Explorations of the implementation of a parallel IDW interpolation algorithm in a Linux cluster-based parallel GIS

Fang Huang a,*, Dingsheng Liu b, Xicheng Tan c, Jian Wang b, Yunping Chen a, Binbin He a

a Institute of Geo-Spatial Information Technology, College of Automation, University of Electronic Science and Technology of China, 2006 Xiyuan Road, Chengdu 611731, China
b Center for Earth Observation and Digital Earth, Chinese Academy of Sciences, 45 Beisanhuan Xi Road, Beijing 100086, China
c Institute of Geo-Spatial Information Technology, College of Automation, University of Electronic Science and Technology of China, 2006 Xiyuan Road, Chengdu 611731, China

A R T I C L E   I N F O

Article history:
Received 16 June 2009
Received in revised form 28 April 2010
Accepted 10 May 2010
Available online 10 November 2010

Keywords:
IDW interpolation algorithm
PCs cluster
GRASS GIS
MPI
LiDAR

A B S T R A C T

To design and implement an open-source parallel GIS (OP-GIS) based on a Linux cluster, the parallel inverse distance weighting (IDW) interpolation algorithm has been chosen as an example to explore the working model and the principle of algorithm parallel pattern (APP), one of the parallelization patterns for OP-GIS. Based on an analysis of the serial IDW interpolation algorithm of GRASS GIS, this paper has proposed and designed a specific parallel IDW interpolation algorithm, incorporating both single process, multiple data (SPMD) and master/slave (M/S) programming modes. The main steps of the parallel IDW interpolation algorithm are: (1) the master node packages the related information, and then broadcasts it to the slave nodes; (2) each node calculates its assigned data extent along one row using the serial algorithm; (3) the master node gathers the data from all nodes; and (4) iterations continue until all rows have been processed, after which the results are outputted. According to the experiments performed in the course of this work, the parallel IDW interpolation algorithm can attain an efficiency greater than 0.93 compared with similar algorithms, which indicates that the parallel algorithm can greatly reduce processing time and maximize speed and performance.

© 2010 Elsevier Ltd. All rights reserved.

1. Introduction

GIS applications have not been limited to the geographic field, but also been introduced in many other domains. However, certain special applications require that the GIS software provide high-performance capabilities for certain operations, including storage, retrieval, and processing of geospatial data. These requirements, in a conventional GIS, have increasingly led to bottlenecks in computational efficiency (Healey et al., 1998; Zhao et al., 2005; Huang et al., 2007).

With its maturity and successful application in many scientific fields, parallel computing technology has been rapidly adopted in the geoscientific domain (Armstrong and Marciano, 1994, 1996; Healey et al., 1998; Shekhar et al., 1995; Wang, 1993; Li et al., 2006). Under these circumstances, some researchers have started to study the use of new GIS software in a high-performance computing environment; in other words, a new GIS research field, the parallel GIS (Healey et al., 1998; Shekhar et al., 1995), has emerged. In particular, current research has integrated parallel computing technology with grid computing in the context of a geocyberinfrastructure, providing one of the major high-performance techniques for distributed geospatial information processing (DGIP) to support digital-earth (DE) research (Yang et al., 2008).

To design a new parallel GIS and its dedicated high-performance GIS services, we proposed a reasonable and practical approach to exploring various parallelization algorithms on a particular open-source GIS software package based on a Linux cluster (the Linux cluster-based OP-GIS) (Huang et al., 2007, 2008), which considered both ease of access to the parallel platform and the GIS source codes, and the entire process of how to parallelize the entire GIS package on the this platform, rather than focusing on the parallelization of single typical GIS algorithms, as has been done in other approaches. Because of differences in computing platform and programming method, the computing mode, architecture, relevant data structures, and software framework of the Linux cluster-based OP-GIS are different from those of a conventional GIS (Huang, 2008). Moreover, how to develop and implement parallel algorithms for a particular basic GIS algorithm, namely, the parallelization pattern, formed the technological basis of the Linux cluster-based OP-GIS. The parallelization pattern of the OP-GIS includes the data parallel pattern (DPP) and the algorithm parallel pattern (APP) (Huang, 2008), which are based mainly on the usual classification of parallel programming techniques, data partitioning, and function partitioning, which together constitute the effective parallel programming approach for most applications.
In the APP, to explore the working model and principle of the APP, the parallel inverse distance weighting (IDW) interpolation algorithm has been chosen as an example because this interpolation algorithm is computation-intensive, well-documented, easy to implement, and widely applied in many fields, including geology, soil research, geophysics, oceanography, meteorology (Papari and Petkov, 2009), and ecology and environmental studies (Chang et al., 2006; Jason and Jeffrey, 2002). Typically, this algorithm is available in many GIS software packages to create a continuous surface from a set of discrete points for the use of terrain analysis and light detection and ranging (LiDAR) point-cloud processing (Jason and Jeffrey, 2002; Guan and Wu, 2008; Brovelli et al., 2002). It should be noted that, because LiDAR point-cloud data often consist of very dense points and require a large amount of time to process, they are highly suitable for high-performance processing methods.

In existing research in parallel IDW algorithms, Armstrong and Marciano (1994, 1996) implemented its parallel algorithms based on a “brute force” method (Hodgson, 1989) and the Clarke (1990) approach respectively, using FORTRAN 77 on shared-memory parallel supercomputers, and achieved an efficiency close to 0.9. Currently, research efforts are being devoted to parallel IDW algorithms on multi-core platforms (Guan and Wu, 2008). In their experiment, the parallel IDW algorithms used open multi-processing (OpenMP) running on an Intel Xeon 5310, achieving an excellent efficiency of 0.92. However, because of the different purposes, different platforms, and different languages of this research, the implementation of the IDW parallel algorithm is very different from that of other researchers.

For this reason, based on a systematic examination of the serial IDW interpolation algorithm of the Geographic Resources Analysis Support System (GRASS GIS), an open-source GIS software package, a specific parallel IDW interpolation algorithm has been proposed and designed with message passing interface (MPI) in this research, which is adapted both to the platform and to the GRASS GIS, and incorporates both single process, multiple data (SPMD) and master/slave (M/S) programming modes. With regard to the GRASS GIS 5.4, Alexei Popov, a Russian researcher, developed a parallel version of the IDW algorithm with the sites data type in 1995,2 publishing the result as a parallel version for GRASS 5 on the Internet. However, in GRASS 6.X, the data types have been changed greatly, and the earlier version can no longer be used. Moreover, our parallel IDW algorithm in the GRASS software is a single monolithic algorithm that can accommodate both raster and vector input files. In other words, it can be used to implement parallel modules either with r.surf.idw or with r.surf.idw.

In comparative experiments, the parallel algorithm as implemented was found to attain a speedup factor of almost 6 and an efficiency greater than 0.93 under a Linux cluster linked with six independent PCs. This indicates that the parallel IDW interpolation algorithm can greatly reduce processing time and achieve excellent speed and performance compared with other similar algorithms.

The paper is organized as follows. Section 2 gives a brief introduction to various background topics for parallel IDW interpolation, including the Linux cluster, the MPI, the GRASS GIS, and the principle and processing details of the IDW algorithm. Section 3 concentrates mainly on the implementation of the parallel IDW algorithm. Finally, Section 4 discusses the experimental results and draws some conclusions.

2. Brief introduction and background

This research has focused mainly on the parallelism of the IDW algorithm in the GRASS GIS by adopting the MPI for the Linux cluster platform. The following sections provide a brief introduction to relevant technologies such as parallel computing and IDW algorithm implementation in GRASS.

2.1. Parallel computing

Parallel computing is one of the major types of high-performance computing (HPC). This term refers to any computational technique that solves a large problem faster than is possible using stand-alone, off-the-shelf systems. Parallel computing has had a tremendous impact on a variety of areas, ranging from computational simulations for scientific and engineering applications to commercial applications in data mining and transaction processing (Grama et al., 2003). In essence, parallel processing entails dividing a problem into multiple tasks that can be executed within several processes upon a number of processors. How this is accomplished depends on the hardware and software environments (Mineter et al., 2000).

With the rapid evolution of computer hardware and software technologies, it has become easily possible to obtain and build effective parallel hardware platforms, such as the Beowulf cluster, linking personal computers across Ethernet networks. This makes it feasible to design and implement a parallel GIS based on a high value-cost ratio parallel computing environment. For this very reason, in the current research, the parallel platform used is a Linux PC cluster connected by a legacy system with a Gigabit Ethernet network.

Meanwhile, the MPI has become the predominant software environment for parallel programming. MPI is a standard for implementing parallel programming and is a library rather than a programming language. The implementation of the MPI standard is MPICH, which is dedicated to providing an MPI implementation for common parallel platforms, including clusters, SMPs (symmetric multiprocessors), and MPPs (massively parallel processors). It also provides a platform for MPI implementation research and for developing new and better parallel programming environments. The version of MPICH in our Linux PC cluster is 1.2.6, which can be downloaded at http://www-unix.mcs.anl.gov/mpi/mpich.4

2.2. GRASS GIS

GRASS is a geographic information system (GIS) used for geospatial data management and analysis, image processing, production of graphics and maps, spatial modeling, and visualization. GRASS is currently used in academic and commercial settings around the world, as well as by many governmental agencies and environmental consulting companies. It is widely used in many fields such as resource management, land-use planning, hydrology, mineral exploration, groundwater modeling, and fire management (Neteler and Mitasova, 2008). Because GRASS is an open-source software package released under the GNU General Public License (GPL), anyone can obtain and edit its source code to meet different needs.

2.3. Principle of the IDW interpolation algorithm

In the GRASS GIS, the IDW interpolation algorithm is one of the techniques used for spatial interpolation.

2.3.1. Spatial interpolation

Spatial interpolation is the process of estimating the values of properties at unknown points within an area covered by existing observed points. In many situations, spatial interpolation operations are performed in a GIS, e.g., to provide contours for displaying

---

1 http://grass.itc.it/.
data graphically, to calculate some property of the surface at a given point, or to analyze and predict a trend surface. Spatial interpolation is a very important feature of a GIS, and it is also used for prediction and representation in many fields.

There are several different ways to classify spatial interpolation procedures, such as point interpolation/area interpolation, global/local interpolation, and exact/approximate interpolation. Many techniques exist for both global and local interpolations: trend surface analysis and Fourier series are examples of global techniques, while proximal, Kriging, and B-splines techniques are local techniques. In particular, the inverse distance weighted (IDW) spatial interpolation algorithm is a typical local interpolation algorithm.

2.3.2. Principle of IDW

IDW basically depends on two assumptions: (1) the unknown value of a point is influenced more by nearby control points than by those farther away, and (2) the degree of influence (weight) of points on each other is directly proportional to the inverse of the distance between the points raised to a power and can be represented by the following equation (Bartier and Keller, 1996; Wu and Shi, 2003):

\[ Z = \frac{\sum_{i=1}^{n} w_i Z_i}{\sum_{i=1}^{n} w_i} = \frac{\sum_{i=1}^{n} Z_i / D_{pi}^p}{\sum_{i=1}^{n} 1 / D_{pi}^p}, \]

where \( Z \) is the interpolated value for a point with an unknown observed value, \( w_i \) the weighting function that determines the relative importance of each individual control point \( Z_i \) in the interpolation procedure, \( Z \) the observed value at control point \( i \) \((i = 1, \ldots, n)\), which is in the closest neighborhood of the interpolated point, and \( n \) the total number of such points that are used in the interpolation and is typically set to between 20 and 30. There are different ways to determine the \( n \) closest neighborhood control points, and their computation times vary considerably (Armstrong and Marciano, 1994, 1996). In the present research, \( n \) is a parameter, and its default value is 12. Meanwhile, the determination of \( n \) can require a considerable amount of computation because all control points will need to be searched to calculate each grid cell; \( D_{pi}^p \) is the distance from the observed control point \( i \) to the interpolated point, where \( p \) is an arbitrary positive real number called the weighting exponent. When \( p \) is set to 1, the interpolation method is called inverse distance weighting (IDW), while if \( p \) equals 2, the method is called inverse square distance weighting (ISDW). However, \( p \) is defaulted to 2 in many weighting functions. Here, the weighting function depends on Euclidean distance and is radially symmetric about each scatter point. As a result, the interpolating surface is somewhat symmetric about the spatial interpolation point.

The IDW implementation in GRASS has six steps, as illustrated in Fig. 1:

Step 1: Initialize the GRASS GIS;
Step 2: Acquire the relevant input parameters and validate their correctness. Among those parameters, npoints is the maximum number of reserved points, usually set to 12;
Step 3: According to the type of the input file, get the elements belonging to the Point data structure from either a raster or a vector file. The acquisition procedure is different for the two file types; however, the result for a single reserved point is of type Point, and all the Point data are stored in the Point array;
Step 4: Open the destination raster file and write the basic information required in raster format;
Step 5: Calculate interpolated values for one row and write them sequentially into the output raster file. The iteration will continue until the procedure for all rows has been completed;
Step 6: Deallocate memory and terminate the program.

Although GRASS has two types of IDW interpolation modules, there is no noticeable difference in their implementations except for preprocessing to different input file formats. In both cases, the output file can be a raster file. (Note that v.surf.idw can also produce a vector-format file, e.g., Thiessen polygons when the parameter npoints is equal to 1.) However, good knowledge about the surface to be interpolated is necessary to guarantee an optimal result over the entire area by means of interactive control of the interpolation parameters (GDF Hannover, 2005).

2.4. Implementation of the serial IDW interpolation algorithm

The implementation of the serial IDW interpolation algorithm refers to the abovementioned modules. During implementation, the data structure Point is defined to describe and store the scattered point data elements extracted from the input file, which can be either raster or vector data. The procedure for reading each of these is different; however, both can be converted into the Point structure and stored in the Point array. This structure can be described by the following pseudocode:

```c
struct Point
{
    double north; //Geodetic coordinates;
    double east; //Geodetic coordinates;
    double z; //Attributes values, such as elevation.
    double dist; //Distance value to one interpolation point.
};
```

The IDW implementation in GRASS is available for interpolating areas in the raster model format: (1) changing the resolution of raster data (resampling) and (2) filling in of incomplete data (interpolation). For the first application, several GRASS interpolation modules can be used: the nearest neighbor method (NN) and spline interpolation (regularized splines with tension, or RST); furthermore, bilinear, bicubic, and bspline methods are also available (Mitavskaya et al., 2007). For the second application, another two modules can be used: IDW and RST (Neteler and Mitavskaya, 2008; GDF Hannover, 2005).

Because the data type of each location is transcribed into vector format in GRASS 6.0, the GRASS modules for IDW interpolation can be divided into:

- v.surf.idw, whose input file is vector data,
- r.surf.idw and r.surf.idw2, whose input file is raster data in both cases.

It should be noted that, in a GIS, the values of the observed control points are not necessarily elevations, but can also be values of other attributes of the locations being processed. One of the procedures most frequently performed is the interpolation of data points regularly or irregularly distributed over a closed data surface, which is frequently used for modeling and simulation.

2.4.1. Interpolation modules in GRASS

Generally, in the GRASS GIS, two different applications are available for interpolating areas in the raster model format:

3. Parallel IDW algorithm in GRASS

3.1. Analysis of parallelism opportunities

From a time-consuming test using the serial IDW interpolation algorithm, it can be observed that the elapsed time for Step 5, as shown in Fig. 1, was almost 84% of the time required by the whole process. Within this step, the interpolated values for each row are iteratively calculated cell by cell and row by row in the raster file and are finally stored in cell[row], the row array for storing the raster file. These iterations accounted for nearly 95% of the time spent in Step 5.

Obviously, the opportunity for parallelism in the algorithm focuses on this step. It is possible to assign different column intervals to different computing nodes in the cluster, e.g., columns 1–100 are assigned to node 1, columns 101–200 to node 2, columns 201–300 to node 3, and columns 301–400 assigned to node 4, assuming that an entire column contains 400 elements and that there are four nodes in a cluster. Only when the task of calculating interpolated values row by row is evenly distributed to all the nodes can the processing time be considerably reduced. After each node completes its own subtask for its specified column extent according to the IDW interpolation formula, the master node will gather the interpolated data into its cell array and write them to the raster file. In this way, it is possible to implement the detailed parallel IDW algorithm.

3.2. Implementation of the parallel algorithm

According to the above approach, this research has proposed and designed a specific parallel IDW interpolation algorithm, incorporating both the Linux platform and the GRASS GIS, and integrating both SPMD and M/S programming modes. There are nine main steps in the parallel IDW interpolation algorithm; an entire flowchart is illustrated in Fig. 2. Among them, Step 7 can be divided into the following substeps:

Step 7 (1): Row iteration. Start from the first row;
Step 7 (2): Each node calculates the interpolated values in its own column extent for a specific row according to the sequential equation, i.e., Eq. (1);
Step 7 (3): The master node gathers the sub-row data from all nodes after each node has finished its own calculation. The procedure is accomplished by the functions MPI_Probe(), MPI_Get_count(), and MPI_Recv() in the master node and MPI_Send() in the slave node. Then the master node writes the interpolated values row by row into the output raster file. This substep can be described by the following code:

```c
if (rank==MASTERNODE) {
    MPI_Barrier(MPI_COMM_WORLD);
    rescell=dcell+window.cols/size;
    for (node=1; node<size; node++){
        MPI_Probe (node, 10, MPI_COMM_WORLD, &stat);
        MPI_Get_count (&stat, MPI_DOUBLE, &recvsize);
        MPI_Recv (rescell,recvsize, MPI_DOUBLE, node, 10, MPI_COMM_WORLD, &stat);
        rescell+=recvsize;
    }
    G_put_d_raster_row (fd, dcell);
    MPI_Barrier(MPI_COMM_WORLD);
}
else{
    MPI_Barrier(MPI_COMM_WORLD);
    info=MPI_Send (dcell, sendsize, MPI_DOUBLE, 0, 10, MPI_COMM_WORLD);
    MPI_Barrier(MPI_COMM_WORLD);
}
```

Step 7 (4): The iteration will continue until all rows have been processed, and then the master node outputs the result of the interpolation.

It should be pointed out that in Step 4, to broadcast the corresponding known information, besides the number of measured sample points (npoints), the number of search points (nsearch) comprising the Point array for observed points and some additional information related to the Windows (one data structure in GRASS GIS), which are essential for other nodes to calculate the interpolated value, it was necessary to design and declare new data types, MPI_Points and MPI_Window, as described by the following code:

```c
1. MPI_Datatype MPI_Points;
2. int blocklens[4] = {1, 1, 1, 1};
3. MPI_Datatype types[4] = {MPI_DOUBLE, MPI_DOUBLE, MPI_DOUBLE, MPI_DOUBLE};
4. MPI_Aint displacements[4] = {0, sizeof (double), 2*sizeof (double), 3*sizeof (double)};
5. MPI_Type_struct (4, blocklens, displacements, &MPI_Points);
6. MPI_Type_commit (&MPI_Points);
```
In Step 4, each node will ascertain the value of `npoints` and `nsearch` and obtain the geodetic coordinate values of the interpolated point ((`Point`) `points[npoints]`) according to the specific row and column numbers of the known points. Moreover, each node will also malloc some specific arrays, e.g., (`Point`) `list[nsearch]` is for searching the nearest measured sample points and (**`double`) `dcell[window.cols]` stores the interpolation values for one specific row. Therefore, except for the master node, which has to read the initial dataset, each other node has only to deal with its own set of interpolation information extracted from the initial dataset, which was received from the master node.

In Steps 6 and 7, all the nodes, including the master node, are involved in the processing. At first, each node calculates its own column interval using code such as the following:

```c
// Each process calculates the right cell extent among the column array.
int nAverage, StartNo, EndNo;
nAverage = window.cols / size; // size is the number of nodes
if (rank == size-1){
```

---

**Fig. 2.** Flowchart for parallel IDW interpolation algorithm in GRASS.
After this, each node processes only its own column extent in one row iteration according to the IDW interpolation equation, once it obtains a copy of the input data. This procedure includes the following main steps:

(1) Start of row iterations;
(2) Ascertain the starting north and east coordinates for this specific row in this node, as described by the following:
   north = window.north - window.ns_res/2.0;
   east = window.west - window.ew_res/2.0;
   east + = window.ew_res/2.0 - window.cols/size;

(3) Column iteration starting from the first cell to the last cell in its own column extent, i.e., from StartNo to EndNo for one node. For one specific cell (known row and column), its coordinates are calculated;
(4) Find the nearest reserved points for this cell; the search procedure is the so-called “brute force” approach (Hodgson, 1989), divided into the following main substeps:
   (4-1) Calculate the distances from the previous set of nsearch reserved points (points[npoints]) to this cell, and enter these distances into the search array, i.e., list[nsearch] as the initial values;
   (4-2) Find the largest distance, i.e., maxdist, from list[nsearch];
   (4-3) Compare the rest of the calculated distances (dist) in points[npoints] with maxdist, ranging from nsearch to npoints-nsearch. If dist < maxdist, then replace the largest distance with dist and record its other information into list[nsearch];
   (4-4) After the comparison, the information stored in list[nsearch] is the nsearch nearest reserved points to this specific cell;
(5) Calculate the interpolated value for this cell using Eq. (1), i.e., the value of Z; store it in the dcell array; calculate the next column in the column extent.

From this point, the parallel algorithm uses the SPMD programming mode. However, it also needs the master node to carry out such functions, e.g., broadcasting basic information, gathering row results, and writing into the output raster file. In this algorithm, because each node calculates only the interpolation values that fall into its own column extent, the whole calculation task is divided evenly over the nodes, which can greatly reduce the processing time.

4. Experimental results and conclusions

In the experiments reported here, the parallel IDW interpolation modules constructed as described above were executed using the existing GRASS GIS package on a Linux PC cluster, together with an extra abbreviation of the prefix (p) in their module names. In this scenario, the performance of serial and parallel algorithms on some unsolved problems was compared.

4.1. Linux PC cluster configurations

The experimental parallel platform was set up using six independent PCs linked with an Ethernet network with Gigabit speed. The configuration of the test cluster is described in Table 1. The parallel modules were constructed using C and MPICH 1.2.6.

4.2. Experimental results and analysis

According to the implementation principle described above, several modules derived from the IDW interpolation algorithm, e.g., r.surf.idw2, and v.surf.idw, can be parallelized in the same way. As a matter of fact, in GRASS, other modules, such as r.surf.idw, r.surf.rst, and v.surf.rst, which are derived from a different interpolation algorithm, can also be parallelized by refereeing the process using the parallel IDW algorithm.

To address issues related to the correctness and the performance acceleration of the parallel IDW algorithm, the sequential module, v.surf.idw, and its corresponding parallel module, p.a.v.surf.idw (where “p” means that the algorithm type is parallel-programmed and “a” indicates that the module is derived from the APP pattern), as well as several different groups of experimental data, were selected, as shown in Table 2. The size and the number of discrete points contained in these group datasets vary widely: the small dataset has 6506 points and occupies 747.3 K, the medium-size dataset has 64,828 discrete points and occupies almost 7.3 M, and the large dataset is a LiDAR point-cloud dataset, which has 3,265,110 discrete points and occupies almost 242.9 M. It is important to note that the serial and parallel algorithms discussed here must malloc enough memory for all points at the beginning of processing each node. If the input points are very dense, the program may not be able to obtain all the memory needed from the system.

To execute the parallel module, the following command lines are needed:

**Example Command**

```
mpirun -np 6 p.a.v.surf.idw input=Serpent_Mount_Model_LAS_sites input=r.surf.idw input=Model_LAS_sites output=Model_LAS_sites_idw mpi06 npoints=12 layer=1 column=cat
```

In this MPI command line, “-np” means the numbers of processors (NP) that are operating on this parallel module; here, npoints is set to 12.

<table>
<thead>
<tr>
<th>Node</th>
<th>Hardware</th>
<th>Software</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td>Intel® Core(TM)2 Duo CPU E8400 3.0 GHz, 2G RAM, 320G Disk</td>
<td>CentOS 4.7, GRASS GIS 6.2.0, MPICH 1.2.6, etc.</td>
</tr>
<tr>
<td>C2</td>
<td>Pentium® Dual-Core CPU E5200 2.50 GHz, 2G RAM, 320G Disk</td>
<td></td>
</tr>
<tr>
<td>C3</td>
<td>Intel® Pentium® 4 CPU 2.50 GHz, 512M RAM, 120G Disk</td>
<td></td>
</tr>
<tr>
<td>C4</td>
<td>Intel® Celeron® CPU 2.53 GHz, 512M RAM, 80G Disk</td>
<td></td>
</tr>
<tr>
<td>C5</td>
<td>Intel® Celeron® CPU 2.53 GHz, 512M RAM, 80G Disk</td>
<td></td>
</tr>
<tr>
<td>C6</td>
<td>Intel® Celeron® CPU 2.53 GHz, 512M RAM, 80G Disk</td>
<td></td>
</tr>
</tbody>
</table>

Table 1

Configuration of the test cluster.
First of all, the very point being addressed here is the correctness of the parallel IDW algorithm. Therefore, an additional comparison of the output files from the serial and parallel modules is first performed. The output files (generated from both the serial and the parallel IDW modules) are both loaded into GRASS, and then the "Raster Map Calculation" operation is carried out: subtraction result $= \text{serial output file} / \text{parallel output file (NP = 2, 3, 4, 21)}$. The following command line is used as an example to verify the correctness of the parallel-processing result when \( np = 6 \).

```
r.mapcalculation
amap=Serpent_Mound_Model_LAS_pts_idw
bmap=Serpent_Mound_Model_LAS_pts_idw_mpi06
formula=A-B outfile=result
```

The result is that all values of the subtraction raster grid are zero (Fig. 3), which means that there is no difference between the outputs of the serial and parallel algorithms, and therefore that the parallel optimization procedure does not affect the final interpolation results. It should be noted here that, because of space limitations, Fig. 3 illustrates only the subtraction result from the parallel module performed with \( np = 6 \), but the other experiments on other groups of data appear to show the same degree of correctness.

When a parallel program is implemented in a cluster using MPI, two important indices, speedup and efficiency, are used to evaluate its performance. Speedup is a measure of the improvement in execution time for the parallel algorithm and is expressed as the ratio of \( T_0 \) (the execution time for the single-stream algorithm) to \( T_p \) (the execution time for the parallel algorithm).
(Brawer, 1989):

\[
\text{Speedup} = \frac{T_s}{T_P}
\]

Efficiency is the ratio of speedup to \(P\), the number of processes executing the algorithm:

\[
\text{Efficiency} = \frac{\text{Speedup}}{P}
\]

The closer the efficiency is to one, the more efficient is the parallel algorithm.

From the above explanation, to evaluate the performance of a parallel system, first of all, it is necessary to obtain the elapsed time for the serial module and the parallel module under different numbers of processors (\(NP\)) processing the same data. For a very small dataset, e.g., arch_sites, which has only 25 discrete points and occupies only 56 K, both sequential and parallel processes are very fast because of its small size, but the elapsed time for the parallel module increased markedly with the value of \(NP\), because the cost of communication in this process is dramatically increased. As for the small, medium, and large groups listed in Table 2, the processing times of the serial and parallel modules were extremely different, as illustrated in Fig. 4.

From Fig. 4, it is known that 57.3 h are required to compute the interpolation map (to the large dataset) when a single processor (such as a C4 node) is used. The run time is reduced to 8.3 h, however, when all six PCs are used. Moreover, it can be found that the minimum time is obtained when \(NP\) is set to 6 or 7; after that, the elapsed time rises rapidly and then levels off, but with ongoing large oscillations. An interesting question is how it could happen that the minimum time was obtained for \(NP = 7\), when the experimental Linux cluster has only six PCs. Through additional contrast experiments on homogeneous and heterogeneous parallel platforms, it can be found that the phenomenon is related to: (1) the fact that two of the PCs have dual cores; and (2) the order of the machines used to run MPI jobs in the configuration file (\(y/share/machines.LINUX\)). The parallel platform illustrated in Table 1 can attain the minimum elapsed time and the best speedup when the \(NP\) value is between 6 and 8. Because of space limitations, the details of these comparison experiments are not presented here.

Then, according to Eqs. (2) and (3), the speedup and efficiency of the parallel IDW interpolation algorithm can be calculated, obtaining the results illustrated in Figs. 5 and 6.
Although in the current experiment the Linux cluster was made up of six PCs, the parallel algorithm could not achieve substantial optimization efficiency with fewer than six processors. However, for those groups, from the results obtained, it is clear that the parallel IDW interpolation algorithm is capable of obtaining better speedup (up to a factor of 6) than other similar algorithms as well as excellent efficiency (over 0.93). The achievement of such high efficiency can be attributed mainly to two factors: (1) the IDW interpolation algorithm is simple and highly suitable for parallelization; and (2) because our parallel platform is part of an independent LAN with 1000 M speed, the state of the network is excellent.

Obviously, the performance acceleration obtainable by the parallel IDW interpolation algorithm varies with the dataset size. With a small dataset, processing is not very time-consuming, and therefore the parallel algorithm cannot achieve much acceleration because more time is consumed in the parallel program communication. On the other hand, with a large volume of data, the parallel IDW interpolation algorithm can achieve excellent performance acceleration, and performance is much closer to a linear speedup. However, when the data reach a certain size, the performance will not keep pace any more.

4.3. Conclusions

In this paper, the implementation and optimization of a parallel IDW algorithm in a Linux cluster with the GRASS GIS package is presented, together with the results of extensive experimentation and analysis. Although the parallel IDW interpolation algorithm can achieve excellent performance acceleration, its problem with memory allocation needs to be considered further. Moreover, the great potential of such an approach to OP-GIS remains to be fully demonstrated. In particular, further research is needed on how to use this method in models for near-real-time simulation or efficient forecasting, such as dust-storm forecasting in a very limited time using Web Map Service (WMS) (Xie et al., 2010).

Acknowledgements

This work was supported by the National Natural Science Foundation of China (Grant nos. 41001221 & 40701146), the Fundamental Research Funds for the Central Universities (Grant no. ZYGX2009073), and the National High-Tech R&D Program of China (Grant no. 2007AA112227). Thanks to Boyan Brodaric, the Deputy Editor of C&G, and the anonymous reviewers for their positive and constructive suggestions to improve the paper. We thank Yefei Li for providing their obsolete PCs to construct the new platform. We would also like to thank the websites: www.appliedimagery.com and www.liblas.org, for freely supplying experiment LiDAR datasets and LibLAS tools.

References
