

Efficient Parallel Out-of-core Isosurface Extraction

Huijuan Zhang*

Department of Computer Science
University of Alabama in Huntsville

Timothy S. Newman†

Department of Computer Science
University of Alabama in Huntsville

Abstract

A new approach for large dataset isosurface extraction is presented. The approach's aim is efficient parallel isosurfacing when the dataset cannot be processed entirely in-core. The approach focuses on reducing the memory requirement and optimizing disk I/O while achieving a balanced load. In particular, an accurate model of isosurface extraction time is exploited to evenly distribute work across processors. The approach achieves processing efficiency by also avoiding unnecessary processing for portions of the dataset that are not intersected by the isosurface. To reduce the redundant computations and the storage requirements, a flexible, variably-granular data structure is utilized, thereby achieving excellent time and space performance.

CR Categories: I.3.5 [Computing Methodologies]: Computer Graphics—Computational Geometry and Object Modeling Curve, surface, solid, and object representations; I.3.6 [Computing Methodologies]: Computer Graphics—Methodology and Techniques Graphics data structures and data types

Keywords: isosurface extraction, load balancing, out-of-core, parallel processing

1 Introduction

Volume visualization can be helpful in the study of scalar volumetric datasets used in many science, engineering, and medical application areas. The isosurface extraction and rendering is one of the popular techniques in volume visualization. Isosurface extraction involves extracting a surface (typically, a faceted polygonal mesh) of constant scalar value (i.e., isovalue) from the dataset.

Probably the most popular [Grevera et al. 2000] isosurface extraction approach is the well-known Marching Cubes (MC) algorithm [Lorensen and Cline 1987]. The Marching Cubes operates on a regular, rectilinear grid of scalar data and produces a collection of triangular facets that approximate, for that dataset, the isosurface associated with a given isovalue.

One limitation of most conventional isosurface extraction methods (e.g., the MC) is that they have been designed for single-CPU environments. Thus, their processing steps are not always highly amenable to parallel computation. In addition, such methods usually cannot efficiently handle large datasets that cannot be pro-

cessed completely within resident memory (i.e., in-core). In this paper, an approach that uses MIMD-parallelism to efficiently extract isosurfaces from very large datasets is presented. The approach is designed to effectively organize the isosurface extraction process's access to memory and to evenly distribute components of processing in a way that yields a low runtime for datasets that cannot be processed completely in-core using conventional approaches, such as the standard MC. Specifically, the approach uses an accurate model of isosurface extraction time to evenly distribute workload among a multiprocessor environment's CPUs. The work distribution is performed in a data-centric manner that also allows regions of inactivity to be skipped during the building of the isosurface's facetization. The data-centric approach also enables efficient use of main memory and well-organizes access to secondary memory, primarily through use of a variably-granular data structure that allows load-balancing to be achieved without undue work duplication.

This paper is organized as follows. Section 2 describes related, background information. Our approach's computational framework and steps are described in Sections 3 and 4, respectively. Results and analysis are presented in Section 5. Section 6 contains the conclusion.

2 Related Work

Isosurface extraction is a time-consuming process, especially for a large dataset. In order to achieve interactive or real-time isosurfacing, many acceleration approaches have been developed. Since most isosurfaces intersect only a small fraction of the volumetric dataset's cells, many such approaches have attempted to limit unnecessary processing in (e.g., traversal through) the regions (i.e., cells) that the isosurface does not pass through. A region (or cell) not intersected by the isosurface is said to be *non-active*, whereas an intersected region (or cell) is said to be *active*.

2.1 Avoiding Non-Active Regions

Hierarchical geometric data structures, such as octrees, are one mechanism for accelerating isosurface extraction by avoiding non-active cells. Wilhelms and van Gelder [1990; 1992] were apparently the first to utilize the octree in such a manner. Others (e.g., Chiang et al. [1998]) have used interval trees to avoid traversal of non-active cells. The interval tree hierarchically organizes cells by minimum and maximum scalar values rather than by geometric relationship.

In addition, variant traversal orderings (i.e., traversals that do not proceed in cell-order) such as seed cell propagation approaches [Bajaj et al. 1996; Howie and Blake 1994; Shekhar et al. 1996] have been utilized to search only the active cells (and cells adjacent to the active cells) during isosurfacing. Wood et al. [2000] have presented a novel adaptive isosurface (i.e., the *semi-regular mesh*) extraction approach from distance volumes that uses a surface wavefront propagation method to extract a coarse mesh that is then refined using multi-scale refinement to control aspect ratios and sizes of mesh triangles. Their approach generates fewer triangles than does the Marching Cubes, yet the resulting meshes are still faithful to the original data.

*e-mail: hzhang@cs.uah.edu

†e-mail: tnewman@cs.uah.edu

2.2 Parallel Computation

Parallel processing is another way to improve isosurface extraction's performance. The literature includes reports of a few SIMD-parallel approaches, such as Hansen and Hinker's [1992] massively parallel isosurface extraction that evenly partitions the cells among the processors and Newman and Tang's [2000] vector-parallel Marching Cubes that uses a reorganized processing methodology to allow inherent data parallelism to be exploited. Most of the parallel isosurfacing methods have been MIMD-parallel approaches, such as the method of Mackerras [1992] which first divides the dataset into meta-cells (i.e., subvolumes composed of adjacent cells) and then distributes the same number of meta-cells to each processor of a large-scale MIMD-parallel computer. Each processor finally extracts the portion of the isosurface contained in its meta-cells. Miguet and Nicod [1995] have also presented a MIMD-parallel approach to Marching Cubes isosurface extraction that divides the dataset between processors in a slice-by-slice manner. In their approach, the isosurfacing work load is estimated for each slice as a weighted sum of the slice's cell and interpolated vertex counts. The slices are distributed to processors such that each processor has nearly equal work.

Sometimes, datasets are too large to be processed entirely within main memory (i.e., in-core). A few parallel isosurface extraction approaches aimed at efficient out-of-core operation have been presented. For example, Chiang et al. [1998] have presented an interactive out-of-core approach using the interval tree data structure. The approach divides the volume into many meta-cells and only loads into memory the meta-cells that intersect the isosurface. Chiang et al. [2001] have also parallelized this approach for operation on a cluster of computers. Their parallel approach employs a dynamic task dispatching scheme to balance computation among the processors. Another MIMD-parallel out-of-core approach is Bajaj et al.'s [1999] two-level decomposition method. Their approach first decomposes the volume into fixed-sized blocklets (i.e., small sets of adjacent cells). Then, blocklets are clustered into blocks so that each block is of a similar contour spectrum. The contour spectrum of a blocklet is a collection of estimates of the amount of computation to extract an isosurface in the blocklet for various isovalues. The computation work estimate is approximately related to the number of active blocklets. The isosurface extraction is performed by a seed cell propagation approach in each block. Blocks are distributed evenly among processors to achieve load balancing. Zhang, with Bajaj and others [2001], has also used the accelerated isocontouring approach of Bajaj et al. [1999] to achieve scalable isosurface visualization of large datasets on clusters of common off the shelf processors by using both parallel processing and parallel disks [Zhang et al. 2001]. In that work, they have used an I/O-optimal interval tree to minimize the time to load data from disk. Zhang et al. [2002] have also presented an efficient, parallel, out-of-core view-dependent isocontouring algorithm that distributes dataset blocklets randomly across processors. In addition, Gregorski et al. [2002] have presented a hierarchical Marching Tetrahedra (i.e., a variant of MC) algorithm for interactively extracting and rendering isosurfaces of large datasets on a multi-processing system in a view-dependent fashion.

One challenge in many parallel isosurface extraction approaches, including out-of-core approaches, is that performance gains have been limited. In particular, most parallel approaches have exhibited sub-optimal processor utilizations, partially due to load imbalances. The load imbalances have resulted primarily from either the use of a static load-balancing scheme that is based on a coarse workload estimation or from the use of a dynamic load-balancing scheme that involves moderate overhead, usually from synchronizing communications. In [Zhang and Newman 2001], we suggested that a very accurate work estimation model could enable a better load balancing for parallel isocontouring. (Our work estimation model

is briefly described in Section 3.2.) The approach [Zhang and Newman 2001] operates as follows. A preprocessing stage decomposes the volume into fixed-sized blocklets which are grouped based on cell extents and stored in particular range partition files. During the parallel computation stage, only blocklets whose values fall into a given range are loaded into memory. The work is estimated for each loaded blocklet based on the work estimation model, and a work scheduling scheme assigns the blocklets so that each processor has a nearly identical workload. However, some overhead is introduced (by that work estimation and by the process of work scheduling), which limits the model's efficiency. In this paper, we describe an approach that enables improved performance for parallel out-of-core isosurface extraction.

3 Computational Framework

As described in Section 2, historically the main strategies to reduce isosurface extraction time have been avoiding unnecessary processing in non-active cells or using parallel computation. Our approach's computational framework, which (like the frameworks of Zhang et al. [2001] and Chiang et al. [2001]) incorporates both strategies, is described next.

3.1 Reducing Storage Consumption and Access

Our approach is aimed at both efficient processing of datasets that are too large to be entirely resident in the main memory and at supporting the process of dataset exploration via isosurface extraction. Such exploration usually involves a series of extractions, quite often for a small range of isovalues (e.g., for queries on isovalues $k - \epsilon, \dots, k, \dots, k + \epsilon$). By loading into memory only blocklets that are intersected by isosurfaces associated with the range of isovalues involved in a set of queries, the extractions can be processed with less consumption of main memory, thus typically allowing in-core isosurface extraction for large out-of-core datasets.

Our approach also minimizes secondary memory accesses through several preprocessing mechanisms. First, the range \mathbf{R} of potential isovalues is partitioned into n intervals R_1, \dots, R_n , where each interval is non-overlapping with other intervals and the collection of intervals $\{R_i\}$ spans \mathbf{R} . Then, the dataset is partitioned into subfiles, which we call (range) partition files. Each partition file is associated with one isovalue interval, R_i . The blocklets that contain the active cells for any isosurfaces associated with isovalues in R_i are stored in partition file P_i .

3.2 Work Estimation Model

In order to fully utilize the computational resources in a parallel system, it is important to make the load balanced across the processors. Exploiting accurate work estimation is one way to achieve a balanced load. Many researchers have considered decomposing a volume into many sub-volumes, as described in Section 2.2. When coupled with a static load balancing strategy, these sub-volumes will be distributed to the processors and isosurface extraction will be performed by each processor for the sub-volumes assigned to it. In Marching Cubes, the amount of computation necessary to extract the portion of the isosurface that passes through an individual cell varies. For example, in some cells, several facets must be formed whereas in other cells only one facet must be formed. Thus, evenly dividing sub-volumes among processors is unlikely to yield highest performance. As mentioned earlier, several methods for estimating the isosurface extraction work for a sub-volume have been suggested. Many of the methods estimate work based primarily on the count of active cells within the sub-volume. In parallel isosurfacing, the work estimate is then used to distribute nearly equal amounts of work to each CPU.

Our work estimation model considers two factors, the number of cubes (N) and the number of active cubes (A), which have a significant impact on the isosurface extraction time. Determining N is straightforward. Determining A requires each cube to be tested to determine if the isosurface intersects it. We estimate the work W for isosurface extraction as a linear function:

$$W = \alpha N + \beta A, \quad (1)$$

where α is the time related to operations performed in every cube and β is the time for operations, such as interpolation and triangulation, performed only in the active cubes. Since we use a variant on Marching Cubes isosurface extraction which has linear computation time in N and A , this work estimation model seems reasonable. We empirically obtained the weights for α and β on each computer (i.e., on Origin2000, $\alpha = 1.3 \times 10^{-6}$, $\beta = 4.9 \times 10^{-6}$).

This work estimation model is useful to estimate work within a whole volume as well as within sub-volumes. For application to parallel, out-of-core isosurface extraction, we have applied the model to each blocklet. While the work estimation model is specifically for estimating the work in Marching Cubes-type isosurfacing, the model can probably be straightforwardly extended to accurately predict work in other isosurface extraction methods (e.g., the Marching Tetrahedra algorithm [Payne and Toga 1990]).

3.3 Hybrid Granularity Scheme

In data-parallel schemes, the data’s granularity plays an important role in the performance of a parallel algorithm. By data granularity, we mean the size of the data units which are the basis for dividing the work. Using coarse division (i.e., distributing large units of data), it is often difficult to evenly-divide the work to the CPUs. On the contrary, using fine granularity generally allows dividing the work fairly evenly. As mentioned earlier, some accelerated isosurface extraction approaches—in particular those using data-parallel methodologies to divide the extraction task among the CPUs—have used smaller data units as the basis for division of processing. In the isosurface extraction techniques in which the volume decomposition scheme is employed, the size of a sub-volume determines the degree of granularity. However, when the data is divided in a fine-grained manner, cube vertices lying on the boundaries between adjacent sub-volumes must be duplicated. That is, processing involving such vertices must usually be performed two or more times. To reduce the duplicated computation involving such vertices on the boundaries between adjacent sub-volumes and to reduce the size of output data generated by the duplicated computation, a more coarse-grained division would be a better choice if load imbalances could be avoided. In order to leverage both coarse-grained and fine-grained division’s advantages, we use a hybrid granularity approach in which variable-sized blocklets (specifically, small-sized and moderate-sized blocklets) are employed. (To our knowledge, the well-known parallel out-of-core isosurface extraction approaches (e.g., [Bajaj et al. 1999; Chiang et al. 2001; Zhang et al. 2001; Zhang and Newman 2001]) have used fixed-sized blocklets in dividing the work among the CPUs.)

We use blocklets that are an integral multiple of disk block size in order to optimize disk access. A bottom-up scheme is used to generate the variable-sized blocklets. The scheme first decomposes the whole volume into a large number of fixed-sized blocklets, each of which is a collection of spatially contiguous cells. If the ranges of scalar values within eight adjacent blocklets span the same intervals (described in Section 3.1), these eight adjacent blocklets are merged into a larger sub-volume. Thus, such a merger cannot cause merged blocklets to be stored in a large number of range partitions.

Our hybrid granularity approach’s exploitation of spatial coherence between adjacent blocklets also reduces the storage requirements of range partition files by eliminating duplicated vertices on

the boundaries between adjacent blocklets. In addition, since our work estimation model can consider each sub-volume individually and since the sub-volumes are assigned to each CPU based on the individual work estimates, the accuracy of our work estimation and flexibility of our work assignment is suitable both for fixed-sized sub-volumes and for variable-sized sub-volumes. The work assignment is discussed next.

3.4 Static Work Scheduling Scheme

Two general strategies for work load balancing across processors are the *static* and *dynamic* load balancings. In static load balancing, a fixed (and usually equal) amount of work is assigned to each processor before parallel computation; work assignment does not change during parallel computation. In dynamic load balancing, the work is distributed (or re-distributed) during parallel computation. Dynamic load balancing strategies can usually achieve nearly equal loads on the CPUs, but the balancing usually incurs overheads, such as work rescheduling computation and synchronization delays. Such overheads tend to reduce the total performance of dynamically-balanced isosurface extraction techniques. However, the static load balancing strategy typically can achieve a highly-balanced load only if the data granularity is fine and there is very accurate estimation of work. As suggested earlier, the duplication of some work in fine-grained division can limit total performance since unnecessary computation is performed.

In this paper, we use a static load balancing strategy to distribute work among processors. The work to extract the portion of the isosurface in each blocklet is estimated using our work estimation model (in a preprocessing stage), and the blocklets are assigned in a way that provides each processor with a nearly equal amount of work.

4 Approach

In this section, we describe the steps of our parallel out-of-core isosurface extraction approach in detail.

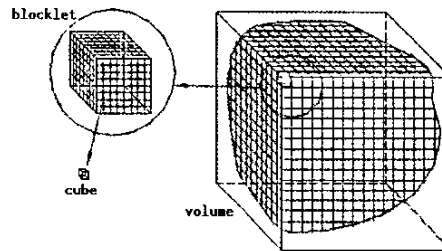


Figure 1: Illustration of relationship between processing units: volume, blocklet, cube (from [Zhang and Newman 2001]).

The preprocessing stage of our approach decomposes the volume into many $8 \times 8 \times 8$ blocklets, each an integral multiple of disk block size. (We have also explored use of $4 \times 4 \times 4$ units, but use of $8 \times 8 \times 8$ blocklets produces better total performance, as described later.) The relationship between processing units (i.e., volume, blocklet, and cube) is illustrated in Figure 1. In Figure 1, the shaded region illustrates an example of a blocklet. If the non-active blocklets of a very large dataset must be loaded into the memory, the performance of the isosurface extraction will suffer (due to unnecessary disk I/O). Moreover, when extracting an isosurface of interest, often multiple isovalue queries over some small range of isovalues are applied, especially near the end of the user’s querying process. Hence, the

approach loads into the memory only the blocklets that are active for at least one isovalue in a range of isovalues.

The range of all isovalues is partitioned as follows. First, the range of scalar values in a volume is partitioned into several equal-sized sub-ranges. Then the volume is processed in a blocklet-by-blocklet manner to check if each blocklet can be merged with its adjacent blocklets. The interval of each blocklet (i.e., its minimum and maximum values) is used by the merging operation. Two lists are built during merging, one to record the big (i.e., merged) blocklets and the other to record the small blocklets. For each $8 \times 8 \times 8$ blocklet, a binary flag indicates whether it (1) has been merged (i.e., is a part of a big blocklet in the big blocklets list) or (2) has not been merged (i.e., is an element of the small blocklets list). If the flag corresponding to the blocklet is not marked merged, the blocklet is examined to see if it can be merged with adjacent blocklets. Eight adjacent blocklets are merged only if their intervals span the same sub-ranges. Since merger does not occur for adjacent blocklets with different subranges, the merging operation itself does not cause a blocklet to be replicated in multiple range partitions. Whenever eight adjacent blocklets are merged, the resultant big blocklet is stored in the big blocklets list and the flag states for the eight adjacent blocklets are set to “merged.” The range of isovalues spanned by each big blocklet and the scalar values stored in each big blocklet are also written to the appropriate range partition files. Moreover, the global index (i.e., the start position of the big blocklet) and interval information of each big blocklet are recorded in a separate file. Otherwise, non-merged blocklets are stored in the small blocklets list. The global index and interval information of each blocklet are also stored. The two blocklet lists are stored in different files.

Figure 2 illustrates a 2D example of merging. In the figure, each 2D grid cell represents a small blocklet. The shaded grid cells represent merged blocklets. Whenever four adjacent cells (i.e., eight adjacent blocklets in 3D) satisfy the merging condition, they are merged to form a big cell.

The merging operation is repeated until all the blocklets have been considered. At the end of this process, the volume will be decomposed into variable-sized blocklets. In our implementation, only two sizes of blocklets are formed.

Subsequently, work estimation and work scheduling algorithms are performed on those variable-sized blocklets in a range partition of interest. For each isovalue in the sub-range, the number of active cubes within a blocklet can be determined by comparing the extents of each cube within the blocklet with the isovalue. The work to extract the isosurface in each blocklet is estimated using Equation 1. The work scheduling algorithm assigns the big blocklets and small blocklets separately across processors so that each processor has an approximately identical amount of work to do. Each processor is assigned a contiguous set of big and a contiguous set of small blocklets, with each set designed to involve an approximately equal work load. A look-up table is built to hold work assignment patterns for any number of processors between one and the predefined maximum number of processors. The use of the look-up table provides flexibility to distribute work on systems of different configurations.

At the end of the preprocessing stage, four files (i.e., two files for big blocklets and two files for small blocklets) associated with each range partition and a work assignment look-up table are available for use in the subsequent isosurface extraction.

At the start of parallel computation, our approach consults the look-up table. According to the number of processors in the system, the starting indices of big and small blocklets assigned to each processor in the system are found (with little cost). According to the information of each CPU’s starting indices, it is easy to compute the number of big and small blocklets assigned to each processor. On each processor, the appropriate big and small blocklet data are loaded (i.e., the scalar values, global index, and interval values) from the range partition files. An interval-based data struc-

ture is employed to skip blocklets that are not intersected by the isosurface. Each processor extracts the portion of the isosurface in its own blocklets using a reorganized MC algorithm [Newman and Tang 2000]. The facets are then finally gathered for display.

5 Experimental Results

Our parallel out-of-core isosurface extraction approach was tested on a SGI Origin2000 at the National Center for Supercomputing Applications (NCSA). The Origin2000 is a scalable distributed, shared memory multiprocessor with 112 MIPS R10000 (250 MHz) processors connected by hubs. The hubs direct the data flow between processors, memory, and I/O. Our implementation uses MPI for data exchange and CPU synchronization. Our tests and their analyses are discussed next.

5.1 Blocklet Size Effects

The appropriate blocklet size ($8 \times 8 \times 8$) was chosen based on experiments on size $4 \times 4 \times 4$ and $8 \times 8 \times 8$ blocklets using the two-level decomposition approach [Bajaj et al. 1999]. Experiments were performed on a magnetic resonance angiography (MRA) brain dataset (MRAHigh) of size $1024 \times 1024 \times 144$ containing 8-bit scalars on the Origin2000. Isosurfaces associated with isovalues 50 and 75 were extracted for the dataset. At isovalues 50 and 75, the extraction time using $8 \times 8 \times 8$ blocklets was 31.6% and 42.3% faster, respectively, than the time using $4 \times 4 \times 4$ blocklets (due to less redundant computation for duplicated vertices on the boundaries between adjacent blocklets). (The merging process results in some $15 \times 15 \times 15$ ¹ big blocklets, of course.) The isosurfaces are shown in Figures 3 and 4.

5.2 Propagation Effects

As mentioned, we used Newman and Tang’s [2000] reorganization of the MC algorithm. This approach processes cube vertices first in a vertex-by-vertex march to find all vertex statuses. Then, a series of vertex-by-vertex processings is used to find cube topologies. A series of edge-by-edge processing is next used to compute the interpolations, followed by cube-by-cube faceting to construct triangles. Such an approach has been found to usually allow more efficient serial computation of Marching Cubes than using the original Marching Cube’s organization of processing.

Isovalue	% cells active	Seed propagation (secs)	Reorganized MC (secs)
50	22.1%	5.56	3.39
75	1.3%	0.33	1.75

Table 1: Comparison of seed propagation and reorganized MC extraction times on Athlon PC, low-res dataset.

We have compared the reorganized Marching Cubes methodology with Howie and Blake’s² seed propagation isosurface extraction [1994] for both serial and parallel computation. The comparison of serial computation performance was done using a MRA brain dataset of size $256 \times 256 \times 72$ on a PC with an AMD Athlon 750MHz CPU. This dataset is a low-resolution version of the MRAHigh dataset. Table 1 shows the extraction times for these two isosurface extraction algorithms. For isovalue 75, the performance

¹We also made experiments on blocklets of size $16 \times 16 \times 16$ by padding each $15 \times 15 \times 15$ blocklet with zeroes. The performance is similar for these two sizes of blocklets.

²Bajaj et al.’s out-of-core approach also uses Howie and Blake’s methodology.

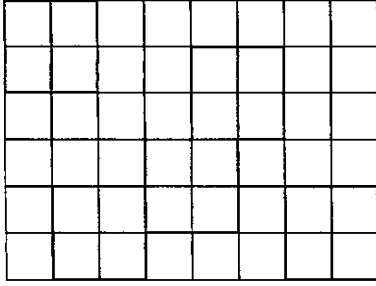


Figure 2: Example of the merging pattern in a volume in 2D space.

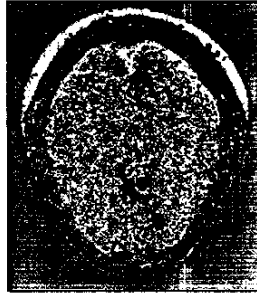


Figure 3: Isosurface with isovalue 50 from MRAHigh dataset ($34M$ triangles).

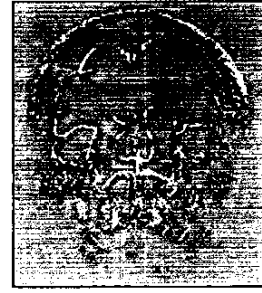


Figure 4: Isosurface with isovalue 75 from MRAHigh dataset ($1.3M$ triangles).

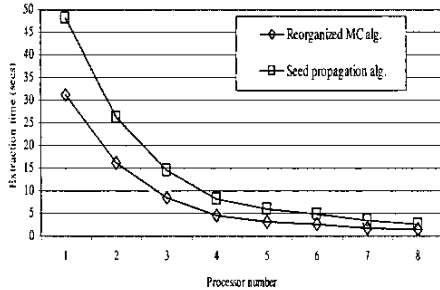


Figure 5: Comparison of extraction time of (a) the reorganized MC algorithm and (b) the seed propagation algorithm for MRA2 dataset at isovalue 75 on Origin2000.

of the seed propagation algorithm is better, while for isovalue 50, the performance of the reorganized MC is better. In this dataset, there are more non-active cells for the isovalue 75 isosurface than for the isovalue 50 isosurface. The seed propagation algorithm avoids processing many non-active cells for isovalue 75 which improves its performance significantly. However, as the number of non-active cells decreases, the seed propagation algorithm skips the processing of fewer non-active cells.

More importantly, the reorganized MC has unit stride data access, which can allow better cache performance. Every item in a cache line is accessed for the reorganized MC while the seed propagation approach generally accesses only a small number of items in each line. Thus, seed propagation tends to exhibit a higher cache miss rate. In addition, the CPU's predictive prefetching capability can more easily predict future cache accesses for unit-stride access than for the less structured access pattern in seed propagation. The hardware's predictive prefetching thus allows the cache miss rate to be further reduced for the reorganized MC. Therefore, in serial processing environments when the percentage of active cells is moderate or better, isosurfacing approaches using highly structured access patterns, such as our reorganized MC, are likely to outperform approaches with less structured access patterns, such as seed propagation approaches.

We also tested the parallel performance for these two algorithms on the Origin2000 for configurations involving one to eight CPUs. An MRA brain dataset (MRA2) of size $512 \times 512 \times 576$ was used in the test. The parallel seed propagation approach tested was Bajaj et al.'s [1999] volume decomposition and range partitioning scheme. Figure 5 shows the extraction time of the seed propagation algorithm and the reorganized MC for isovalue 75. Although the seed propagation algorithm produces good performance for serial computation on an entire volume, its advantage is limited for cases in which the unit of processing are sub-volumes that are active in a

range of isovalues. The results suggest that the seed propagation algorithm is thus not a good choice for parallel computation in cases in which the percentage of active cells in the whole dataset or in a range partition is high. Therefore, the reorganized MC seems to be a reasonable basis (considering strictly its computational aspects) for extracting isosurfaces in a parallel computation environment.

5.3 Blocklet Merging

We also conducted an experiment on the MRAHigh dataset to compare the *hybrid granularity scheme* using variable-sized $8 \times 8 \times 8$ and $15 \times 15 \times 15$ blocklets with our previous *single granularity scheme* [Zhang and Newman 2001] using fixed-sized blocklets (e.g., $8 \times 8 \times 8$). For the range partition file associated with range 40 to 79, using the hybrid granularity scheme results in 95,932 blocklets of size $8 \times 8 \times 8$ and 9,398 blocklets of size $15 \times 15 \times 15$. Using the single granularity scheme, the total number of blocklets of size $8 \times 8 \times 8$ is 171,116. Compared to the single granularity scheme, in the hybrid granularity scheme the total storage space is reduced by 6.9 MB (about an 8% reduction). For isovalue 50, the number of triangle vertices is reduced by about one million, which is a 4.7% reduction.

The merging efficiency of the hybrid granularity scheme depends very much on the characteristics of the dataset. If the number of adjacent blocklets which have similar range of values is high, the scheme can have more potential to reduce the disk storage space, data read time and computation time of duplicated vertices.

5.4 Load Balancing

In order to determine the degree of load balancing achieved by our hybrid granularity parallel out-of-core approach, experiments were performed on the MRAHigh dataset on the Origin2000. For this scenario, the preprocessing time to generate the range partition files and the work assignment look-up tables is about 1054 seconds. Figures 6 and 7 show the isosurface extraction time using our hybrid granularity approach at isovalue 50 and 75, respectively, on each system configuration for 8, 12, 16, 24, and 32 processors. It is apparent that the load is very balanced for these isovalues. The speedup of our hybrid granularity approach appears linear.

We have also performed an experiment to compare the performance of our hybrid granularity parallel out-of-core approach with our previous single granularity [Zhang and Newman 2001], Bajaj et al.'s two-level decomposition [1999], and with Zhang et al.'s random data distribution [2002] approaches. For both the hybrid and single granularity schemes, the linear model in Equation 1 was used to estimate the work within each blocklet. However, the hybrid granularity scheme uses variable-sized blocklets and performs work estimation and work scheduling in the preprocessing stage,

	8 CPUs	12 CPUs	16 CPUs	24 CPUs	32 CPUs
Our hybrid granularity appr.	0.81	0.64	0.7	1.05	1.54
Our single granularity appr.	2.37	2.59	2.41	2.89	3.05
Two-level decomposition appr.	2.51	2.29	2.15	2.11	2.16
Random data distribution appr.	347.29	327.39	433.08	335.4	362.24

Table 2: Comparison of read time (in seconds) for hybrid granularity, single granularity, two-level decomposition, and random data distribution approaches for 8, 12, 16, 24, and 32 processors (MRAHigh volume).

while in the single granularity approach, fixed-sized blocklets were used and work estimation and work scheduling were performed during the parallel computation. Table 2 shows the time for the range partition file to be read for our hybrid granularity approach, our single granularity approach, the two-level decomposition approach and the random data distribution approach for 8, 12, 16, 24, and 32 processor configurations. Among these four approaches, our hybrid granularity approach achieves the best read performance and the random data distribution approach exhibits the worst read performance, since in the random data distribution approach there are many disk head movements due to assignment of non-consecutive blocklets on each processor.

Figures 8 and 9 show the isosurface extraction time and the number of triangles generated on each processor of a system with 24 processors using our hybrid granularity approach, our single granularity approach, the two-level decomposition approach and random data distribution approach for isovalue 50. Figure 10 shows the isosurface extraction time for each processor of a system with 24 processors using our hybrid granularity approach, our single granularity approach, the two-level decomposition approach and random data distribution approach for isovalue 75. In these figures, we can see that both our approaches achieve higher performance than does the two-level decomposition approach and than does the random data distribution approach. The extraction time of our hybrid granularity approach and single granularity approach is faster than that of the two-level decomposition approach and the random data distribution approach due to more balanced load across the processors and the use of interval-based data structure. Also, the extraction time of our hybrid granularity approach is better than that of our single granularity approach due to the lower work estimation and scheduling overhead in our hybrid granularity approach. For isovalue 75, the number of triangles in our approaches and the random data distribution approach is approximately equal on each processor, while in the two-level decomposition approach, the number of triangles varies and even reaches zero for some processors. This scenario occurs because in the two-level decomposition approach, the blocklets are determined to be similar if their ranges cover the same number of buckets (i.e., each of which is a small range of continuous isovalues). Even if two blocklets fall into the same bucket, there may exist one or more isovalues that only one of them intersects. However, for isovalue 50, the distribution of triangles generated by the random data distribution approach is more balanced than that by our hybrid granularity approach. This is caused by the random data distribution approach’s random assignment results in blocklets in a given region to be distributed to different processors. The hybrid granularity approach assigns blocklets based on an accurate work estimation model and groups nearby blocklets such that they are distributed to the same processor. If the distribution of triangles in the region is very sparse, it happens that the processor has a large number of blocklets, within most of which the distribution of triangles is very sparse, while other processors have a relatively small number of blocklets, within most of which the distribution of triangles is dense. For example, in our MRA dataset, for isovalue 50, in Figure 9 the number of triangles on the last two processors is less than that on other processors, but the

number of blocklets assigned to those two processors is more than that assigned to other processors. This scenario is determined by the characteristic of the dataset. In this case, our approach still can achieve very balanced load across the processors since our work estimation is based on two factors, instead of the number of triangles. In addition, the performance of the random data distribution approach depends highly on the random number generator and the characteristic of the dataset. It is possible for the random distribution to assign blocklets that have a dense distribution of triangles to some processors and blocklets that have a sparse distribution of triangles to other processors, resulting in imbalanced load.

In addition, we also have made an experiment to compare our hybrid granularity approach with the random data distribution approach using the interval tree to load into memory only active blocklets for a given isovalue. The experimental result shows that extraction time in our hybrid granularity approach is still 6% faster than that in the random data distribution approach. However, the read time for the random data distribution approach is much longer than ours. Therefore, our approach is very flexible to various kinds of datasets and robust to yield good load balancing.

Furthermore, we have also compared our hybrid granularity approach with Miguet and Nicod’s [1995] slice-based approach. Figures 11 and 12 compare extraction time (on configurations of 4 to 32 processors) for our hybrid granularity approach with the slice-based approach for isovalues 50 and 75, respectively. The results show that our approach is 3 times faster than the slice-based approach for isovalue 50 and 15 times faster for isovalue 75. Hybrid granularity is better due to its finer granularity, avoidance of inactive regions in the dataset, and employment of an accurate work estimation model.

5.5 Scalability

In order to validate the scalability of our hybrid granularity approach, a group of 8 MRA brain datasets of size $256 \times 256 \times 141$, $256 \times 256 \times 281$, $256 \times 256 \times 561$, $256 \times 256 \times 1121$, $256 \times 256 \times 1681$, $256 \times 256 \times 2241$, $256 \times 256 \times 3361$, and $256 \times 256 \times 4481$ were used to perform isosurface extraction on a system of 1, 2, 4, 8, 12, 16, 24, or 32 processors, respectively. Seven of the datasets are scaled-up versions of the first dataset; the work in those datasets is a multiple of the work in $256 \times 256 \times 141$ dataset. These datasets allow exact determination of the effects of a quantifiable increase in work on system performance. Then, we used the well-known Gustafson’s *scaled speedup* [1988] to compute the speedups of our hybrid granularity approach on those datasets mentioned above. Gustafson’s scaled speedup scales the size of a problem with the scale-up in the number of processors. Such an approach to measuring speedup limits the potential for caching artifacts to produce the illusion of super-linear speedup, as well as truly scaling the work, thus making measured speedups more accurate and more reflective of the additional processing capability afforded by parallel computation. Scaled speedup is the number of processors times the ratio of the time (T_1) taken by a single processor to the time (T_n) taken by n

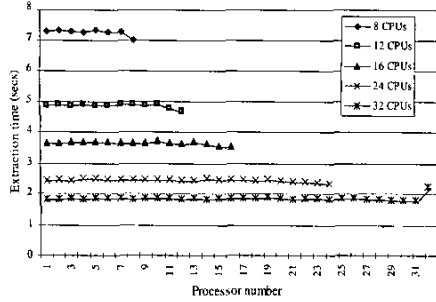


Figure 6: Illustration of load balancing on 8, 12, 16, 24, or 32 processors by our hybrid granularity approach (MRAHigh volume, isovalue 50).

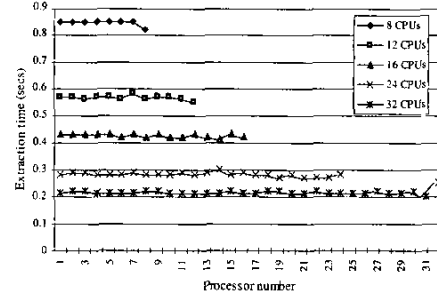


Figure 7: Illustration of load balancing on 8, 12, 16, 24, or 32 processors by our hybrid granularity approach (MRAHigh volume, isovalue 75).

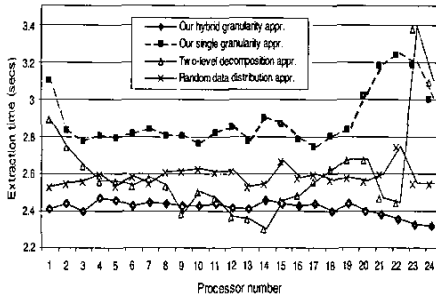


Figure 8: Comparison of isosurface extraction time across 24 processors among different approaches (MRAHigh volume, isovalue 50).

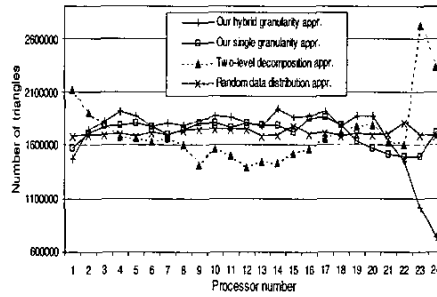


Figure 9: Distribution of triangles across 24 processors among different approaches (MRAHigh volume, isovalue 50).

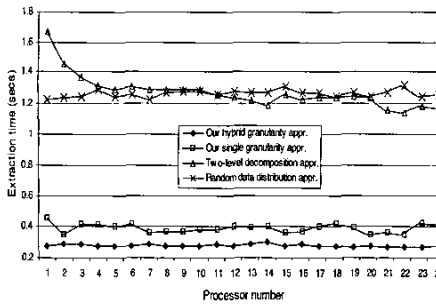


Figure 10: Comparison of isosurface extraction time across 24 processors among different approaches (MRAHigh volume, isovalue 75).

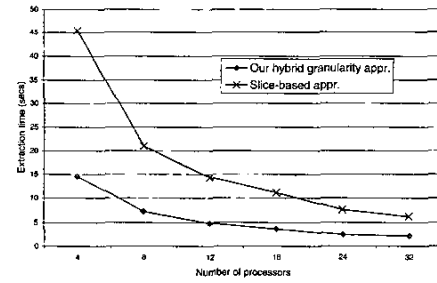


Figure 11: Comparison of extraction time between our hybrid granularity approach and Miguet and Nicod's slice-based approach (MRAHigh volume, isovalue 50).

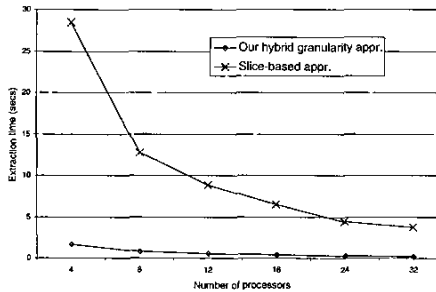


Figure 12: Comparison of extraction time between our hybrid granularity approach and Miguet and Nicod's slice-based approach (MRAHigh volume, isovalue 75).

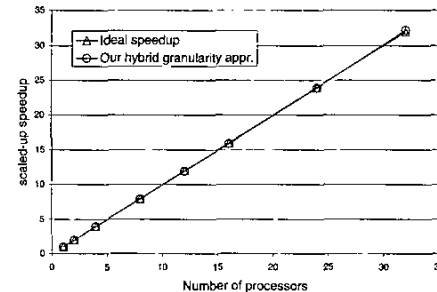


Figure 13: Scaled-up speedup for our hybrid granularity approach (solid line) vs. ideal (dashed line) (MRA volumes, isovalue 50).

processors, expressed as

$$\text{speedup}_{\text{scaled}} = \frac{n \times T_1}{T_n}.$$

In the ideal n -CPU case, each processor should finish its work in the same time as one processor working alone on the $256 \times 256 \times 141$ dataset. Figure 13 shows the observed speedups for configurations of 2 to 32 CPUs, which nearly match the ideal speedup. The linear speedup suggests that our approach scales well.

6 Conclusions

In this paper, we have presented a new approach to improve the total performance of parallel out-of-core isosurface extraction of large datasets and validated its performance against competing approaches. The approach is aimed at optimizing the disk I/O, reducing the memory requirement in out-of-core extraction and achieving load balancing for parallel computation. The approach decomposes the volume into blocklets, each of which is a multiple of disk block size. To reduce work duplication and to reduce storage requirements (i.e., arising from duplication of vertices on the boundaries between adjacent blocklets), variable-sized blocklets (formed by merger of adjacent, similar blocklets) are used. The approach minimizes secondary memory access and main memory pressure by loading into memory only blocklets that fall into a range partition. An accurate work estimation model based on the processing behaviour of the isosurfacing algorithm allows work to be quite evenly distributed among processors. The preprocessing of the data can also be performed without keeping the entire dataset in the main memory. Furthermore, the approach's use of an interval-based data structure allows non-active regions to be avoided during isosurface extraction.

However, since our hybrid granularity approach focuses on the achievement of load balancing for each isosurface extraction, there exists some overhead to reload small parts of blocklets or transmit small parts of blocklets between processors when extracting isosurface from one isovalue to another isovalue in a distributed environment or distributed shared-memory environment. In short, our hybrid granularity approach can obtain high performance for isosurface extraction and is especially suitable for multiple isovalue queries in shared-memory environments. By loading the portion of the dataset that is active for multiple isovalue queries in a range of values, load balancing across processors can be achieved during each isosurface extraction.

Acknowledgement: This work was partially supported by NSF grants ASC 97-02401 and ACI 02-22819. We also acknowledge a grant of computation time on the NCSA SGI Origin2000 and access provided by Professor R. Sass to Clemson University's cluster computer (for preliminary experiments). The MRA data used in our experiments was collected by the Cleveland Clinic Foundation.

References

BAJAJ, C. L., PASCUCCI, V., AND SCHIKORE, D. R. 1996. Fast isocontouring for improved interactivity. In *Proc., 1996 IEEE Symp. on Volume Vis.*, 39–46.

BAJAJ, C. L., PASCUCCI, V., THOMPSON, D., AND ZHANG, X. Y. 1999. Parallel accelerated isocontouring for out-of-core visualization. In *Proc., 1999 IEEE Parallel Vis. and Graphics Symp.*, 97–104.

CHIANG, Y.-J., SILVA, C. T., AND SCHROEDER, W. J. 1998. Interactive out-of-core isosurface extraction. In *Proc., Vis.'98*, 167–174.

CHIANG, Y.-J., FARIAS, R., SILVA, C. T., AND WEI, B. 2001. A unified infrastructure for parallel out-of-core isosurface and volume rendering

of unstructured grids. In *Proc., IEEE Symp. on Parallel and Large-Data Vis. and Graphics*, 59–66.

GREGORSKI, B., DUCHAINEAU, M., LINDSTROM, P., PASCUCCI, V., AND JOY, K. I. 2002. Interactive view-dependent rendering of large isosurfaces. In *Proc., Vis.'02*, 475–482.

GREVERA, G., UDUPA, J., AND ODHNER, D. 2000. An order of magnitude faster isosurface rendering in software on a pc than using dedicated, general purpose rendering hardware. *IEEE Trans. on Vis. and Computer Graphics* 6, 4, 335–345.

GUSTAFSON, J. L. 1988. Reevaluating amdahl's law. *CACM* 31, 5, 532–533.

HANSEN, C. D., AND HINKER, P. 1992. Massively parallel isosurface extraction. In *Proc., Vis.'92*, 77–83.

HOWIE, C. T., AND BLAKE, E. H. 1994. The mesh propagation algorithm for isosurface construction. *Computer Graphics Forum* 13, 3, 65–74.

LORENSEN, W. E., AND CLINE, H. E. 1987. Marching cubes: A high resolution 3d surface reconstruction algorithm. *Computer Graphics* 21, 4, 163–169.

MACKERRAS, P. 1992. A fast parallel marching-cubes implementation on the fujitsu ap1000. In *Tech. Report TR-CS-92-10*, Australian Nat'l Univ.

MIGUET, S., AND NICOD, J.-M. 1995. A load-balanced parallel implementation of the marching-cubes algorithm. In *Proc., High Performance Computing Symp.'95*, 229–239.

NEWMAN, T. S., AND TANG, N. 2000. Approaches that exploit vector-parallelism for three rendering and volume visualization techniques. *Computers and Graphics* 24, 5, 755–774.

PAYNE, B. A., AND TOGA, A. W. 1990. Surface mapping brain function on 3d models. *IEEE Computer Graphics and Applications* 10, 5, 33–41.

SHEKHAR, R., FAYYAD, E., YAGEL, R., AND CORNHILL, J. F. 1996. Octree-based decimation of marching cubes surfaces. In *Proc., Vis.'96*, 335–344.

WILHELMS, J., AND GELDER, A. V. 1990. Topological considerations in isosurface generation—extended abstract. *Computer Graphics* 24, 5, 79–86.

WILHELMS, J., AND GELDER, A. V. 1992. Octrees for faster isosurface generation. *ACM Trans. on Graphics* 11, 3, 201–227.

WOOD, Z. J., DESBRUN, M., SCHRÖDER, P., AND BREEN, D. 2000. Semi-regular mesh extraction from volumes. In *Proc., Vis.'00*, 275–282.

ZHANG, H., AND NEWMAN, T. S. 2001. High-performance mmd computation for out-of-core volume visualization. In *Proc., Supercomputing '01 (Poster Presentation Abstract)*.

ZHANG, X., BAJAJ, C., AND BLANKE, W. 2001. Scalable isosurface visualization of massive datasets on cots clusters. In *Proc., IEEE Parallel and Large-Data Vis. and Graphics '01*, 51–58.

ZHANG, X., BAJAJ, C., AND RAMACHANDRAN, V. 2002. Parallel and out-of-core view-dependent isocontour visualization using random data distribution. In *Proc., Joint Symp. on Vis. (VisSym'02)*, 9–18.