Eurographics Symposium on Parallel Graphics and Visualization (2004)
Dirk Bartz, Bruno Raffin and Han-Wei Shen (Editors)

Case Study of Multithreaded In-core Isosurface Extraction Algorithms

Huijuan Zhang,[1] Timothy S. Newman[1] and Xiang Zhang[1]

[1] Department of Computer Science, University of Alabama in Huntsville, Huntsville, AL 35899, USA

Abstract

A comparative, empirical study of the computational performance of multithreading strategies for Marching Cubes isosurface extraction is presented. Several representative data-centric strategies are considered. Focus is on in-core computation that can be performed on desktop (single- or dual-CPU) computers. The study's empirical results are analyzed on the metrics of initialization overhead, individual surface extraction time, and total run time. In addition, an analysis of cache behavior and memory storage requirements is presented.

Categories and Subject Descriptors (according to ACM CCS): I.3.6 [Computer Graphics]: Graphics data structures and data types

1. Introduction

Volume visualization can allow discovery of structures or phenomena from volumetric datasets. Indirect volume rendering is a volume visualization paradigm in which an intermediate representation is synthesized from the volumetric dataset and then rendered using traditional graphics means. Often, the intermediate representation is a surface of constant value (i.e., an isosurface). In many types of data, certain constant values (i.e., isovalues) define surfaces that correspond to the boundary of an object or structure of interest in the volume. One very popular isosurface extraction method is Lorensen and Cline's Marching Cubes (MC) algorithm [LC87]. In this paper, we consider the class of MC-based isosurface extractions in which the volume dataset and the isosurfacing's supporting data structures can be totally resident within the main memory (i.e., *in-core* isosurfacing).

Multithreaded computing is one high performance computing strategy. The availability of multithreading libraries allows development and portability of parallel programs for a range of low- and moderate-cost computers, such as popular dual-processor shared memory multiprocessing (SMP) systems. In this paper, we present a comparative study of several schemes for multithreaded in-core Marching Cubes isosurface extraction. The schemes considered are data-centric in that they seek to divide the isosurfacing work via partitioning of the volume among the threads. (Since Marching Cubes has an inherently serial ordering of its sub-tasks, its efficient parallelizations have been data-oriented rather than task-oriented.) The schemes' extraction performance, total computational performance (i.e., including time for I/O and initialization), cache behavior, and memory consumption are evaluated.

This paper is organized as follows. Section 2 describes related work. The multithreaded approaches to isosurfacing are described in Section 3. Experimental results and analysis are presented in Section 4. Section 5 contains the conclusion and future work.

2. Related Work

In order to achieve interactive or real-time MC-based isosurface extraction, often approaches that avoid some computations or that use parallel processing have been used.

2.1. Limiting Processing

The computation avoidance strategies typically exploit the fact that most isosurfaces only pass through a small portion of the volume. Such strategies attempt to avoid unnecessary processing in the portions of the volume that are not intersected by the isosurface. Such regions are termed *inactive*. In contrast, a region is *active* if it is intersected by the isosurface. Likewise, a cell (i.e., cube) of a volume dataset is called an *active cell* if it is intersected by the isosurface. Otherwise, a cell is called an *inactive cell*.

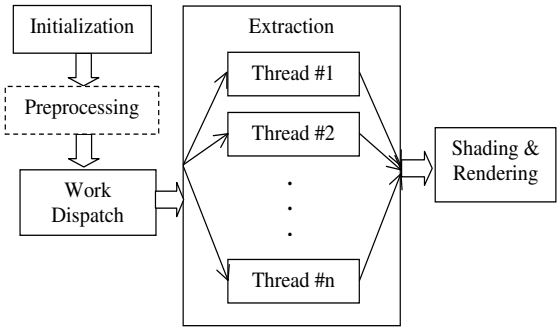


Figure 1: *Framework for multithreaded Marching Cubes algorithms.*

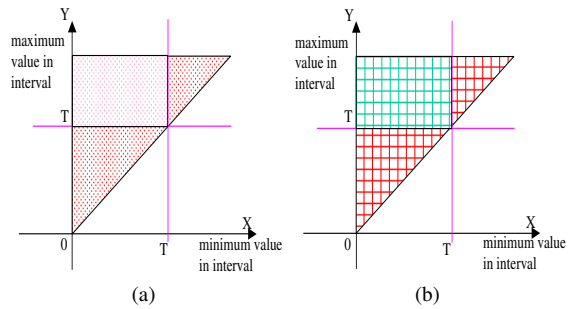


Figure 2: *Illustration of (a) span space and (b) span space bucket structure (adapted from [SG02]).*

Algorithms that accelerate isosurface extraction by avoiding unnecessary processing in inactive cells can be categorized into three groups [SG02]: hierarchical geometric space algorithms, range-based algorithms, and propagation algorithms, which we discuss next.

Hierarchical geometric space algorithms organize volume data in data structures based on the data's geometric space. Data structures of this type, such as octrees, group neighboring cells together. Wilhelms and van Gelder [WG92] were one of the first teams to accelerate isosurface extraction via use of octrees. They also introduced the branch-on-need octree (BONO), which we discuss in Section 3.2.

Range-based algorithms organize the data in data structures based on the range of values within each cell. Two data structures of this type are the interval tree [CMM*97] and Livnat et al.'s [LSJ96] span space which organize cells by minimum and maximum scalar values. In span space approaches, each cell is mapped to a point in 2D span space. Shen et al. [SHLJ96] have presented the Isosurfacing in Span Space with Utmost Efficiency (ISSUE) algorithm which uses a quantized span space. ISSUE restricts the scope of the search over the span space, making it faster than many other span space-based techniques.

Propagation algorithms first find a set of seed cells (i.e., that are active) and then propagate outward from the seed cells to find all other active cells. (At least one seed cell per connected component is required.) Propagation only visits the inactive cells that are connected to the active cells; other inactive cells aren't visited. The work of Shekhar et al. [SFYC96] and Bajaj et al. [BPS96] are examples of isosurfacing-by-propagation approaches. Sutton et al. [SHSS00] have compared the serial processing performance of one seed set propagation approach to other isosurface acceleration approaches, including approaches based on the BONO, the interval tree, and the span space.

2.2. Parallel Processing

Several parallel computation strategies to accelerate isosurfacing have been presented. For example, Newman and Tang [NT00] have reorganized the Marching Cubes' processing steps to allow the algorithm's inherent data parallelism to be exploited on a vector-parallel supercomputer. Some other methods have focused on dividing work evenly among processors via data-oriented subdivision. For example, Hansen and Hinker [HH92] have presented a SIMD Marching Cubes approach that uses cell-based partitioning (i.e., which assigns an identical number of cells for processing on each processor). Miguet and Nicod [MN95] have presented an MIMD Marching Cubes which divides the dataset's processing among processors in a slice-by-slice manner that attempts to balance workload on the CPUs. In their approach, the isosurface extraction time (i.e., workload) for each slice is estimated as a weighted sum of the number of cells and the number of interpolated vertices in the slice. Bajaj, Zhang, and colleagues [BPTZ99, ZBR02] have presented MIMD-parallel out-of-core isosurfacing approaches which divide processing among processors in a blocklet-by-blocklet manner that attempts to balance CPU loads. (Blocklets are small sets of adjacent cells.) In their approaches, workload is approximately related to the number of active blocklets. Previously, our team has presented an MIMD Marching Cubes approach that divides the dataset into many blocklets and then uses a very accurate work estimation model [ZN01] in distributing blocklets among processors such that each processor has about the same amount of work to do. Zhang and Newman [ZN03] have also presented a hybrid granularity scheme that aids in achieving reduced disk I/O and low total elapsed time for MIMD-parallel isosurfacing on a supercomputer.

However, only a few multithreaded isosurface extraction approaches—in particular, approaches that combine multithreading and computation avoidance—have been reported. For example, Bartz et al. [BSGE98] have presented a balanced multithreaded approach to octree (e.g., BONO) construction. In order to generate a load-balanced work distribution for isosurfacing, the octree is recursively traversed to locate active cells. The active cells are then distributed among threads in a round-robin manner. Sulatycke and Ghose [SG02] have presented a multithreaded in-core span space-based isosurfacing approach for SMPs that overlaps I/O with rendering computations.

3. Framework of Multithreaded In-core Isosurfacing

The comparative study of multithreaded isosurfacing presented here focuses on operation on dual-processor and hyperthreading single-processor computers, which are popular computers for desktop scientific computation. The isosurface extraction algorithm used in the study is the standard Marching Cubes (MC) algorithm. The MC processes a regular rectilinear volume (i.e., a set of scalar values distributed on a rectilinear grid) in a cell-by-cell fashion. If a cell is intersected by the isosurface, the Marching Cubes uses linear interpolation to estimate locations of isosurface intersection with the grid segments that bound the cell. A predefined look-up table is typically used to determine the isosurface topology in each cell. The output of the MC algorithm is a set of triangle facets that approximate the isosurface.

Fig. 1 shows the processing framework for our multithreaded Marching Cubes strategies. The framework is composed of initialization, preprocessing, workload dispatch, isosurface extraction, and rendering stages. In the initialization stage, the dataset is loaded into the memory. The strategies that use a span space or a BONO structure (described later) build the structure in the preprocessing stage. In the work dispatch stage, the work load of the Marching Cubes is estimated and the data is distributed among threads so that each thread obtains an almost equal amount of work. After work dispatch, each thread performs the Marching Cubes on the data it was assigned. The isosurface facets produced by the threads are then rendered by the graphics hardware.

Generally, in parallel processing, the goal is to divide work such that computational resources are best-utilized. When using multithreading parallelism, in the ideal case, threads cooperating on a task will not delay one another. One strategy to achieve synchronized cooperation among threads is *static load balancing*, which involves assigning a fixed amount of work to each thread before parallel computation begins. The schemes compared here use static load balancing.

We focus primarily on BONO-based and span space-based isosurfacing due to a prior study which suggests they have serial processing performance that is superior to other strategies [SHSS00]. We also consider other approaches in the comparison, however.

3.1. Span Space Structure

The use of the span space structure can greatly aid in efficiency of isosurface extraction by skipping processing of inactive regions. Fig. 2(a) illustrates the span space, which represents cell intervals (i.e., the minimum and maximum of the values stored at the cell's vertices) as points in a two-dimensional space. The point (x_i, y_i) in the span space represents a cell with interval $[x_i, y_i]$ (i.e., a cell with minimum x_i and maximum y_i). All cell intervals can be mapped onto span space points that lie above or on the diagonal line $X = Y$ (i.e., the shaded regions in Fig. 2(a)). A span space whose points represent *cell* intervals is called a *cell-based* span space. For an isovalue T, the active cells are those mapped into the span space region bounded by the lines $X = T$ and $Y = T$ (i.e., the lightly shaded rectangular region in Fig. 2(a)).

Since the datasets used in this study contain byte data, we used a quantized span space with unit-sized tiles (i.e., $(257 \times 256)/2$ tiles, as illustrated in Fig. 2 (b)). The tiles that correspond to active cells are called *active tiles* of the span space. A *bucket* is a collection of tiles with the same maximum value (i.e., a bucket is a row of tiles in the span space). A bucket is *active* if the maximum value corresponding to the bucket is greater than or equal to the isovalue.

An extension of the span space data structure, the block-based span space, can be used to organize data blocks [ZN04]. (By *block*, we mean a collection of adjacent cells in the volume dataset). In the block-based span space, each point represents a block interval.

3.2. BONO Structure

The BONO [WG92] is a space-efficient variation of the traditional octree. The BONO recursively subdivides the volume based on cell geometric positions. The subdivision is performed in such a way that the "lower" subdivision in each dimension of the volume covers the largest possible power of two cells. Each tree node (except the root) represents a subvolume of the volume. At each node, the extremes of the data in the subvolume associated with the node are stored. The subdivision process continues until subvolume size reaches a minimum. Each subvolume associated with the terminal node can be viewed to be a block. In cases where volume dimensions are not powers of two, such an approach makes tree traversal more efficient. Search of the BONO allows quick determination of the subvolumes that don't contain the isosurface, allowing isosurfacing to skip inactive regions.

3.3. Partitioning Schemes

In order to achieve a balanced load across the threads, data-centric static load balancing approaches require accurate estimation of the work in each data partition. To some extent, partition granularity determines work estimation accuracy; with coarsely granular division of data, it tends to be more difficult to form partitions with equal processing requirements. The eight data-centric partitioning schemes compared in this study differ in degree of data granularity and in how work is estimated. These partitioning schemes are described next.

(Span Space) Bucket-based Partitioning Scheme. The (span space) bucket-based partitioning scheme partitions the dataset's active *buckets* among the threads. The active buckets are determined by a search of a cell-based span space. The active buckets are divided across threads; each thread is assigned an approximately equal number of active cells. The bucket-based partitioning is exactly the strategy used in Sulatycke and Ghose's [SG02] multithreaded isosurface extraction.

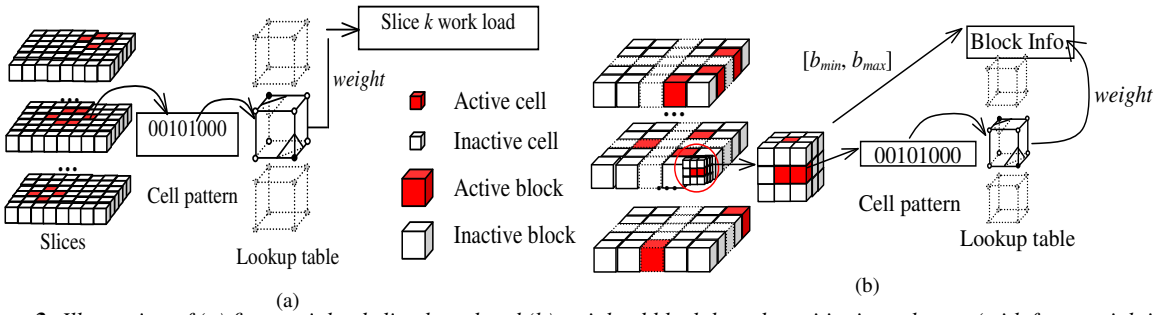


Figure 3: *Illustration of (a) finer weighted slice-based and (b) weighted block-based partitioning schemes (with facet weightings).*

(Span Space) Cell-based Partitioning Scheme. The (span space) cell-based partitioning scheme partitions the span space active tiles evenly among the threads. The active tiles are determined by a search of a cell-based span space [ZN04]. The active tiles are divided such that each thread is assigned an approximately equal number of active cells.

(Span Space) Block-based Partitioning Scheme. The (span space) block-based partitioning scheme partitions the (dataset's) *span space active tiles* evenly among the threads. The active tiles are determined by a search of a block-based span space [ZN04]. The active tiles are divided such that each thread is assigned a roughly equal number of active blocks. We consider this method because it organizes memory accesses in a way that can increase locality of reference.

Next, partitioning schemes which do not use the span space are described.

Slice-based Partitioning Scheme. The slice-based partitioning scheme partitions the dataset's slices evenly among the threads; each thread is assigned an approximately equal number of slices. Such a scheme is essentially a coarsely granular version of the partitioning scheme used by Hansen and Hinker [HH92] for their SIMD Marching Cubes.

Weighted Slice-based Partitioning Scheme. The weighted slice-based partitioning scheme (with active cell weighting) partitions the dataset slice-by-slice among the threads. Since the Marching Cubes requires more computations in some cells than others (e.g., active cells require more processing than inactive cells), the weighted slice-based partitioning computes an estimate of the isosurfacing work for each slice, and then each thread is assigned a set of slices requiring a roughly equal amount of isosurfacing computations. Each slice work estimate is simply that slice's active cell count (determined by searching cell-based span space). In this scheme, the cell-based span space is used only for active cell search in data partitioning, not for active cell search in isosurface extraction. This scheme is a multithreaded version of Miguet and Nicod's [MN95] MIMD approach with a variant estimate of work.

Finer Weighted Slice-based Partitioning Scheme. The finer weighted slice-based partitioning scheme (with facet weighting) enhances the work estimation of the weighted slice-based partitioning scheme. Specifically, the number of *isosurface* facets in a slice is taken as an estimate of the isosurfacing work for that slice.

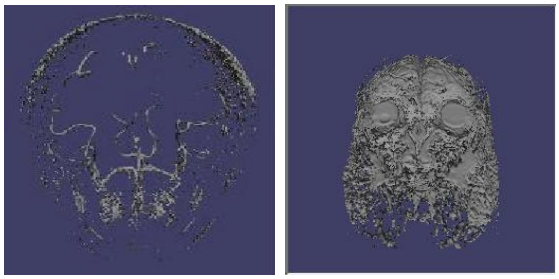
In this scheme, cells are processed slice by slice (as illustrated in Fig. 3 (a)). The scheme requires the active cells to be pre-determined and those cell's faceting topologies to be determined. The slices are then partitioned among threads such that each thread is assigned a set of slices requiring roughly equal isosurfacing computation. In addition, we aid the process by storing the cell topologic patterns in a table that is re-used in the triangle-generation stage. This scheme is a multithreaded variation of Miguet and Nicod's [MN95] MIMD approach with a variant estimate of work. We consider this method because it is designed to accurately estimate work. Its cost is a large amount of early processing to estimate work, however.

Weighted Block-based Partitioning Scheme. The weighted block-based partitioning scheme (with facet weighting), which we introduce here, uses the number of isosurface facets in the block as an estimate of isosurfacing work for that block. Four steps are involved to implement this scheme: first, the volumetric dataset is divided into blocks of size $m \times m \times m$; second, for each block b, the block's interval $[b_{min}, b_{max}]$ and the cells' topologic patterns are determined; third, the number of facets per cell is found by referral to the Marching Cubes topology lookup table (using the computed cell pattern as the index); last, each thread is assigned blocks with an approximately equal number of triangles. Fig. 3 (b) shows the process of this scheme.

BONO-based Partitioning Scheme. The BONO-based partitioning scheme partitions active terminal nodes (i.e., dataset's active blocks) evenly among the threads. The active blocks are determined by a search of a BONO. The active blocks are divided across threads so that each thread is assigned an approximately equal number of active blocks.

4. Experimental Results and Analysis

Experiments to test the performance of the eight partitioning schemes for multithreaded in-core MC isosurface extraction



(a) (b)

Figure 4: *Illustration of (a)* $\alpha = 75$ *isosurface for MRA Head and (b)* $\alpha = 30$ *isosurface for MRI Brain.*

were performed on two computers with 512 MB of RAM — one with dual Intel Xeon 2.4GHz CPUs and the other with one hyperthreading Intel Pentium IV 3.06GHz CPU. The Linux Pthreads library (version 0.10) running under a hyperthreading-supportive Linux 2.4 kernel was used to perform multithreading. Two datasets with 8-bit data items (a magnetic resonance angiography (MRA) head dataset (MRA Head) of size $256 \times 256 \times 72$ and a magnetic resonance imaging (MRI) brain dataset (MRI Brain) of size $256 \times 256 \times 128$) were used in the experiments. Sample isosurfaces extracted from the datasets are shown in Fig. 4, in which α represents isovalue. The isosurfacing processing includes the triangle vertex normal computation. The block-based strategies were tested using $4 \times 4 \times 4$ blocks.

On each computer, runs with one to eight threads were performed for each partitioning scheme. The reported extraction times are reported as trimmed means over 10 runs. The experiments reported here are for extractions at $\alpha = 75$ for MRA Head and at $\alpha = 30$ for MRI Brain. 1.3% of the cells are active for this MRA Head extraction. 5.1% of the cells are active for this MRI Brain extraction.

4.1. Dual-CPU Computation

This section reports the schemes' cache behavior, extraction performance, total performance, and the degree of load balancing for extractions performed on the dual-CPU computer.

4.1.1. Cache Behavior

First, we consider the L1 and L2 cache misses and TLB misses among the eight schemes for isosurfacing on the MRA Head dataset at $\alpha = 75$. (It is well-known that the gap between CPU and main memory speeds can be overcome to some degree by algorithms which make good use of cache [HP03].) Misses were determined using the PAPI [DLM*01] interface to the Pentium performance counters.

In Fig. 5, we show the results for the one thread case (which is reasonably representative). The figure shows the L1 and L2 cache misses and TLB misses for the eight schemes over the MRA Head dataset at $\alpha = 75$ on the dual-CPU machine.

The L1 and L2 cache misses were slightly higher for multi-threaded approaches than for uni-threaded ones. The L1 and L2 cache miss counts were essentially unchanged as the number of threads was increased beyond two. For each scheme, the number of TLB misses tended to increase marginally as the number of threads increased.

The L1 cache misses of the eight schemes varied, but the variation was not extreme—the maximum difference was a factor of about 2 among eight schemes. The bucket-based and cell-based schemes had substantially more L2 cache misses and TLB misses than the other schemes, however. The large number of TLB and L2 cache misses imply that these two schemes probably have low data locality of reference. Since a bucket or active tile can contain cells from any part of the volume, this outcome is perhaps not unexpected. In addition, these two schemes' more random pattern of memory accesses probably limits the CPU's predictive pre-fetching capability to accurately predict future cache accesses. In contrast, the slice-based, the weighted slice-based, and the finer weighted slice-based schemes had the fewest L2 cache and TLB misses, because they exhibit higher data locality of reference, and because their memory access patterns are more predictive, leading to more accurate predictive pre-fetching by the CPU. The block-based, weighted block-based, and BONO-based schemes had moderately high L2 cache misses, probably because these three schemes have at least a modest degree of locality of reference. However, the weighted block-based and BONO-based schemes had less TLB misses than the (span space) block-based scheme because the (span space) block-based scheme processes blocks based on the block-based span space. In the block-based span space, the blocks adjacent to each other in a tile are not necessarily stored in contiguous data locations, leading to lower data locality of reference than that in the weighted block-based and BONO-based schemes. The BONO-based scheme had more TLB misses than the weighted block-based scheme, probably because the BONO-based scheme processes only active blocks of the dataset and order of active block encounter in BONO traversal may not be the same as their memory order.

4.1.2. Extraction Performance

The isosurface extraction time of the schemes is reported in Fig. 6 for extractions with 1, 2, 3, 4, 6, and 8 threads for (a) the MRA Head and (b) the MRI Brain datasets. Use of multiple threads tended to decrease extraction time for all schemes. Of these schemes, the BONO-based scheme achieved modestly better extraction performance.

Comparison of Span Space-based Schemes. In most cases, the cell-based scheme outperformed the bucket-based scheme by a small amount, probably because the former employs the finer work assignment unit (i.e., tiles). Due to dividing active tiles rather than active buckets across threads, the cell-based scheme is more flexible than the bucket-based scheme and will perform better in the case where all active cells for a given isovalue are mapped into very few buckets.

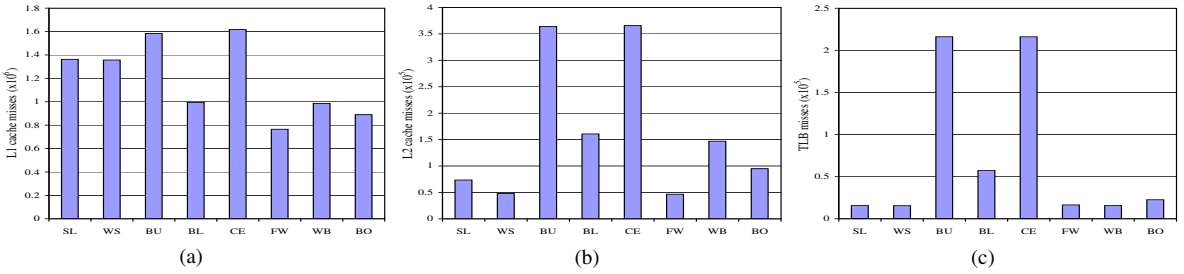


Figure 5: *Comparison of (a) L1 cache misses, (b) L2 cache misses, and (c) TLB misses among eight schemes (SL=Slice-based, WS=Weighted slice-based, BU=Bucket-based, BL=Block-based, CE=Cell-based, FW=Finer weighted slice-based, WB=Weighted block-based, BO=BONO-based) on dual-CPU machine for MRA Head dataset at* $\alpha = 75$*, using one thread.*

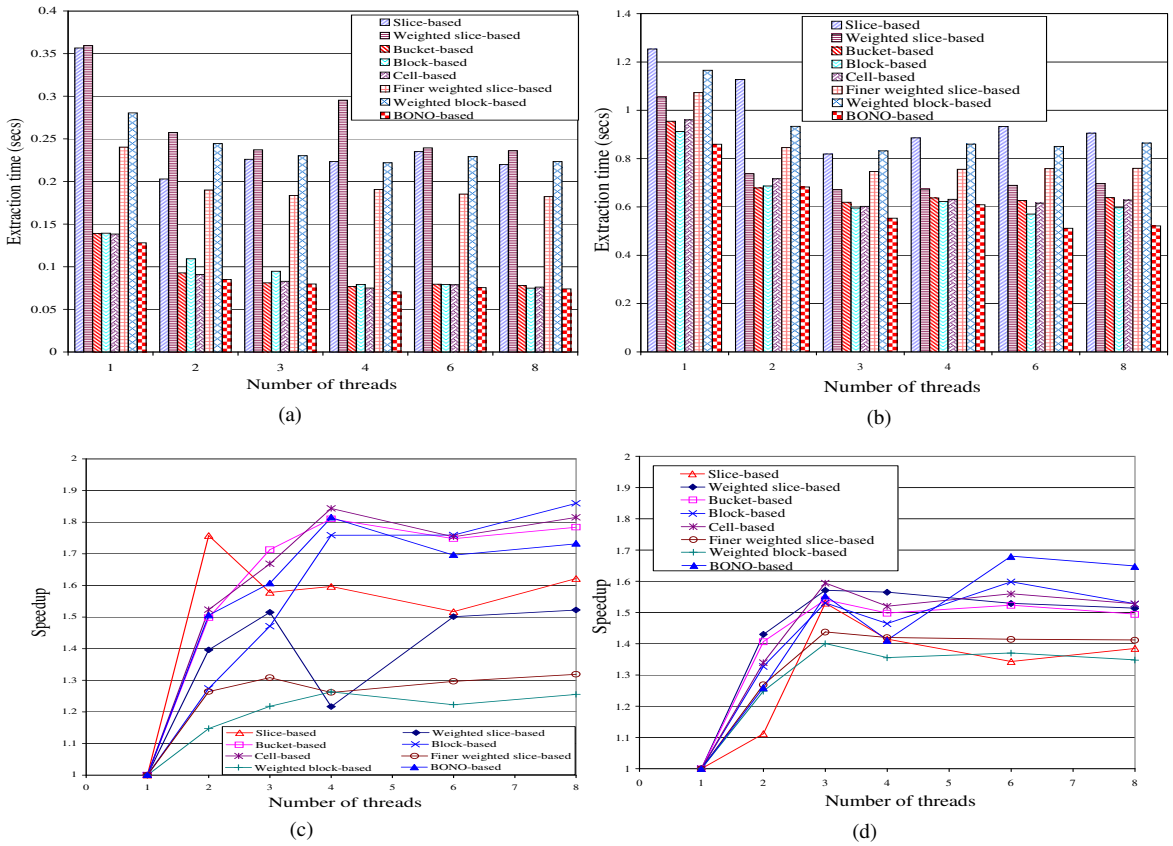


Figure 6: *Extraction times for (a) MRA Head dataset at* $\alpha = 75$ *and (b) MRI Brain dataset at* $\alpha = 30$ *and extraction speedups for (c) MRA Head dataset at* $\alpha = 75$ *and (d) MRI Brain dataset at* $\alpha = 30$ *on dual-CPU machine using various numbers of threads.*

Although the cell-based and the block-based schemes had quite similar performance for the MRA Head extraction at $\alpha = 75$, the block-based scheme performed better than the cell-based scheme for the MRI Brain dataset at $\alpha = 30$. This outcome is because there is a higher percentage of inactive cells in the MRA Head extraction, and the cell-based scheme more accurately eliminates the inactive cells from the processing; the block-based scheme skips a relatively smaller portion of the inactive cells. However, the block-based scheme can utilize the cache more effectively. Additionally, for the cell-based scheme, as the number of threads increases, there is likely relatively more conflict for memory per unit of time.

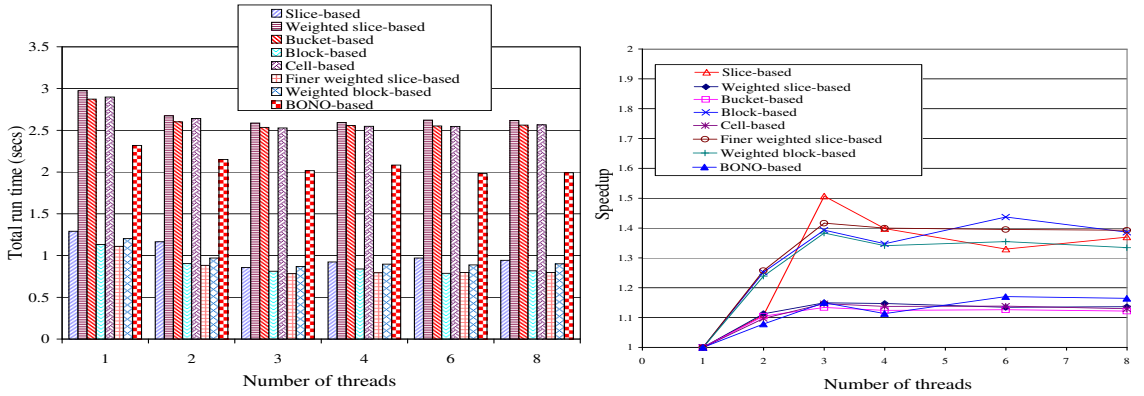


Figure 7: *Comparison of (a) total run time and (b) speedup of total run time on dual-CPU machine for MRI Brain dataset at* $\alpha = 30$*, using various numbers of threads.*

Comparison with other Schemes. For the $\alpha = 75$ isosurface on the MRA Head dataset, the performance of the slice-based scheme was better than that of the weighted slice-based scheme, in part because the distribution of the active cells is very unbalanced among slices. Thus, a thread assigned few slices each with many active cells may execute much more quickly than a thread with many slices each with few active cells; both may process an equal number of active cells but the second thread has many more total cells to process, making it run more slowly. For the extraction of the $\alpha = 30$ isosurface on the MRI Brain dataset, the distribution of the active cells was relatively balanced among the slices. Over all, estimating work using the number of active cells on each slice seems more accurate than estimation using the count of slices.

The finer weighted slice-based scheme also outperformed the slice-based scheme, probably due to its relatively more accurate estimate of work. However, whenever the finer weighted slice-based and weighted block-based schemes make a new isoquery, there is the higher overhead of re-estimating the work by traversing each cell to compute the number of isosurface facets. The finer weighted slice-based scheme did outperform the weighted block-based scheme, probably because the finer weighted scheme computes cell patterns for work estimation in a slice-based manner that can better-utilize the cache.

Overall Extraction Comparison. The BONO-based scheme likely outperformed the span space-based schemes because the BONO-based scheme might better-support locality of reference—nearby active blocks are more likely to be nearby in the octree, whereas they might have different extrema and be far apart in span space.

The span space-based schemes always outperformed the non-data-structure-based schemes because they exploit the advantages of both avoiding inactive cells and having the finer granularity (i.e., blocks or cells, rather than slices). However, the discrepancy of extraction times between span space-based schemes and non-data-structure-based schemes for $\alpha = 30$ on the MRI Brain dataset was less than that for $\alpha = 75$ on the MRA Head dataset, because the MRI Brain dataset extraction has a higher percentage of active cells.

Speedup Comparison among Eight Schemes. Fig. 6 shows the isosurface extraction speedups for the schemes for (c) the MRA Head and (d) the MRI Brain extractions. The block-based scheme achieved the best speedup of 1.86 using 8 threads (MRA Head dataset, $\alpha = 75$). The BONO-based scheme achieved the best speedup of 1.68 using 6 threads (MRI Brain dataset, $\alpha = 30$). This achievement likely occurred because these two schemes exhibit moderately high data locality of reference and employ a moderately fine data granularity that allows more accurate estimate and assignment of work.

4.1.3. Total Performance

We have also tested total isosurfacing performance, which includes the I/O and initialization overhead times as well as the extraction time for a single isoquery. Here, the initialization overhead is the time to generate the span space or the BONO (if necessary). Fig. 7 (a) shows the eight schemes' total run time for the MRI Brain dataset at $\alpha = 30$. The weighted slice-based, bucket-based, and cell-based schemes exhibited similar behavior and achieved the worst overall performance. Their relatively worse performance was due to use of the cell-based span space, which has a relatively long generation time. For the MRI Brain dataset, the cell-based and block-based span space generations took 1.86 seconds and 0.17 seconds, respectively, and the BONO generation took 1.36 seconds. The generation of the block-based span space is much faster than that of the cell-based span space due to the block-based span space's more coarse granularity (i.e., blocks are more coarsely granular than cells) and than that of the BONO due to the BONO's complex data structure (i.e., tree). In addition, the overall performance

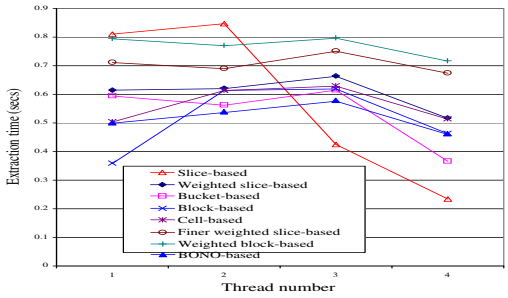


Figure 8: *Comparison of load balancing of eight schemes on dual-CPU machine for the MRI Brain dataset at* $\alpha = 30$*, with 4 threads.*

of the slice-based, finer weighted slice-based, and weighted block-based schemes is approximately on par with that of the block-based scheme due to those schemes having no overhead for span space generation. However, we must note that it is common to make multiple isoqueries, and the marginal time for each additional extraction, as mentioned earlier, is lowest for span space-based and BONO-based approaches.

Fig. 7 (b) shows the total performance speedups for the schemes. The results show that the slice-based scheme's speedup was best. That speedup was 1.51 for the MRI Brain dataset at $\alpha = 30$. Its good performance is because the slice-based scheme does not need to do as much serialized work (i.e., generation of the span space and work estimation), thus the parallelized extraction time is a relatively large component of the total run time. The bucket-based, cell-based, and BONO-based schemes exhibited the worst speedup, although their extraction times were low, due to computation avoidance.

Work in medical dataset processing [HSPK92] has suggested that practitioners will not tolerate delays of more than about 2 seconds for the production of renderings during interactive operation. The experiments reported in our previous work [ZN04] have demonstrated that conventional Marching Cubes and even single-threaded span space approaches are often slower than 2 seconds for extractions performed on tomographic datasets. However, the multithreading span space schemes allow the extractions to be reduced below 2 seconds. Thus, the multithreading span space-based and BONO-based schemes could be a great benefit to practitioners using in-core isosurfacing. It is apparent from the results in Fig. 7 that even for the moderate-sized datasets with a moderate degree of cell activity studied here, achieving an image in less than 2 seconds can be a challenge.

4.1.4. Load Balancing

Fig. 8 compares the extraction times for each thread in a 4-thread job for the MRI Brain dataset. The cell-based, finer weighted slice-based, and weighted block-based schemes could obtain a well-balanced load across threads. Among these three schemes, the weighted block-based scheme achieved the best load balancing. The percentage of imbalance (i.e., the relative time difference between the fastest and slowest thread) was 1.5% for the MRA Head dataset at $\alpha = 75$ and 10.1% for the MRI Brain dataset at $\alpha = 30$. The results suggest that the data granularity (i.e., block or cell) and accuracy of work estimation (i.e., using the number of active cells or triangles) are key factors influencing the load balance.

4.2. Single-CPU Computation

This section reports experiments performed on the single-CPU computer. For the MRI Brain dataset, the preprocessing times to generate the cell-based span space and the block-based span space were 1.65 seconds and 0.14 seconds, respectively; the preprocessing time to generate the BONO was 1.01 seconds. Due to space limits, we only report one group of experiments. Fig. 9 shows (a) the total run time and (b) the speedup in total run time using the eight schemes on the MRI Brain dataset at $\alpha = 30$. The relative behavior of the multithreading strategies is similar to that reported for the dual-CPU machine. However, the maximum speedup was about 1.2 — and use of more than 2 threads resulted in performance degradation.

4.3. Memory Consumption

Each strategy's memory requirement is the sum of the size of the input dataset, the output (i.e., the facetized mesh), and, if necessary, the span space or BONO data structure. Next, we estimate the memory requirement in a mathematical way, assuming the volume dataset with unsigned byte data items contains $N \times N \times N$ data points, where N is a power of two. Thus there are $(N-1)^3$ cells in the volume; the number of blocks (each of size of $4 \times 4 \times 4$ data points) should be about $\frac{(N-1)^3}{27}$ since each non-boundary block contains 27 cells. Storing the output facets' information requires G bytes. If the span space data structure uses 3 bytes to store the index of each unit (i.e., block or cell), the total storage should be at least $3(N-1)^3$ bytes for the cell-based span space and at least $\frac{(N-1)^3}{9}$ bytes for the block-based span space. Since N is a power of two, the BONO will contain approximately $\frac{1}{7}(N^3-1)$ nodes [WG92]. If the BONO uses 13 bytes to store necessary information for each node, the total storage should be at least $\frac{13}{7}(N^3-1)$ bytes. The other storage for supporting variables for any scheme is relatively small, and we assume it to be a constant b bytes for all the schemes.

Hence, total memory usage is about (N^3+G+b) bytes for the slice-based scheme, about $(N^3+G+3(N-1)^3+b)$ bytes for the weighted slice-based, bucket-based, and cell-based schemes, about $(N^3+G+\frac{(N-1)^3}{9}+b)$ bytes for the block-based scheme, about (N^3+G+N^3+b) bytes for the finer weighted slice-based and weighted block-based schemes, and about $(N^3+G+\frac{13}{7}(N^3-1)+b)$ bytes for the BONO-based scheme. (The finer weighted slice-based and weighted block-based schemes need N^3 bytes more memory than the slice-based scheme since these two schemes keep the cell patterns

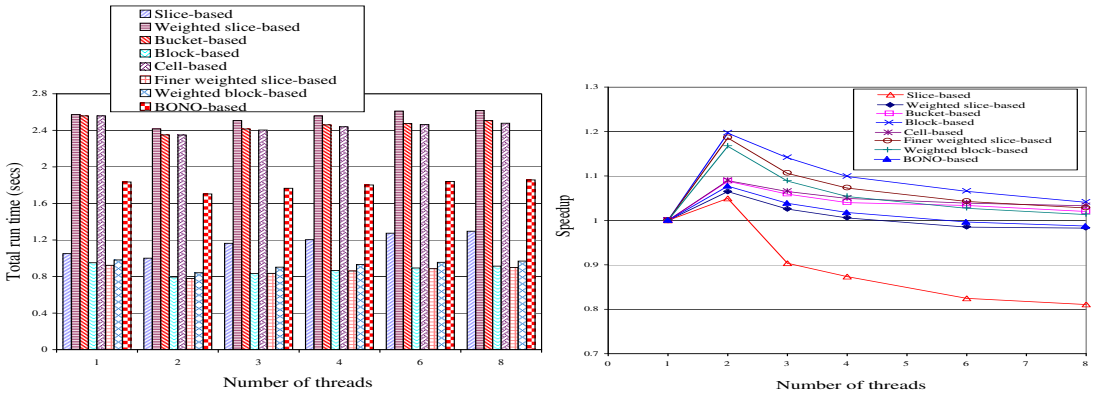


Figure 9: *Comparison of (a) total run time and (b) speedup of total run time among eight schemes on single-CPU machine for a MRI Brain dataset at* $\alpha = 30$*, using various numbers of threads.*

computed during the work estimation for later use in isosurface extraction.)

It is apparent that among three types of data structures (i.e., cell-based span space, block-based span space, and BONO), the cell-based span space needs the most memory storage, the block-based span space needs the least memory storage, and the BONO needs a moderate-sized memory storage. Of the eight schemes, the slice-base scheme requires the least memory since it uses no auxiliary data structure. The weighted slice-based, bucket-based, and cell-based schemes have nearly equal memory requirements, but use more memory than the other schemes. For many applications, the block-based scheme's memory requirement may be a good compromise since it's between that of the slice-based and the other six schemes. Thus, when the dataset is large, the block-based scheme can achieve high isosurfacing performance with less memory usage than the other schemes using auxiliary data structures.

5. Conclusion

In this paper, we have examined multithreading strategies for efficient Marching Cubes isosurface extraction on single- and dual-CPU computers. Eight data-centric multi-threading schemes' cache behavior, computational performance, and memory requirement were considered. Factors such as different data granularity (e.g., slices, blocks, and cells) and different work estimation strategies impact these schemes' performance.

Of the eight schemes tested, the slice-based, weighted slice-based, and finer weighted slice-based schemes exhibited the best cache behavior. The BONO-based scheme appeared to achieve the best isosurface extraction performance, however. The bucket-based, block-based, and cell-based span space schemes also had good isosurface extraction performance. The block-based scheme obtained the best speedup from multi-threading. The slice-based scheme produced the best speedup of total performance for a single isoquery due to its lack of overhead (no span space or BONO is generated). Its total performance using a given number of threads appears similar to the behavior of the block-based, finer weighted slice-based, and weighted block-based schemes, however. The weighted block-based scheme can achieve the best load balancing across threads due to its finer data granularity and accurate work estimation. While use of the cell-based and block-based span space structures aids in acceleration of isosurfacing due to avoiding traversal of inactive regions, if the number of isosurface facets tends to be large for a given isovalue, the performance of extraction might not be greatly improved by using the span space. In such a case, the BONO-based scheme can achieve good extraction performance due to high data locality of reference. In the case where many isoqueries are needed, the span space-based and BONO-based schemes can achieve better performance than other schemes due to avoiding traversal of inactive regions.

In summary, the (span space) block-based scheme appears to be fairly efficient scheme for multithreaded Marching Cubes on current generation 32 bit single- or dual-CPU systems, considering the metrics of cache behavior, computational performance, and memory requirement together. Its extraction performance is similar to that of the cell-based approach, but its memory requirement and set-up time are improved. In the future, we plan to parallelize the construction of the span space and BONO structures and extend our comparative study of multithreaded in-core Marching Cubes isosurfacing to consider the impact of parallel construction of auxiliary data structures.

Acknowledgements

This work was partially supported by NSF grant ACI 02-22819. The MRA and MRI datasets used in our experiments were collected by the Cleveland Clinic Foundation. We also thank Pa-

van Emani for providing the code of BONO construction and Dan Terpstra for help with cache performance monitoring.

References

[ARK03] AGARWAL A., ROWBERG A. H., KIM Y.: Fast JPEG 2000 decoder and its use in medical imaging. *IEEE Trans. on Information Technology in Biomedicine 7*, 3 (2003), 184–190. 10

[BPS96] BAJAJ C. L., PASCUCCI V., SCHIKORE D. R.: Fast isocontouring for improved interactivity. In *Proc., 1996 IEEE Symp. on Volume Visualization* (1996), pp. 39–46. 2

[BPTZ99] BAJAJ C. L., PASCUCCI V., THOMPSON D., ZHANG X. Y.: Parallel accelerated isocontouring for out-of-core visualization. In *Proc., 1999 IEEE Parallel Vis. and Graphics Symp.* (1999), pp. 97–104. 2

[BSGE98] BARTZ D., STRAßER W., GROSSO R., ERTL T.: Parallel construction and isosurface extraction of recursive tree structures. In *Proc., the Sixth Int'l Conf. in Central Europe on Computer Graphics and Visualization (WSCG'98)* (1998), pp. 479–486. 2

[CMM*97] CIGNONI P., MARINO P., MONTANI C., PUPPO E., SCOPIGNO R.: Speeding up isosurface extraction using interval trees. *IEEE Trans. on Visualization and Computer Graphics 3*, 2 (1997), 158–170. 2

[DLM*01] DONGARRA J., LONDON K., MOORE S., MUCCI P., TERPSTRA D.: Using PAPI for hardware performance monitoring on linux systems. In *Conference on Linux Clusters: The HPC Revolution* (2001). 5

[HH92] HANSEN C. D., HINKER P.: Massively parallel isosurface extraction. In *Proc., Visualization'92* (1992), pp. 77–83. 2, 4

[HP03] HENNESSY J. L., PATTERSON D. A.: *Computer Architecture: A Quantitative Approach*, third ed. Morgan Kaufmann, 2003. 5

[HSPK92] HAYNOR D. R., SMITH D. V., PARK H. W., KIM Y.: Hardware and software requirements for a picture archiving and communication system's diagnostic workstations. *J. Dig. Imag. 5*, 2 (1992), 107–117, as referenced by [ARK03]. 8

[LC87] LORENSEN W. E., CLINE H. E.: Marching Cubes: A high resolution 3D surface reconstruction algorithm. *Computer Graphics 21*, 4 (1987), 163–169. 1

[LSJ96] LIVNAT Y., SHEN H.-W., JOHNSON C. R.: A near optimal isosurface extraction algorithm using the span space. *IEEE Trans. on Visualization and Computer Graphics 2*, 1 (1996), 73–84. 2

[MN95] MIGUET S., NICOD J.-M.: A load-balanced parallel implementation of the Marching-Cubes algorithm. In *Proc., High Performance Computing Symp.'95* (1995), pp. 229–239. 2, 4

[NT00] NEWMAN T. S., TANG N.: Approaches that exploit vector-parallelism for three rendering and volume visualization techniques. *Computers & Graphics 24*, 5 (2000), 755–774. 2

[SFYC96] SHEKHAR R., FAYYAD E., YAGEL R., CORNHILL J. F.: Octree-based decimation of Marching Cubes surfaces. In *Proc., Visualization'96* (1996), pp. 335–344. 2

[SG02] SULATYCKE P. D., GHOSE K.: Multithreaded isosurface rendering on SMPs using span-space buckets. In *Proc., Int'l Conf. on Par. Proc. (ICPP'02)* (2002), pp. 572–580. 2, 3

[SHLJ96] SHEN H.-W., HANSEN C. D., LIVNAT Y., JOHNSON C. R.: Isosurfacing in span space with utmost efficiency (ISSUE). In *Proc., Visualization'96* (1996), pp. 287–294. 2

[SHSS00] SUTTON P. M., HANSEN C. D., SHEN H.-W., SCHIKORE D.: A case study of isosurface extraction algorithm performance. In *Proc., Joint Symp. on Vis.* (2000), pp. 259–268. 2, 3

[WG92] WILHELMS J., GELDER A. V.: Octrees for faster isosurface generation. *ACM Trans. on Graphics 11*, 3 (1992), 201–227. 2, 3, 8

[ZBR02] ZHANG X., BAJAJ C., RAMACHANDRAN V.: Parallel and out-of-core view-dependent isocontour visualization using random data distribution. In *Proc., Joint Symp. on Vis.* (2002), pp. 9–18. 2

[ZN01] ZHANG H., NEWMAN T. S.: High-performance MIMD computation for out-of-core volume visualization. In *Proc., Supercomputing'01 (Poster Presentation Abstract)* (2001). 2

[ZN03] ZHANG H., NEWMAN T. S.: Efficient parallel out-of-core isosurface extraction. In *Proc., IEEE 2003 Symp. on Parallel and Large-data Visualization and Graphics (PVG'03)* (2003), pp. 9–16. 2

[ZN04] ZHANG H., NEWMAN T. S.: Span space data structures for multithreaded isosurfacing. In *Proc., IEEE SoutheastCon 2004* (2004), pp. 290–296. 3, 4, 8