Eurographics Symposium on Parallel Graphics and Visualization (2006)
Alan Heirich, Bruno Raffin, and Luis Paulo dos Santos (Editors)

Piggybacking for More Efficient Parallel Out-of-Core Isosurfacing

Timothy S. Newman [1] and Wenjun Ma[1]

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

Abstract

A scheme for improving the efficiency of parallel isosurfacing for very large datasets is presented. The scheme is aimed at improving performance in multi-processor environments, especially for environments in which inter-processor communication limitations become a bottleneck, such as when the number of processors can scale up without commensurate scale up in inter-processor communication bandwidth. The scheme enables load-balanced computation while also limiting unnecessary communication between processors through the use of communication piggybacking and interleaving. Empirical results are also presented and suggest that the scheme reduces communication by about 15% and overall isosurfacing time by about 13% over a highly efficient non-piggybacked parallel isosurfacing approach.

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

1. Introduction

Visualization techniques are often used to aid the process of information discovery from scientific or medical data. One of the most popular of the visualization techniques [Ano05] is the Marching Cubes isosurfacing [LC87, Nie03]. Isosurfacing involves the extraction and display of a surface of constant value. Typically, isosurface-aided discovery processes involve performing a series of isoqueries (i.e., extractions of surfaces of different constant values). A common goal for the series of isoqueries is to discover structures or phenomena captured in the dataset.

The standard Marching Cubes (MC) approach is applicable to volumetric data organized as scalar values sampled on a rectilinear lattice. Such datasets, which include Computerized Tomography (CT) and Magnetic Resonance Imaging (MRI) data, are widely used. The MC processes the dataset in a cube-by-cube fashion to produce an approximate isosurface in the form of a collection of triangular facets.

Various methods for improving the performance of the Marching Cubes and other isosurfacing mechanims have been described in the literature. While methods such as the standard Marching Cubes can be performed relatively quickly on desktop PCs and workstations for small datasets, achieving fast performance for large datasets can be a challenge. Parallel approaches to isosurfacing have been explored as one means for fast performance. However, there are a number of challenges that limit the performance of parallel isosurfacing, especially for large datasets. A chief challenge is effectively handling the input data and the produced facets, especially for datasets too large to be processed entirely in-core. In-core processing means that the memory required for the isosurfacing is less than the amount of resident RAM. The standard approach to Marching Cubes can require out-of-core processing for large datasets, which causes at least some components of processing to slow by several orders of magnitude once the storage requirements exceed the amount of RAM. This slow-down is primarily due to the secondary storage (i.e., virtual memory) having a higher latency than the primary storage (i.e., RAM). A second performance challenge that is endemic to multi-processor realizations of the MC is that the amount of interprocessor communication (IPC) scales up as problem size scales up, yet the network is a shared resource and its capacity, like RAM, is usually fixed. Thus, the shared resource becomes a performance bottleneck that serializes components of MC processing. While parallel approaches optimized for out-of-core processing of large datasets have been developed, none of the published approaches have addressed IPC-related bottlenecks. IPC bottlenecks are particularly limiting for deployments for the quite popular cluster computing environments since those environments nearly always have fixed inter-processor communication (IPC) bandwidths.

In this paper, a new scheme that addresses both the out-of-core processing challenge and the IPC bandwidth limitations without sacrificing computational performance is introduced. Although the scheme can improve performance on any multi-processor platform, it is aimed primarily at improvement of isosurfacing on cluster computers. The paper is organized as follows. Related work is described in Section 2. The new scheme is presented in the Section 3. Results of applying the scheme and an analysis of those results are described in the Section 4. The conclusion is in Section 5.

2. Background

Many of the attempts to achieve fast isosurfacing have involved optimization of Marching Cubes-like isosurfacing. In MC, the dataset is processed cube-by-cube in sequential, forward-marching order, where the i-th cube is the subset of the lattice bounded by points (x_i, y_i, z_i) and $(x_i + 1, y_i + 1, z_i + 1)$. For each cube that is *active*, that is, intersected by the isosurface, a high resolution triangulation that approximates the isosurface within the cube is constructed. The construction process involves first *marking* lattice points whose scalar values are greater than or equal to the isovalue. The triangular facet vertices are located on cubes edges that are terminated by one marked and one unmarked lattice point. There are 23 unique triangulation topologies that can be formed if cubes that are rotationally symmetric are held to be topologically equivalent [Nie03]. The 23 topological cases are shown in Figure 1. Typically, these topologies are encoded in a look-up table (LUT) to support quick determination of each active cube's triangulation.

2.1. Related Work

Approaches for improving isosurface performance include techniques that avoid regions of space containing no active cells, techniques that use parallel computation, and techniques that do both. Since our focus is improvement of parallel computation, we focus on techniques that include such a component.

2.1.1. Activity Determination

Isosurfacing performance can be improved by avoiding processing in regions of the dataset that are not active. Active region determination has been considered using a number of means, including approaches based on the octrees and encodings of intervals, such as the span space [LSJ96] and the interval tree.

2.1.2. Parallel Computation

Most of the parallel approaches for Marching Cubes-based isosurfacing have used data-parallel strategies since each cube can be processed independently of other cubes. Task-parallel strategies are not as attractive since processing each cube involves a sequence of activities in which most steps in the sequence are dependent on the results of the preceding step. For multi-processor environments, the data-parallel processing typically involves a final phase of assembling each processor's output on a "master" CPU. The master CPU is simply the CPU that is responsible for sending the scene description (that is, the isosurface facets) to the graphics head for display. This assembly phase is where the primary IPC bottleneck occurs.

The existing approaches attempt to achieve load-balanced computation on the individual processors using various schemes. For instance, estimates of computation for each layer of a dataset have been used to divide layers of a dataset among processors such that each processor has an approximately equal amount of work [MN95]. Load-balancing on a blocklet basis [BPTZ99] has also been used. A blocklet is a small set of contiguous cells. The blocklet-based approaches have included strategies that address out-of-core concerns as well as accurate workload balancing [ZN03]. One recent study has suggested that blocklet-based processing can allow better parallel performance than other schemes for dividing the dataset [ZN04].

In blocklet-based processing, the isosurface can be represented with moderate compactness by first storing the set of facet vertex locations followed by storing the set of facet boundaries. These boundaries can be stored as the IDs of the vertices of each facet. Such a representation mechanism is effective for blocklets because most facet vertices are shared among four cubes of the blocklet; storing each vertex location only once avoids unnecessary duplication. Moreover, since blocklet sizes are typically no larger than 16^3 in size [ZN03], the vertex IDs can usually be represented in a small number of bits (e.g., at most 16 bits each).

A few other parallel, out-of-core isosurfacing methods have also been described. Like the methods in [BPTZ99, ZN03], those methods have sought to reduce the amount of data loaded and to organize processing in ways that limit the need to access data multiple times (e.g., using an on-disk interval tree of clusters of cells [CFSW01]).

Multi-threaded span-space approaches have also been used for in-core isosurfacing (e.g., [ZN04]), including a method that avoids distribution of unnecessary data to threads [NSG04]. Span-space approaches organize the cubes based on the range (i.e., the span) of scalar values contained in them. Isosurfacing using span-spaces considers only the cubes whose spans include the isovalue.

An IPC bottleneck in facet assembly is inherent in virtually all of the parallel approaches. (Approaches in which

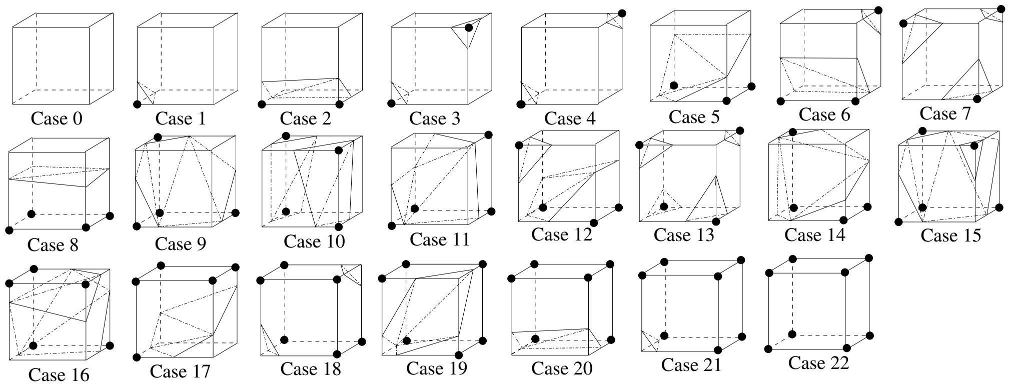


Figure 1: *The 23 Topologies in Marching Cubes*

each CPU independently drives one display on a display wall sometimes do not experience this bottleneck.) Essentially, the problem is that the triangles must be "funneled" single-file to the CPU connected to the graphics head. As problem size scales up, not only does the communication network become clogged with facet broadcasts, but the compositing CPU is reduced to mostly performing the compositing activity; the master CPU becomes unable to meaningfully participate in the isosurface extraction computations. In fact, nearly all load-balanced isosurfacing approaches have sought to have all slave CPUs complete their work at the same moment—the blocklet-based approaches, for example, have achieved outstanding success in synchronizing work completion among the CPUs. As a result, there is underutilization of the communication channel during computation as well as serialization during facet assembly. Thus, a key to further improvement of isosurfacing's performance in parallel environments is to reduce the amount of time spent in the serialized funneling stage. Although approaches to balance network transmission activities have been a focus in the grid communities (e.g., [YSF05]), we are unaware of of such approaches being used in parallel out-of-core isosurfacing.

2.1.3. Reducing Size of Output

Three types of methods for producing a smaller isosurface mesh have been presented. The first class of methods yields simplified meshes (e.g., [MSS94]). We do not consider simplification-based approaches further since we desire isosurfaces that are qualitatively equivalent to standard MC meshes. Another class of methods produces only the isosurface facets that are visible (e.g., [GS01]). Such approaches have some limitations, including not allowing exploration of all of the isosurface at one time and exhibiting poor performance if the viewpoint is changing since each change requires recomputation of visibility and re-isosurfacing. A third class of methods yields compressed meshes (e.g., [LNW04]). Such approaches have been popular in distributed isosurfacing applications. Since most of the compression methods impose a substantial computational overhead, they are generally not suitable for multiprocessor environments, however. Specifically, they are ill-suited because (1) the compression computations constitute extra computational overhead and (2) the computation to decompress the isosurface for display can be a substantial non-parallelized activity. In parallel isosurfacing, requiring the CPU connected to the graphics head to perform the decompression increases the degree of serialization, leading to rapid performance degradation as work scales up.

The facet compression approach of Lakshmipathy et al. [LNW04] follows a strategy that yields a mesh that can be straightforwardly decompressed if necessary. The approach involves merging connected facets in adjacent cubes into triangle strips. Its use of triangle strips yields a more compact isosurface representation since some vertex connection information need not be stored. One cost of the representation is that the isosurface extraction process cannot proceed independently in each cube. In addition, it must maintain supporting data structures to allow merger of triangle facets during isosurface extraction, for example to allow merging of triangles in adjacent cube layers. These supporting data structures increase memory pressure, making the approach ill-suited for out-of-core applications. Use of them also requires additional computation to check for and then achieve joins between initially disconnected strips. The Lakshmipathy et al. approach uses a different LUT strategy than the standard MC; the approach uses an edge LUT to allow facet vertices to be stored in a regular manner that nicely supports creation of triangle strips. The scheme that we introduce here is motivated by the Lakshmipathy et al. approach.

3. Description of Approach

Our scheme uses span space-based processing and an accurate static load-balancing mechanism to achieve efficiency in parallel computation. The scheme is also able to achieve improved overall performance via communication piggybacking and interleaving that effectively reduce the degree of serialization in parallel isosurfacing. The details of the scheme are described next.

The scheme employs static load-balancing similar to the efficient blocklet-based parallel out-of-core approach described by Zhang and Newman [ZN03], which builds on the earlier Bajaj et al. [BPTZ99] work. Like these other static load-balancing approaches, in our scheme the dataset is first processed off-line to create a set of partitions of the data. These partitions are organized on-disk in a manner that allows them to be quickly retrieved when needed. Each partition is associated with a range of isovalues. All chunks (a chunk is a set of adjacent blocklets) with a span that overlaps a particular partition's range are stored in the partition. Whenever an isosurface extraction is needed, only the partitions that contain the active chunks for the isovalue of interest are loaded into memory. Organizing data access to minimize I/O between the CPUs and secondary storage aids greatly in achieving good performance for out-of-core problems. The loaded chunks are then divided on a blocklet-by-blocklet basis among the CPUs of the system. An accurate estimate of isosurfacing workload for each blocklet, described later in Section 3.2, is used to assign approximately equal amounts of work to each CPU. Each CPU buffers its results and each buffer is periodically emptied and sent across the IPC network to a master CPU that composites the results for display on its graphics head.

The scheme is aimed at out-of-core isosurfacing on cluster computers composed of general-purpose CPUs interconnected by a fast but off-the-shelf interconnection network (i.e., rather than a custom inter-processor switching network such as is used in some supercomputers). The scheme is also suitable for high-end multiprocessing environments, including supercomputer environments, however.

Next we describe the new aspects of the scheme, namely its means to reduce the serialization imposed by the communication channel.

3.1. Reducing IPC: Piggybacking

The scheme's primary means to reduce the serializing effect of limited IPC bandwidth is to form triangle strips on pairs of adjacent cubes. In parallel isosurfacing, this practice leads to communication piggybacking as the description of the isosurface in the second cube of a pair is piggybacked to the description of the first cube; the piggybacked facets are transmitted as a unit. The use of piggybacking essentially reduces inherent redundancy in the isosurface mesh by not generating and transmitting unneeded vertex connectivity information. Since most parallel MC isosurfacing approaches have followed data-parallel approaches and since the MC itself is organized to operate on each cube independently, it has been natural for parallel isosurfacing approaches to compartmentalize each cube's processing. Piggybacking partially breaks the compartmentalization paradigm. Thus, it must be used in a way that does not negatively impact the data parallelism.

Our piggybacking is restricted to pairs of cubes. This restriction allows formation of a compressed isosurface representation without much additional computational overhead since the strips are formed between successive cubes encountered during the MC's sequential forward march through each blocklet. In addition, the restriction eliminates the need to maintain data structures that allow for initially disparate triangle strips to be merged, especially between dataset layers. Avoiding such data structures results in our scheme imposing negligible memory pressure, making it well-suited for out-of-core application. Moreover, restricting the strips to adjacent cubes allows processing without backtracking to a cube that was processed previously. (3-cube strips can require the vertex order from the first cube to be reordered, especially when the sheet of facets "folds" back into the first cube and merges with a previously disconnected sheet there.) Without reordering, the strips vertices could be ordered in a way that prevents proper rendering. In addition, in our approach the strips sent to the master CPU are ready to be rendered immediately, resulting in negligible additional computational pressure on the master.

The piggybacking proceeds as follows. As blocklet-based isosurfacing is carried out, whenever a cube is detected to have an isosurface intersection on the face it shares with the next cube to be visited in the march, information for triangle strip merger is saved. This information includes the number of facets that intersect the face and the edges on that face that contain the facet vertices. When the next cube is visited, one of its triangle strips is merged with one of the strips in the preceding cube. Whenever such a pair of cubes is found, the triangle strip can be formed directly from the cubes' look-up table information and the retained information about their shared face. Since we restrict the triangle strips to two cubes, each time the second cube of the pair is encountered, a new search begins for the next first cube of a new pair. To keep the amount of data transmitted to a minimum, we store each vertex coordinate as an 8B tuple—because each vertex is on a cube edge, its 3D coordinate must include 2 integer components and one floating point value. The integers can be stored in 2B and the floating points can be stored in 4B. The vertex ID information can be stored using a 4B value for the first vertex's actual ID. IDs of subsequent vertices in the strip can be stored as 2B offsets from the first ID. Since the piggybacking links adjacent cubes, the vertices forming the strip tend to be confined to a small region of the file as they were encountered either in processing the preceding cube, the adjacent cube in the preceding row, or the adjacent cube in the preceding layer.

Each piggybacking reduces the amount of information transmitted to the master CPU due to removal of unnecessary connectivity information. The savings come from the redundancy involved in storing each vertex that lies on the cube face shared by the adjacent cubes. Such vertices are stored with the facet information for each of the adjacent cubes. For example, consider a strip that can be formed on two adjacent Case 8 cubes. Standard blocklet-based operation would require storing 4 facet vertices from the first cube and 2 more facet vertices from the second cube. The facets for each cube would be represented as 4 vertex IDs, with each set of IDs defining a triangle strip in one cube; 8 IDs would be stored. Piggybacking the connectivity information across both cubes allows the two-cube strip's connectivity to be encoded using a total of 6 vertex IDs. Each time piggybacking can be used, 2 less IDs need to be stored.

From empirial study on many datasets, we have found that it is typically the case that half of the information transmitted to the master CPU is vertex connectivity information. We have observed that using piggybacking reduces the transmitted connectivity information by 15 to 30%, leading to an overall IPC savings of about 10 to 15%.

Our formation of piggybacked triangles employs a cube-based topology look-up table, like the MC's standard 23-case look-up table shown previously in Figure 1. Use of such a table allows quick and consistent strip formation based directly on the look-up table entries. However, our approach does slightly modify the standard table's facetization pattern for two of the cases, Cases 16 and 19. The modification involves treating these cases, each of which has multiple facets connected in a single sheet (i.e., neither has disconnected groups of facets), as if they were two disconnected sheets. These cases have two facet edges on one cube face, creating the potential for an adjacent cube with disconnected facet sheets (such as cubes of the Case 3, 6, 7, 10, 12, 13, or 15 types) to use the cube as a "bridge" between its disconnected facet sheets. Forming such a bridge would call for detecting this special situation and rearranging the vertices to allow for it. Such complex processing overhead is not worthwhile since Cases 16 and 19 rarely occur (one study suggests that they comprise much less than 1% of the active cubes [NH91]); allowing a bridge to be formed will result in negligible further reduction in communication while increasing computation. In fact, we have found that allowing any of the cubes with disconnected facet sheets to be members of two different triangle strips results in just a 3% additional savings in IPC from the extra piggybacking. This small savings is accompanied by extra computational complexity that yields slower overall parallel isosurfacing time despite the reduction in IPC serialization.

3.2. Work Estimation and IPC Interleaving

Prior blocklet-based parallel out-of-core methods can enable good (e.g., [BPTZ99]) or excellent (e.g., [ZN03]) balancing of isosurfacing computational work among the slave CPUs. In the Zhang and Newman [ZN03] work, it was shown that the isosurface extraction effort C for a blocklet can be well-estimated by fitting a linear function of the number of cubes (N) and the number of active cells (A) in the blocklet. Thus, workload, C, (where C has units of seconds and expresses the time for isosurface extraction's computations) can be estimated as $C = \alpha N + \beta A$. Blocklets can then be assigned to the CPUs such that each CPU is assigned approximately the same amount of work. However, as mentioned above, if all CPUs complete their work at the same time, the communication channel will be under-utilized.

Next, we describe a new workload estimation mechanism that considers isosurfacing computational workload as well as communication effort to produce a more optimal system utilization which allows better overall performance than the best prior estimation method ([ZN03]). The mechanism better-utilizes the communication channel throughout the isosurface extraction process by scheduling one slave CPU at a time for transmission of its intermediate results to the master CPU.

The new mechanism uses a model that includes the N and A factors (to estimate the amount of isosurfacing computation to be performed) and new factors c_1 and c_2. The new factors model transmission overhead. The cluster computer used in this work has a network in which transmission time is directly proportional to the size of the triangle strip payload to be transmitted. The triangle strip payload size is approximately a multiple of the isosurface extraction effort, so we model the transmission effort, T_i, for the i-th blocklet, as a function of its isosurface extraction work C_i: $T_i = \gamma C_i + c_2$, where c_2 is the overhead for building a message tunnel between the slave and master node. Our cluster computer also has two CPUs per node, so one slave node is on the same node as the master CPU. For that slave node, the transmission overhead, c_1, will be different than the overhead for the other nodes. Hence, for one slave node, the transmission effort, T_j, for the j-th blocklet is $T_j = \gamma C_j + c_1$.

In our model, we allow computation to be interleaved m times with transmission. The master CPU is assigned a small workload of size w. After the master computes the isosurface in the blocklets assigned in its workload, it becomes dedicated to receiving results from the slave CPUs. Thus, for a system with n total processors with the master denoted as processor 1, the computational workload $C_{i,j}$ for the i-th CPU in the j-th interleaving stage will be:

$$C_{1,1} = w;\ C_{1,j} = 0,\ \ j = \{2,3,...,m\}$$

$$C_{2,1} = w;$$

$$C_{i,1} = C_{i-1,1} + T_{i-1,1},\ \ i = \{3,4,...,n\};\text{ and}$$

$$C_{i,j} = \left(\sum_{k=1}^{n-i} T_{i+k,j-1}\right) + \sum_{k=2}^{i-1} T_{k,j},\ \ i = \{2,3,...,n\}, j > 1,$$

where

$$T_{1,j} = 0,\ \ j = \{1,2,...,m\};$$

$$T_{2,j} = \gamma C_{2,j} + c_1;\text{ and}$$

$$T_{i,j} = \gamma C_{i,j} + c_2,\ i > 2, j = \{1,2,...,m\}\ .$$

For example, on CPU 2 at the second interleaving stage,

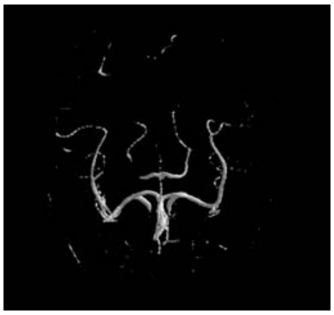


Figure 2: *Isosurface extracted using piggybacking scheme from MRA-Brain dataset at isovalue 100*

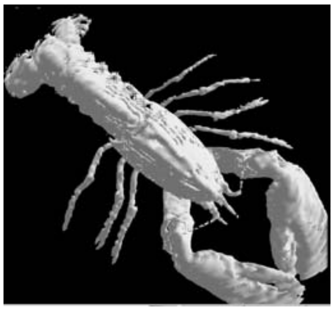


Figure 3: *Isosurface extracted using piggybacking scheme from Lobster dataset at isovalue 100*

$$C_{2,2} = (\gamma^3 + 3\gamma^2 + 2\gamma)w + (\gamma^2 + 2\gamma)c_1 + (\gamma + 2)c_2 \text{ and}$$
$$T_{2,2} = \gamma C_{2,2} + c_1.$$

For the cluster computer available to us, a battery of isosurface extraction benchmarking experiments was performed to collect isosurface times and counts of the active cells and total cells. The battery utilized a large number of isoqueries on several datasets. We used least-squares fitting on the test data to determine that $\alpha = 7E - 7$, $\beta = 8.5E - 8$ for the blocklet based isosurface extraction of [ZN03]. For the new mechanism, we similarly determined that $\alpha = 7E - 7$, $\beta = 1.42E - 7$, $c_1 = -2.8E - 8$, and $c_2 = 3.0E - 9$. We found it best to model the γ factor differently for the slave CPU that was in the same node as the master. We denote γ_1 as that CPU's γ and γ_2 as the γ value for the other CPUs. We also determined that $\gamma_1 = 1.7E - 7$ and $\gamma_2 = 2.4E - 7$, again using least-squares on the test data. The parameter values are machine-specific; application to other clusters will require fitting to data collected for the cluster.

4. Experimental Results and Analysis

Next, the out-of-core scheme's performance is considered. Performance testing was performed using the National Center for Supercomputing Applications (NCSA) linux cluster, which has over 1000 3.2 GHz Intel Xeon dual-processor nodes. The tests reported here use a maximum of 64 of the CPUs at a time. Each node has 3GB RAM and is interconnected to the other nodes via a Myrinet 2000 network.

All experiments reported here used the [ZN03] code as a baseline on which no changes were made save the addition of triangle-stripping, communication piggybacking, and IPC-cognizant work estimation coupled with IPC interleaving. That code is a quite suitable baseline, given the demonstration in [ZN03] and comparison tests based on it in [ZN04].

4.1. Extraction Performance

Two sample isosurface renderings for a 256×256×72 magnetic resonance dataset of the brain (called MRA-Brain) and for a 358×239×232 computerized tomography dataset of a lobster (called Lobster) using our piggybacking scheme are shown in Figures 2 and 3. These renderings are identical to the renderings produced by non-piggybacked isosurfacing since the triangle meshes are the same.

In Figure 4(b), the scaled-up speedup in the isosurface computation is shown for our load-balanced piggybacked scheme versus the load-balanced out-of-core isosurfacing without piggybacking. The figure shows the results for an extraction on the Lobster dataset for isovalue 62. In scaled-up speedup [Gus88], the amount of work increases exactly in step with the increase in the number of processors. Scaled-up speedup measures the degree of additional work that can be performed as computational resources increase. One disadvantage of conventional speedup is that as processor resources increase, the total amount of cache typically also increases, causing more and more of a problem of a given size to be able to be cached. Scaled-up speedup enables a speedup measurement that is less affected by increasing cacheability. To measure scaled-up speedup accurately, we generated larger versions of the MRA-Brain and Lobster datasets by repeatedly duplicating the layers of the dataset. For example, a single duplication results in a dataset that has exactly twice the isosurfacing computational burden as the original dataset. Here, the speedups of the piggybacked and non-piggybacked approaches are essentially identical and linear; the largest divergence is the 32-CPU case where the piggybacking has a speedup of 31.0 and the standard approach has a speedup of 31.3. The largest divergence in scaled-up speedups that we have observed is a 4% relative difference.

For comparison, the conventional speedup for a second dataset, Huge-Molecule, which is a dataset of a molecule, is shown in Figure 4(c). The dataset is of size 190×190×6049. Also for this dataset, the problem faced by out-of-core isosurfacing on cluster computers is demonstrated in Figure 4(a). The figure shows the times for the [ZN03] (i.e., non-piggybacked, non-interleaved) out-of-core isosurfacing's actual isosurfacing computations and the times to communicate the isosurface back to the master CPU on the

NCSA cluster for various numbers of CPUs. As the number of processors increases, processing quickly becomes communication-bound. We have rarely observed continued improvement in overall performance beyond 32 CPUs.

Thus, the impact of piggybacking on isosurface extraction's computational behavior is minimal. In particular, very high efficiencies in slave CPU utilization are maintained by piggybacked out-of-core isosurfacing.

4.2. IPC Reduction

We have performed several experiments to study the reduction in IPC resulting from use of the piggybacking. As suggested above, typically we observe approximately a 10% reduction in data transfer from the piggybacking. A higher degree of reduction in IPC could be achieved if larger blocklet sizes were used. This occurence is because our scheme cannot form triangle strips across blocklet boundaries if the adjacent blocklets are assigned to CPUs on different nodes. For future work, we hope to more fully explore the interaction between blocklet size and IPC reduction.

Choice of the blocklet size involves optimizing a tradeoff between reduction of IPC and reduction of unnecessary computation. (As blocklet size grows, the number of inactive cells that need to be processed also grows since each active blocklet can have inactive cells.) In the case most optimal for IPC reduction, a blocklet the size of the volume can be used. Of course, such a usage produces terrible computational speedup and high overall isosurfacing time and thus is not practical. Such a usage does provide an upper bound on the IPC reduction from our piggybacking, though.

Two experiments in which blocklet size was equal to volume size were performed to test IPC reduction. The experiments consisted of two isoqueries on the MRA-Brain dataset. In the first isoquery, 0.2% of the cells were active. In the second isoquery, 5.2% of the cells were active. Table 1 summarizes the amount of transferred facet information to the master for cases that use and that don't use our piggybacking approach. In each case, the piggybacking reduced IPC by about 15%. This reduction is significant considering that Isoquery 2 completes in a total time of 0.10 seconds on 1 CPU—to achieve an overall speedup of 16 on 16 CPUs would require the isosurfacing to be completed in 0.00625 seconds. Yet, the network bandwidth would have to be greater than 6.5 Gb/sec to support such an achievement.

4.3. Total Performance

In Table 2, the total execution times, including isosurface extraction on the slave CPUs, IPC communication, and rendering from the master CPU, are shown for an isosurface extraction at isovalue 62 for three datasets using 16 CPUs of the NCSA cluster. One dataset is Huge-Molecule, which was described earlier. The other two datasets are scaled-up versions of the brain and lobster datasets described earlier—the MRA dataset of the brain here (Huge-Brain) has 2273 slices and the dataset of the lobster (Huge-Lobster) has 7393 slices. The Huge-Lobster dataset is quite large, encompassing over 600 million data points. The isosurface extracted from Huge-Lobster has over 130 million facets. The parameter values described above were used for workload scheduling. Thus, the non-interleaved run results used the parameters for the model in [ZN03] and the interleaved run results used the parameters for the new mechanism reported here. The piggybacking and interleaving components of our scheme yield approximately equal impacts on total execution time. Overall, approximately 13% improvement in total execution time was observed from use of the scheme.

Approach	Isoquery 1	Isoquery 2
No Piggyback	10.85 (220 KB)	259.8 (5085 KB)
Piggybacked	6.7 (187 KB)	163.6 (4334 KB)

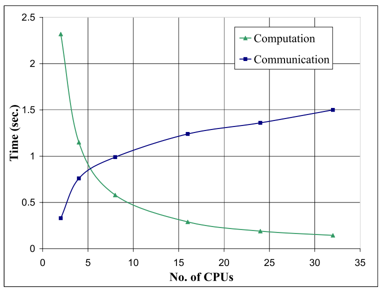
Table 1: *IPC transfers in terms of thousands of facets for piggybacked versus non-piggybacked isosurfacing using maximal blocklet size for two isoqueries on a small MRA dataset. Size in bytes of transfers shown in parens.*

		No Piggyback	Piggybacked
£	No Interleaving	8.05	7.80
£	Interleaving	7.88	7.61
⋆	No Interleaving	1.13	1.07
⋆	Interleaving	1.03	0.96
†	No Interleaving	1.53	1.44
†	Interleaving	1.39	1.32

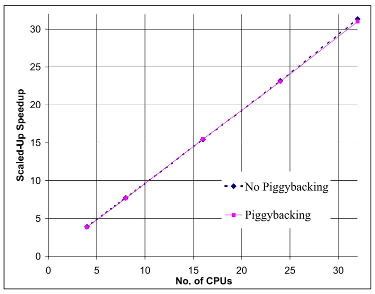
Table 2: *Total execution time (sec.) with and without communication improvements, 16 CPUs, for Huge-Lobster (denoted by £), Huge-Brain (denoted by ⋆), and Huge-Molecule (denoted by †) datasets*

5. Conclusion

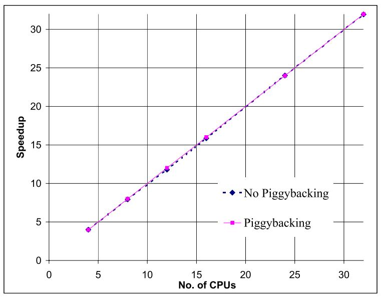
In this paper, the use of communication piggybacking and interleaving has been coupled with an accurate and efficient load-balancing mechanism to allow for fast isosurfacing of large datasets. This coupling allows total system capability to be well-exercised, thus offering improvement over current-generation methods that focus primarily on computational aspects of the system. The scheme can allow about 13% faster isosurfacing than the current fastest approach for large datasets. Piggybacking and interleaving were found to be approximately equal contributors to the observed performance improvement. The approach is especially well-suited to cluster computers, including clusters constructed using off-the-shelf network equipment. In the future, we hope to explore means to further reduce the IPC requirements in parallel out-of-core isosurfacing.



(a) Comp. vs. Comm., Huge-Molecule



(b) Scaled-up Speedup, Lobster



(c) Speedup, Huge-Molecule

Figure 4: *Isosurfacing statistics: (a) computation versus result communication times for Huge-Molecule Dataset, isovalue 62, (b) Speedup (scaled-up) for piggybacked and non-piggybacked isosurfacing computations for Lobster dataset at isovalue 62, (c) Speedup for piggybacked and non-piggybacked isosurfacing computations for Huge-Molecule dataset at isovalue 62*

Acknowledgements

This work was supported by National Science Foundation grant ACI-0222819 and by a computer time allocation from the National Center for Supercomputing Applications. MRA data was provided by Cleveland Clinic Foundation. This paper was strengthened by the reviewers' recommendations and the assistance of H. Zhang and J. Lee.

References

[Ano05] ANON.: The 2004 visualization career award. In *Proc., Visualization '05* (2005), p. xxii.

[BPTZ99] BAJAJ C., 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.

[CFSW01] CHIANG Y.-J., FARIAS R., SILVA C., WEI B.: A unified infrastructure for parallel out-of-core isosurface extraction and volume rendering of unstructured grids. In *Proc., IEEE Symp. on Parallel and Large-data Visualization and Graphics (PVG'01)* (2001), pp. 59–66.

[GS01] GAO J., SHEN H.-W.: Parallel view-dependent isosurface extraction using multi-pass occlusion culling. In *Proc., IEEE Symp. on Parallel and Large-data Visualization and Graphics (PVG'01)* (2001), pp. 67–74.

[Gus88] GUSTAFSON J. L.: Reevaluating amdahl's law. *CACM 31*, 5 (1988), 532–533.

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

[LNW04] LAKSHMIPATHY J., NOWINSKI W., WERNERT E.: A novel approach to extract triangle strips for iso-surfaces in volumes. In *Proc., IEEE Int'l Conf. on VR Continuum and its Apps. in Ind. '04* (2004), pp. 239–245.

[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.

[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.

[MSS94] MONTANI C., SCATENI R., SCOPIGNO R.: Discretized marching cubes. In *Proc., Visualization '94* (1994), pp. 281–287.

[NH91] NIELSON G. M., HAMANN B.: The asymptotic decider: Resolving the ambiguity in marching cubes. In *Proc., Visualization '91* (1991), pp. 83–91.

[Nie03] NIELSON G. M.: On marching cubes. *IEEE Trans. on Visualization and Computer Graphics 9*, 3 (2003), 283–297.

[NSG04] NEEMAN A., SULATYCKE P., GHOSE K.: Fast remote isosurface visualization with chessboarding. In *Proc., EG Symp. on Parallel Graphics and Visualization '04* (2004), pp. 75–82.

[YSF05] YANG L., SCHOPF J., FOSTER I.: Improving parallel data transfer times using predicted variances in shared networks. In *Proc., CCGrid '05* (2005), pp. 734–742.

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

[ZN04] ZHANG H., NEWMAN T.: Case study of multithreaded in-core isosurface extraction algorithms. In *Proc., EG Symp. on Parallel Graphics and Visualization '04* (2004), pp. 83–92.