Partitioning of Vector-Topological Data for Parallel GIS Operations: Assessment and Performance Analysis

Terence M. Sloan[1], Michael J. Mineter[2], Steve Dowers[2], Connor Mulholland[1], Gordon Darling[1], and Bruce M. Gittings[2]

[1] EPCC, University of Edinburgh, EH9 3JZ, UK,
tms@epcc.ed.ac.uk,
http://www.epcc.ed.ac.uk/
[2] Department of Geography, University of Edinburgh EH8 9XP, UK

Abstract. Geographical Information Systems (GIS) are able to manipulate spatial data. Such spatial data can be available in a variety of formats, one of the most important of which is the vector-topological. This format retains the topological relationships between geographical features and is commonly used in a range of geographical data analyses. This paper describes the implementation and performance of a parallel data partitioning algorithm for the input of vector-topological data to parallel processes.

1 Introduction

This paper describes a parallel algorithm for the partitioning of vector-topological geographical data and provides some initial results on its performance. This work has been undertaken as part of a project to develop parallel algorithms for analyses of geographical data. This project is a collaboration between EPCC and GIS-PAL both of whom are located in the University of Edinburgh, UK. It is funded by the UK funding body EPSRC.

2 Vector-Topological Data

The vector-topology model is at the core of most commercial GIS, ranging from traditional products such as ESRI's ARC/INFO through to the more recent object-oriented systems such as Smallworld and LaserScan Gothic. The vector representation of real-world objects by combinations of points, lines and areas allows the description of objects' occupancy of space. The addition of topology allows the expression of the relationships between these components and provides a formal description of the spatial relationships between the real-world objects [1]. Vector-topological datasets are wide-spread and increasingly available from a variety of sources. They include administrative, political, social and environmental data such as census enumeration districts, wards, counties, parishes, national parks, land use, land cover, geology, soil etc. Vector-topological datasets are thus also at the core of a wide range of applications.

P. Amestoy et al. (Eds.): Euro-Par'99, LNCS 1685, pp. 691–694, 1999.

3 The Parallel Data Partitioning Algorithm

The data partitioning algorithm utilised in this paper is based on the design described by Sloan and Dowers in [2]. The algorithm entails partitioning data into sub-areas and assigning one or more to each processor. This approach is preferred to a "pipeline" approach as it maximises the exploitation of existing sequential algorithms. It also allows large datasets to be analysed since the dataset size can be greater than that which can reside in one processor.

The partitioning of vector-topological data is not straightforward since the data comprises multiple types of records, interrelated by means of record identifiers. Typically the vertices are held in one record type, the relationship between polygons and vertices may be held in another and the attribute data are related to polygons by a third record type. Moreover, geographical operations on vector-topological data generally require : the spatial coordinates of the boundaries between geographical features where these boundaries must be sorted by their uppermost y coordinate and the attributes of the geographical features on either side of these boundaries.

The algorithm proceeds through three phases to extract and efficiently merge-sort the data from one or more datasets, according to these requirements. These phases are the Sort, Join and Geom-Attribute Distribution (GAD) phases that partition the data into sub-areas from the multiple input datasets. Parallel processes are used to extract and sort information efficiently. For the Sort and Join phases a minimum of two processes is required. A minimum of three processes is necessary to perform the different roles in the GAD phase. During the Sort and Join phases the participating processes are split into two different groups, the Source and Pool Groups. There is only one process in the Source group which coordinates the actions of the processes in the Pool groups during the Sort and Join phases and gathers the information on data distribution. The Pool processes perform the extraction and merge-sorting of data from the input datasets. During the GAD phase the Pool processes are further divided into the GeomAttributeServer group and the Worker group. Here the processes of the GeomAttributeServer transfer to a Worker process all the data necessary for a GIS operation to be applied on a part of the geographical area of interest.

4 The Implementation and Its Performance

The Sort and Join Phases of this algorithm have been implemented in ANSI C with the MPICH 1.1.1 version [3] of the MPI message passing standard. The original design makes extensive use of parallel I/O facilities where a file is partitioned across a number of disks and accessed by concurrent processes. This implementation does not make use of such facilities. In the Sort phase, the data input file resides on a single disk. The Source process handles all read accesses to the input file from all the other processes. The outputs from the Sort and Join phases are a number of files each containing a sorted list of records of the same type. For each record type there can be one or more files. The number of

files is dependent upon the memory available to each Pool process. The memory available is a tunable parameter.

The implementation was executed during a period of exclusive access to a 4 processor Sun Enterprise 3000 with 1 GB of shared memory. Each processor is a 250 MHz UltraSparc. All executions were repeated to ensure timings were consistent. Three datasets of 2.3 MB, 4.1 MB and 11.6 MB in size of geographical data residing on a local disk were used in the timings.

The graph in figure 1 displays the overall Sort/Join elapsed times for 2, 3 and 4 processors with 1 process per processor. (Note the Sort and Join phases require a minimum of 2 processes.) It is clear from the graph that the algorithm is not scalable and in fact degrades at 4 processes. Figures 2 and 3 display the elapsed times for the Sort and Join phases respectively. From figure 2 it can be seen that there is some speed-up in the Sort elapsed time as the number of processors increase. However this speed-up is insignificant compared to the overall time. Software profiling tools reveal the lack of speed-up is due to the the high number of small messages (less than 30 bytes) passing between the Source process (acting as the file server) and the other processes.

In figure 3 it is clear that Join performance degrades as the number of processors increase. This is due to the small number of input files to the phase. The Join phase allocates the files to the Pool processes for merge-sorting. When there are few files, most of the Pool processes stand idle. This is a feature of the design since it was originally intended for use on systems with a parallel file I/O facility and where the input dataset was much larger than available memory [2]. This situation is simulated by reducing the previously mentioned *memory available parameter* within the Pool processes. When this is reduced more input files for the Join phase are created. Since there is no parallel file I/O facility in the implementation these files must be opened directly by the processes. However, the number of file pointers a process can have open is fixed by the operating system. When this number is exceeded, the code fails. This means that for large datasets, when a small number of processes are used the Join phase fails. It will, however, succeed for larger numbers of processes. This can be demonstrated by the fact that the implementation was not able to operate on a 45 MB input dataset when there were 2 and 3 processes but succeeded when then there were 4 processes.

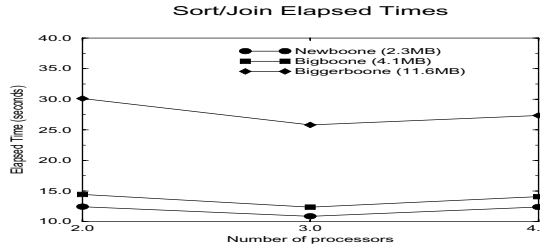

Fig. 1. Elapsed Times for Sort/Join

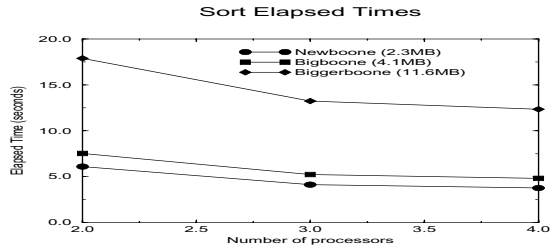

Fig. 2. Elapsed Times for Sort

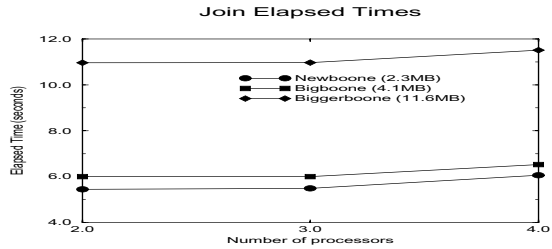

Fig. 3. Elapsed Times for Join

5 Conclusions and Future Work

The performance of the implementation could be improved further by altering the Source process in the Sort phase to buffer records into larger messages to be sent to the Pool processes. Furthermore the provision of a parallel file I/O facility where the input dataset was spread across a number of disks - as intended in the original design - would also improve performance. In the Join phase, such a parallel file I/O facility would also prove beneficial since it would allow valid comparisons of performance between small and larger numbers of processes on large datasets.

References

[1] Worboys, M.: *Representation and Algorithms.* GIS: A Computing Perspective. Taylor and Francis (1995)
[2] Sloan, T.M., Dowers,S.: *Parallel Vector Data Input.* Parallel Processing Algorithms for GIS. Taylow and Francis (1998) 151–178
[3] Gropp, W., Lusk, E.: User's guide for mpich, a Portable Implementation of MPI. University of Chicago ANL/MCS-TM-ANL-96/6 (1998)