High Performance Geographic Information Systems: Experiences with a Shared Address Space Architecture ?

Shashi Shekhar Sivakumar Ravada Vipin Kumar

Douglas Chubb Greg Turner

Abstract
Several emerging visualization applications such as flight simulators, distributed interactive simulation (DIS), and virtual reality are using geographic information systems (GISs) for high-fidelity representation of actual terrains. These applications impose stringent performance and responsetime restrictions which can be met via parallelization of the GIS. Currently, we are developing a high performance GIS on a Shared Address Space Architecture (SASA), namely, Silicon Graphics Challenge, as a part of the virtual GIS project of the Army Research Laboratories.

Our experience shows that data-parallel message-passing algorithms can be fairly effective for implementing high performance GISs on SASAs. Our experimental results with the GIS-rangequery operation show that data-partitioning is an effective approach towards achieving high performance in GIS. As partitioning extended spatial objects is difficult, special techniques such as systematic declustering are needed to parallelize the range-query operation. Systematic declustering methods outperform random declustering methods for range query problems. Static load-balancing methods with systematic declustering even outperform random declustering coupled with dynamic load-balancing (DLB). In addition, the performance of DLB methods can be improved by using the declustering methods for determining the subsets of polygons to be transferred during run-time.

1 Introduction

A high performance geographic information system (HPGIS) is a central component of many interactive applications like real-time terrain visualization, situation assessment, and spatial decision-making. The geographic information system (GIS) [9] often contains spatial data (geometric and feature data) represented as large sets of points, chains of line-segments, and polygons. In practice, the spatial data sets required to model geographic applications are usually very large (10-100 MBytes). Since interactive and real-time applications require short response times, the GIS has to process these large data sets in a very short time. The existing sequential methods for supporting the GIS operations do not meet the real-time requirements imposed by many interactive applications. Parallelization of GIS can meet the high performance requirements of several HPGIS applications. In this paper, we focus on parallelizing the range-query operation, which is important in many GIS applications, on a shared address-space architecture (SASA). We experimentally show that data partitioning using systematic declustering is effective for achieving good performance in many GIS applications.

1.1 Application Domain: Real-Time Terrain Visualization

A real-time terrain-visualization [11] system (e.g., a distributed interactive simulation system [2]) is a virtual environment [6] which lets the users navigate and interact with a three-dimensional computergenerated geographic environment in real-time. This type of system has three major components: interaction, 3-D graphics, and GIS. Figure 1 shows the different components of a terrain-visualization system for a typical flight simulator. The HPGIS component of the system contains a secondary storage unit for storing the entire geographic database and a main memory for storing the data related to the current location of the simulator. The graphics engine receives the spatial data from the HPGIS component and transforms this data into 3-D objects, which are then sent to the display unit.

As the user moves over the terrain, the part of the map that is visible to the user changes over time, and the graphics engine has to be fed with the visible subset of spatial objects for a given location and user's viewport. The graphics engine transforms the user's viewport into a range query and sends the query to the HPGIS unit. The HPGIS unit retrieves the geographic description of the area corresponding to the range query and sends the description back to the graphics engine. For example, Figure 1 shows a polygonal map representing vegetation types at a given location together with a range query. Each polygon in the map represents a land parcel with a given type of vegetation. Usually, a map projection is used to compute a 2D polygonal representation of the real world phenomena, which are measured in latitude and longitude. These polygons are shown with dotted lines. The range query is represented by the rectangle, and the result of the range query is shown in solid lines.
The HPGIS unit retrieves the spatial data from the main memory,

computes its geometric intersection

with the current viewport of the user, and sends the results back to the graphics engine. The frequency of this operation depends on the speed at which the user is moving over the terrain. For example, in the terrain visualization of a flight simulator, a new range query may be generated twice a second, which leaves less than half a second for the intersection computation. A typical map used in this application contains tens of thousands of polygons (i.e., millions of edges), and the range-query size can be 20-30% of the total map. This requires accessing millions of edges and computing millions of intersection-points in less than half a second. In order to meet such response-time constraints, a HPGIS often caches a subset of spatial data in main memory. The main-memory database may in turn query the secondary-storage database to get a subset of data to be cached. The frequency of this operation should be very low for the buffering to be effective.

1.2 Problem Formulation

The range-query problem for the GIS can be stated as follows: Given a rectangular query box B, and a set SP of extended spatial objects (e.g, polygons, chains of line segments), the result of a range query over SP is given by the set fxjx = Pi ? B 8Pi 2 SP g, where ? gives the geometric intersection of two extended objects. We call this problem the GIS-range-query problem. The computational complexity

of this problem for a given polygon is proportional to the number of edges of that polygon. Note that this problem is different from the traditional range query, where the objects in the given range are retrieved from the secondary memory (disk) to main memory without clipping the objects, but it is similar to the polygon-clipping problem in computer graphics. In this paper, we focus on parallelizing the GIS-range-query problem on a shared address-space architecture (SASA).

1.3 Scope

Alternative techniques like polygon simplification via culling [4, 5] and incremental range queries [12] can be used to further improve the performance of range query processing. Parallelization of the GIS-rangequery problem is orthogonal to these alternatives, since they can be used in addition to the parallelization techniques. In this paper, we only focus on the parallelization issues of this problem.

The parallel formulations discussed in this paper do not strictly require a shared address-space architecture. However, the availability of SASA eases the implementation of the dynamic load-balancing routines as discussed in section 4. We also focus on a data-partitioning approach to parallelizing the GIS-range-query. The simple approach of dividing the data statically between processors without using any run-time work-transfers is referred to as the static load-balancing (SLB) approach. The approach of using work-transfers during run-time to improve the load-balance is referred to as dynamic load-balancing (DLB).

2 Parallel Formulation for the GIS-range-query

The GIS-range-query problem has three main components: (i) approximate filtering at the polygon level, (ii) intersection computation, and (iii) polygonization of the result. (See [12] for a detailed discussion of a sequential algorithm.) A search structure is used to filter out many non-interesting polygons from the set of input polygons. The query box is then intersected with each of the remaining polygons, and the output is obtained as a set of polygons by polygonizing the results of the intersection computation.

The GIS-range-query operation can be parallelized either by function-partitioning [1] or by datapartitioning [12]. Function-Partitioning uses specialized data structures and algorithms which may be different from their sequential counterparts. A data-partitioning technique divides the data among different processors and independently executes the sequential algorithm on each processor. Data-Partitioning in turn is achieved by declustering [3, 10] the spatial data. If the static declustering methods fail to equally distribute the load among different processors, the load-balance may be improved by redistributing parts of the data to idle processors using dynamic load-balancing (DLB) techniques.

In this paper, we focus on parallelizing the GIS-range-query problem using the data-partitioning approach. Figure 2 shows the different stages of the parallel formulation. In the following discussion,

let P be the number of processors used in the system. Initially, all the data is partitioned into two sets, local data set Sa and shared data set Sb. The data in Sa is statically declustered into P sets Sia, for i = : : : (P ? 1); and Sia is mapped to the local address space of processor Pi. The purpose of local data set Sa is to reduce the synchronization overhead of accessing the data. The set Sb is shared among the processors for dynamic load-balancing (DLB).

When a bounding box for the next range query is received, each processor Pi performs the approximate polygon-level filtering, retrieves the candidate polygons from its local data set Sia, and starts work on intersection computation and polygonization. When a processor Pi finishes work on the local data, it goes into the DLB mode to process the polygons from Sb.

2.1 Static Declustering of Spatial data

The goal of declustering is to partition the data so that each partition imposes approximately the same load for any range-query. Intuitively, the polygons close to each other should be scattered among different processors such that for each range-query, every processor has an equal amount of work. Polygonal maps can be declustered at the polygonal level or at the sub-polygonal level. We focus only on polygon-level declustering to avoid the additional work of merging clipped sub-polygons.

Optimal declustering of extended spatial data (like polygons) is difficult to achieve, due to the nonuniform distribution and variable sizes of polygons. In addition, the load imposed by a polygon for each range query depends on the size and location of the query. Since the location of the query is not known a priori, it is hard to develop a declustering strategy that is optimal for all range queries. As the declustering problem is NP-Hard [12], heuristic methods are used for declustering spatial data. Random partitioning, local load-balance and similarity-graph-based methods are three popular algorithms for declustering spatial data. Intuitively, a local load-balance method tries to balance the work-load at each processor for a given range query. A similarity based declustering method tries to balance the work-load at each processor over a representative set of multiple range queries.
Refer to [12] for a detailed discussion of the declustering algorithms.

2.2 Dynamic Load-Balancing (DLB) Techniques

If static declustering methods fail to equally distribute the load among different processors, the loadbalance may be improved by transferring some spatial objects to idle processors using dynamic loadbalancing techniques. A typical dynamic load-balancing technique [8] for a shared memory computer addresses two issues: (i) how does an idle processor fetch more work, and (ii) what is the granularity of the work transfer and the data-partitioning method.

How Does an Idle Processor Fetch More Work?

In SMPs, a global work-pool is often maintained in the shared memory. An idle processor accesses this pool to fetch more work. We refer to this method as the Pool-Based Method (PBM). Synchronization may be necessary to access this pool. In addition, the structure of the GIS-range-query problem imposes a limit on the amount of work that can be kept in the shared pool. Since the filtering step is not easily parallelizable, a leader processor takes this responsibility for the shared pool. But if the leader processor has to manage too big a pool, some of the other processors might finish processing their local data before the leader processor can perform the approximate filtering step on the shared pool of data. This processing at the leader processor forces the other processors to wait for the leader processor to finish the filtering step on pool data. On the other hand, too small a pool can also lead to processor idling, due to the static load-imbalance in the rest of the data. These two situations are shown in Figure 3.

Granularity of Transfers and Data-Partitioning Method

Several strategies like self-scheduling, factoring scheduling, and chunk scheduling [7] exist for determining the amount of work to be fetched during DLB. The cost of synchronization affects the performance of these scheduling algorithms, as several processors have to share the knowledge about the remaining work. Small chunks can be used when the synchronization cost is low. In the case of chunks of more than one object, it is desirable to keep a comparable amount of work in each chunk, so that the load-imbalance can be kept low.

Since accurate work estimation is very expensive in the case of extended spatial data objects, the simple scheduling strategies are not suitable for GIS applications. However, if work can be divided into chunks of equal size, any of the scheduling strategies can be applied, which treat each chunk as the unit of work transfer. We note that this problem of dividing work into chunks of equal size is similar to the declustering problem. We hypothesize that the load-balance achieved by any DLB method can be improved by using a systematic declustering method for dividing the work into chunks. Since the declustering operation is expensive to perform during query processing, this chunking can be done during the loading of new data set into main memory. As this loading operation is performed relatively infrequently (less than once in 5 minutes), there will be sufficient time to perform the declustering operation without interrupting the actual range-query operation.

3 Experimental Evaluation

We experimentally evaluate the proposed parallel formulation on a bus-based SASA with 16 processors and 1GByte of main memory (SGI Challenge). Each node of this machine is a MIPS R4400 processor running at 200MHz. We use spatial vector data for Killeen, Texas, for this experimental study. This data has seven themes, representing the attributes slope, vegetation, surface material, hydrology, etc. We used the \slope" attribute-data map with 1458 polygons and 82324 edges as a base map in our experiments (this is denoted by 1X map). In the rest of the discussion, a kX map denotes a map that is k times bigger than the 1X map. Each polygon is represented as a sequence of edges, where each edge is a pair of 2D coordinate points. In addition, a search structure [12] is used to quickly identify the relevant edges of a polygon for a given range query. Each range query is chosen to be about 20% of the total area of the map.

In our experiments, we only measure and analyze the cost per range-query and exclude any preprocessing cost. This preprocessing cost includes the cost of loading the data into main memory and the cost of declustering the data among different processors. Note that this preprocessing cost is paid only once for each data set and that this operation is performed less than once in 5 minutes. The experimental results are compared in terms of the speedups achieved by different methods. We computed the speedup S for P processors as Tseq=TP , where TP is the parallel run-time on P processors, and Tseq is the sequential runtime on a single processor of the same system. Note that there are several parameters involved in evaluating an algorithm for the GIS-range-query problem. Here we present an evaluation of only those parameters which are interesting for SASAs. For a detailed study of other parameters, see [12].

3.1 Evaluation of the Granularity of Work-Allocation in DLB

We compare the effect of different chunk sizes with chunks containing 1 to 100 polygons, using PBM for DLB and the similarity-graph method for declustering. The experiment is conducted using the 16X , map for P = 4 to 16 with a 20% pool. Figure 4 shows the results of this experiment. The main trends observed from this graph are: (i) that chunk-size affects speedups, and (ii) that the chunk-size providing the best speedups goes up with an increasing number of processors. There are two extremes for dividing N polygons into different chunk sizes: (i) chunks of single polygons and (ii) chunks of N=P polygons. In the first case, synchronization cost and communication bottleneck due to limited bus bandwidth limit the speedup achieved, and in the second case, static load imbalance limits the speedup achieved. Hence, the peak speedup is usually achieved somewhere in between these two extremes, and the speedup curves in Figure 4 reflect this effect. In addition, as the number of processors increase, the load imbalance increases, increasing the number of chunks transferred at runtime. Hence, the optimal chunk size for maximum speedup is sensitive to the number of processors.

3.2 Declustering for DLB Methods

In this experiment, the effect of using declustering methods for chunking inside a fixed DLB is evaluated. We used PBM as the DLB method and compared random, similarity-graph, and LLB methods of declustering. The dynamic data Sb is declustered with fixed-size chunks (10 polygons per chunk). Figure 5 shows the experimental results. The x-axis gives the number of processors and the y-axis gives the average speedup over 75 queries. From this data, it is clear that random declustering of data is not as effective as systematic declustering for achieving a good load-balance in the GIS-range-query problem. This shows that systematic declustering of data improves the load-balance. Also, note that SLB with systematic declustering and without any DLB is better than SLB coupled with DLB using random declustering (as in Figure 5).

3.3 Effect of Pool Size for PBM

We evaluate the effect of the pool size using PBM as the DLB method. The number of processors is varied from 4 to 16, and the map is varied from 1X map to 16X map. The pool size is varied from to

100% of the total data. Note that a 0% pool refers to static declustering with no DLB. Figure 6 shows the results of this experiment. The x-axis gives the size of the pool as a percentage of the total data,

and the y-axis gives the average speedups over 75 range queries.

The speedups increase as we increase the pool size, up to a point, and then they start decreasing. The initial increase in speedups may be due to the improved load-balance. The decrease in speedup after achieving a maximum value is due to the non-parallelizable overhead of the approximate filtering. This result is consistent with observations made in Figure 3. The pool size, as a fraction of the total work, that results in the best speedup decreases with increasing map size. This may be due to the improvement in static load-balance due to the increased total work.

4 Parallel Programming Models for GIS-range-query Problem

A parallel implementation for the range-query problem can be developed using either a PRAM algorithm or a message-passing algorithm. PRAM algorithms often assume that the cost of synchronization and processor interaction are negligible. Thus PRAM algorithms often result in speedups inferior to what is predicted by the model. On the other hand, a message-passing algorithm has explicit accounting of communication cost and is more likely to result in an implementation that conforms to the performance predicted by the algorithm. The goal of message-passing algorithm is to enhance locality of access and

to avoid unnecessary contentions over access to shared data.

The parallel architectures are generally classified into two categories, namely, Shared address-space architectures (SASAs) and Message-Passing architectures. The SASAs include uniform memory access (UMA) architectures (e.g., bus-based symmetric multiprocessors) as well as distributed memory architectures which have hardware support to directly access remote memory (e.g. Stanford DASH and Cray T3D). Distributed memory architectures are also called non-uniform access (NUMA) architectures. In message-passing architectures, access to a remote memory location requires cooperation of the remote processor, and as a result, is often orders of magnitude slower than access to a local memory location.

Message-passing algorithms are relevant not only for message-passing architectures, but also for SASAs. A message-passing algorithm can be easily ported to a SASA using the PVM/MPI package. Unfortunately, the performance achieved by such packages did not meet the high-performance requirements of the GIS-range query problem for terrain visualization. Since each call to PVM/MPI is treated as a system call, the PVM/MPI implementation incurred run-time overhead which would otherwise be avoided. To improve performance, we manually implemented the message-passing algorithm on a SASA using the transformations given in Table 1. The transformation reduced the overhead incurred in the PVM/MPI-based implementation. The transformation converted the replicated data in the messagepassing algorithm to shared data in a SASA. This is not done automatically by PVM/MPI type systems. This address space mapping improves the capacity of the system to solve larger problems relative to a message-passing architecture [12].

Table 1 gives the mapping of a message-passing algorithm to a SASA. A message-passing algorithm uses three kinds of data: local data, read-only shared data and modifiable shared data. The read-only shared data is often replicated to facilitate data-sharing, particularly if data transfer is costly. The mapping divides the shared address-space in a SASA into local address spaces, a global address-space without locks, and a global address-space with locks to contain the three categories of data from a message-passing algorithm. Message-passing algorithms facilitate processor interaction via point-topoint communications operations such as Send, Receive, and collective communication operations such as One-to-all broadcast, All-to-all broadcast, and Barrier synchronization. These can be simulated by shared address space operations of read, write, lock, and unlock. Message-Passing algorithms use data partitioning (e.g. declustering) to increase locality and thus reduce processor interaction. The choice of a declustering method is largely independent of the architecture, whereas the choice of the dynamic load-balancing method is sensitive to the cost of processor interaction.

5 Discussion and Conclusions

Data-Partitioning is an effective approach for implementing high performance in GIS on SASAs. We parallelize the GIS-range-query problem using data-partitioning and dynamic load-balancing techniques, where data partitioning is used to reduce the synchronization overhead. Traditional data partitioning techniques like random declustering are not effective for GIS applications. Hence, systematic declustering techniques like local load-balancing and similarity-graph method are needed for efficient parallelization of GIS-range-query problem.

The experimental results show that systematic declustering methods outperform random declustering methods for range query problems. Even random declustering coupled with DLB is outperformed by SLB methods with systematic declustering with no DLB. The performance of DLB methods can also be improved by using the declustering methods for determining the subsets of polygons to be transferred during run-time.

In general, a GIS operation can be parallelized either on SASAs or on message-passing architectures. But there are several advantages to using SASAs, and particularly UMA architectures, for GIS problems, when compared to using a message-passing architecture. Parallelizing GIS problems requires run-time work-transfers to achieve a good load-balance, as the spatial data objects have varying sizes and complex shapes. An UMA architecture facilitates these run-time work-transfers, as all the data is equally accessible to all the processors. A disadvantage of the message-passing architecture is replication of data across processors for DLB. This data replication can also be avoided in NUMA architectures if

the cost of remote memory access is not much larger than the cost of local access. This data replication in message-passing architectures reduces the total main-memory available for storing the spatial data.

References

[1] S. G. Akl and K. A. Lyons. Parallel Computational Geometry. Prentice Hall, Englewood Cliffs, 1993.

[2] The DIS Steering Committee. The DIS vision - a map to the future of distributed interactive simulation. Technical report, Institute for Simulation and Training, May 1994.

[3] M. T. Fang, R. C. T. Lee, and C. C. Chang. The Idea of Declustering and its Applications. In Proceedings of the International Conference on Very Large Databases, 1986.

[4] J. Goldfeather, S. Molnar, G. Turk, and H. Fuchs. Near Real-Time CSG Rendering Using Tree Normalization and Geometric Pruning. IEEE Computer Graphics and Applications, 9(3), 1989.

[5] P. Heckbert and M. Garland. Multiresolution Modelling for Fast Rendering. In Proceedings of Graphics Interface, 1994.

[6] W. Hibbard and D. Santek. Visualizing Large Data Sets in the Earth Sciences. IEEE Computer, Special Issue on Vualization in Scientific Computing, August 1989.

[7] C. Kruskal and A. Weiss. Allocating independent subtasks on parallel processors. IEEE Transactions on Software Engineering, pages 1001{1016, October 1985.

[8] V. Kumar, A. Grama, A. Gupta, and G. Karypis. Introduction to Parallel Computing: Design and Analysis of Algorithms. The Benjamin/Cummings Publishing Company, Inc., 1994.

[9] R. Laurini and D. Thompson. Fundamentals of Spatial Information Systems. Academic Press Inc, 1992.

[10] D. R. Liu and S. Shekhar. A Similarity Graph-Based Approach to Declustering Problem and its Applications. In Proceedings of the Eleventh International Conference on Data Engineering, IEEE. The complete technical report Csci TR 94-018, available via WWW at URL: http://www.cs.umn.edu/research/shashigroup/paper ps/infs.ps., 1995.

[11] D. R. Pratt, M. Zyda, and K. Kelleher. Guest Editor's Introduction: Virtual Reality-In the Mind of the Beholder. IEEE Computer, Special Issue on Virtual Environments, July 1995.

[12] S. Shekhar, S. Ravada, V. Kumar, D. Chubb, and G. Turner. Declustering and Load-Balancing Methods for Parallelizing Geographic Information Systems. To appear in IEEE Transaction on Knowledge and Data Engineering. Also available as Technical Report TR 95-076, Department of Computer Science, University of Minnesota. URL:http://www.cs.umn.edu/Research/shashi-group/paper ps/rqtkde96.ps, 1995.