditional single-process algorithm.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgment

This research was supported by the National Natural Science Foundation Youth Fund of China (41101384).

References


