Hypergraph-Partitioning-Based Remapping Models for Image-Space-Parallel Direct Volume Rendering of Unstructured Grids

Berkant Barla Cambazoglu and Cevdet Aykanat, *Member*, *IEEE*

Abstract—In this work, image-space-parallel direct volume rendering (DVR) of unstructured grids is investigated for distributed-memory architectures. A hypergraph-partitioning-based model is proposed for the adaptive screen partitioning problem in this context. The proposed model aims to balance the rendering loads of processors while trying to minimize the amount of data replication. In the parallel DVR framework we adopted, each data primitive is statically owned by its home processor, which is responsible from replicating its primitives on other processors. Two appropriate remapping models are proposed by enhancing the above model for use within this framework. These two remapping models aim to minimize the total volume of communication in data replication while balancing the rendering loads of processors. Based on the proposed models, a parallel DVR algorithm is developed. The experiments conducted on a PC cluster show that the proposed remapping models achieve better speedup values compared to the remapping models previously suggested for image-space-parallel DVR.

Index Terms—Direct volume rendering, unstructured grids, ray casting, image space parallelization, hypergraph partitioning, screen partitioning, remapping.

1 Introduction

1.1 Direct Volume Rendering

Direct volume rendering (DVR) is a popular volume visualization technique [17], employed in exploration and analysis of 3D data grids used by scientific simulations. DVR applications are rather important in that they foster research studies by letting scientists have better visual understandings of the problems under investigation. In the last decade, DVR research has been accelerated due to the ever-growing size and use of numeric simulations and the need for fast and high-quality rendering. Today, DVR finds application in a wide range of research fields that require interpretation of large volumetric data.

In many scientific simulations, data values are located at the vertices (data points) of a 3D grid that represents a physical phenomena. The connectivity between vertices shapes volumetric primitives (cells) of the grid and forms a volumetric data set to be visualized. Unstructured data sets, which are mainly used in disciplines such as fluid dynamics, shock physics, and thermodynamics, are a special type of grid-based volumetric data sets. The data points in unstructured grids are irregularly distributed. The lack of implicit adjacency information between cells, the high amount of cell size variation, and the large size of the data sets make rendering these grids a challenging problem.

The aim of DVR is to map a set of scalar or vectorial values (e.g., pressure, temperature, and velocity) defined throughout a 3D data grid to some color values, which form a 2D image on the screen. Unlike surface-based rendering techniques, no intermediate representations are generated for the data. Instead, the volume is treated as a whole, and the color is formed by a series of sampling and composition operations performed within the volume. In general, the image is generated by iterating over the object space (data space) or image space (screen space) primitives. Object space (OS) methods [6], [18] visit volumetric data primitives and compute their color contributions on the screen. Image space (IS) methods [26], [29] visit screen pixels and assign a color value to each pixel by compositing the samples taken along the rays fired from the pixels into the volume.

In this work, a slightly modified version of Koyamada's IS DVR algorithm [26] is used as the underlying sequential DVR algorithm. In this algorithm, projected areas of all front-facing external faces of grid cells (in our case, tetrahedral cells) are scan converted to find the pixels covered on the screen. From each such pixel, a ray is shot into the volume, and a ray segment is generated between a front-facing external face and a back-facing external face (Fig. 1). Ray segments are traversed using the adjacency information between cells. While traversing a ray segment, intersection tests are performed between the ray segment and cell faces to find the points where the ray segment leaves the cells. The exit points found are used as the entry points for the following cells. After the entry and exit points of a cell are computed, some sampling points are determined along the ray segment within the cell. The number and location of the sampling points depend on the sampling technique used. In midpoint sampling, which is frequently used for unstructured grids, a single sampling point, located in the middle of the entry and exit points, is used.

At each sampling point, new sampling values are computed by interpolating the data values at the data points

• *The authors are with the Computer Engineering Department, Bilkent University, 06800, Ankara, Turkey.*
E-mail: {berkant, aykanat}@cs.bilkent.edu.tr.

Manuscript received 28 Feb. 2005; revised 9 Jan. 2006; accepted 24 Jan. 2006; published online 28 Nov. 2006.
Recommended for acceptance by G. Karypis.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDS-0193-0205.

1045-9219/07/$20.00 © 2007 IEEE Published by the IEEE Computer Society

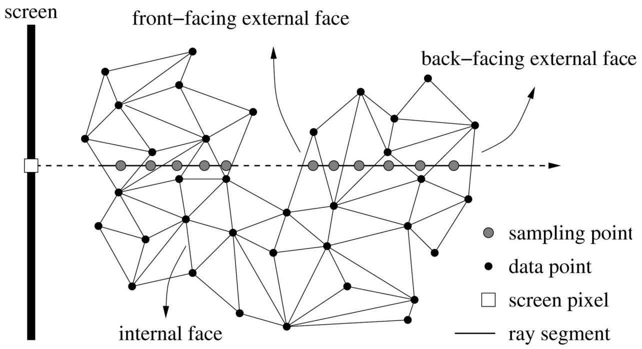


Fig. 1. Ray-casting-based DVR of unstructured grids with midpoint sampling.

of the cell which contains the sampling point. The sampling values are passed from transfer functions, and corresponding color and opacity values are calculated. These values are composited in visibility order, and the color generated for the ray segment is put in the respective pixel's buffer. Due to the concavity of the volume, there may be more than one ray segment generated for the same pixel and, hence, more than one color value may be stored in the same pixel buffer. After all ray segments are traversed, the colors in pixel buffers are composited in visibility order, and the final colors on the screen are generated. Since the rays shot from nearby pixels of the screen are likely to pass through the same cells, IS coherency is utilized by this algorithm. Since the adjacency information between cells is used, OS coherency is also utilized.

1.2 Parallel DVR

Due to the excessive amount of sampling and composition operations, DVR algorithms suffer from a considerable speed limitation. Moreover, memory needs of recent data sets are beyond the capacities of today's conventional computers. These render sequential DVR algorithms inadequate for practical use. In the literature, parallel DVR algorithms exist for shared-memory [47], [48], distributed-memory [4], [27], [31], [32], [38], and distributed-shared-memory [12], [13], [19], [21] architectures. Our work considers distributed-memory parallel DVR, in which OS or IS parallelization approaches can be followed.

OS-parallel methods [4], [31], [32] partition the data into subvolumes and assign them to processors. Each processor locally renders its subvolume and produces a full-screen but partial image. IS-parallel methods [27], [38] partition the screen into subscreens and assign the subscreens to processors. Each processor locally renders its subscreen and produces a small but complete portion of the final image. Both OS and IS parallelizations require a communication step in which IS primitives (pixels) or OS primitives (cells) are transferred between processors, respectively. In OS parallelization, communication is performed after the local rendering to merge the partial images into a final image. In IS parallelization, communication is performed before the local rendering to replicate some OS primitives so that each processor has all OS primitives that it needs in rendering its subscreen. In this respect, OS and IS parallelizations can be, respectively, classified as sort-last and sort-first by the taxonomy of [33]. This work focuses on IS-parallel DVR.

In IS-parallel DVR, visualization parameters (such as view point and viewing direction) determine the computational structure in rendering since they affect both the rendering load distribution on the screen and the interaction between OS and IS primitives. In successively visualizing a data set with different visualization parameters, existing screen partitions turn into poor partitions that cause a rendering load imbalance among processors. Hence, pixel-to-processor mapping is important for balancing rendering loads of processors and minimizing the communication overhead during data replication.

Three approaches can be followed in pixel-to-processor mapping: static, dynamic, and adaptive. In the static approach, nearby pixels are scattered among processors with the assumption that adjacent pixels have similar rendering loads. The advantage of this scheme is simplicity. However, since IS coherency is disturbed, it causes high amounts of data replication. In the dynamic approach, pixels are remapped to processors on a demand-driven basis. This approach solves the load balancing problem in a natural way, but it suffers from disturbing IS coherency since nearby pixels may be processed by different processors. Moreover, each pixel assignment incurs communication on distributed-memory architectures. The adaptive approach, also adopted in this work, rebalances the rendering load explicitly by repartitioning the screen at the beginning of each visualization instance (i.e., a rendering cycle, which generates an image frame) in a series of visualizations on the data. In this approach, current visualization parameters are utilized to maintain the load balance, and nearby pixels are mapped to the same processors to preserve IS coherency.

1.3 Previous Work on IS Parallelization

Challinger [12], [13] presented IS parallelizations of a hybrid DVR algorithm [14]. In [12], scanlines on the screen were assigned to processors using the static and dynamic approaches in two different algorithms. In [13], pixel blocks were considered as atomic tasks for dynamic assignment. Wilhelms et al. [47] presented IS parallelization of a hierarchical DVR algorithm for multiple grids. A survey on parallel DVR can be found in [49].

In the literature, several IS-parallel polygon rendering works exist [30], [39], [40], [41]. Samanta et al. [39] developed an IS-parallel rendering system for a multiprojector display wall. For dynamic load balancing, they developed three screen partitioning algorithms. In [40], they developed a hybrid polygon rendering algorithm on a PC cluster. In [41], they investigated a replication strategy for this algorithm. Lin et al. [30] followed the adaptive approach in their polygon rendering algorithm using a binary tree for screen partitioning.

In DVR, the adaptive approach was investigated in two different works [27], [38]. Palmer and Taylor [38] presented adaptive IS parallelization of a ray-casting-based DVR algorithm. Kutluca et al. [27] presented and discussed 12 screen partitioning algorithms for adaptive IS-parallel DVR of unstructured grids. All of those algorithms are common in that they try to rebalance the rendering load but have no explicit attempt on minimizing the data replication overhead. This work aims to fill this gap in the literature.

1.4 Proposed Work

In this work, we propose a novel model, which formulates the adaptive screen partitioning problem as a hypergraph partitioning problem. In this model, the interaction between OS and IS primitives is represented as a hypergraph. By partitioning this interaction hypergraph into equally weighted parts, the proposed model partitions the screen into subscreens that have similar rendering loads. Also, by minimizing the cost of the partition, the model aims to minimize the total amount of data replication in the parallel system. In this model, minimizing the total replication amount also corresponds to minimizing the upper bound on the total volume of communication during the data replication.

In the parallel DVR framework we adopted, OS primitives are statically owned by their home processors, responsible for sending them to the processors where they are needed and, hence, must be temporarily replicated. As another contribution, the above model is enhanced, and two remapping models are proposed to accurately formulate the communication requirement in this framework: two-phase and one-phase remapping models.

The two-phase model aims to find a screen partition and a pixel-to-processor remapping that minimize the total volume of communication and balance the rendering load distribution. Partitioning and mapping form the two consecutive phases of our two-phase model, in which a screen partition is obtained by partitioning the interaction hypergraph and then subscreens are mapped to processors by the maximum weight matching algorithm for weighted bipartite graphs [15]. The one-phase model directly obtains a remapping by partitioning the remapping hypergraph, which is formed by augmenting the interaction hypergraph. This model tries to balance the sum of the local rendering and communication volume loads of processors while minimizing the total communication volume.

Based on the proposed models and Koyamada's sequential DVR algorithm [26], an adaptive IS-parallel DVR algorithm is developed. Experiments were conducted using well-known data sets, and the performance was tested on a 32-node PC cluster. Comparisons with jagged partitioning, which was found by [27] to be the best screen partitioning algorithm in minimizing data replication, show that the proposed models achieve better speedups by incurring less communication volume and obtaining better load balance.

The rest of the paper is organized as follows: Section 2 discusses the issues in adaptive IS parallelization and the preprocessing techniques we developed. Section 3 presents the proposed models in detail. Section 4 describes our parallel DVR algorithm. Section 5 presents experimental results, which validate the work. Section 6 concludes the paper.

2 Adaptive IS Parallelization Issues and Proposed Solutions

2.1 Screen Partitioning

In the adaptive screen partitioning approach, to be able to partition the screen in a balanced manner, the rendering load distribution on the pixels must be calculated in a view-dependent preprocessing step at the beginning of each visualization instance. The rendering load of a pixel may be assumed to be equal to the number of samples that will be taken along the ray fired from the pixel into the volume. In unstructured tetrahedral grids, with midpoint sampling, this is equal to the number of front-facing faces intersected by the ray and, hence, the screen workload can be calculated as follows: First, the sampling load of each pixel is set to zero. Then, all cells are traversed. The pixels under the projected area of each front-facing face of a cell are found by scan conversion, and sampling loads of those pixels are increased by one. Consequently, after all of the projected areas of front-facing faces are scan converted, rendering loads of all screen pixels are estimated.

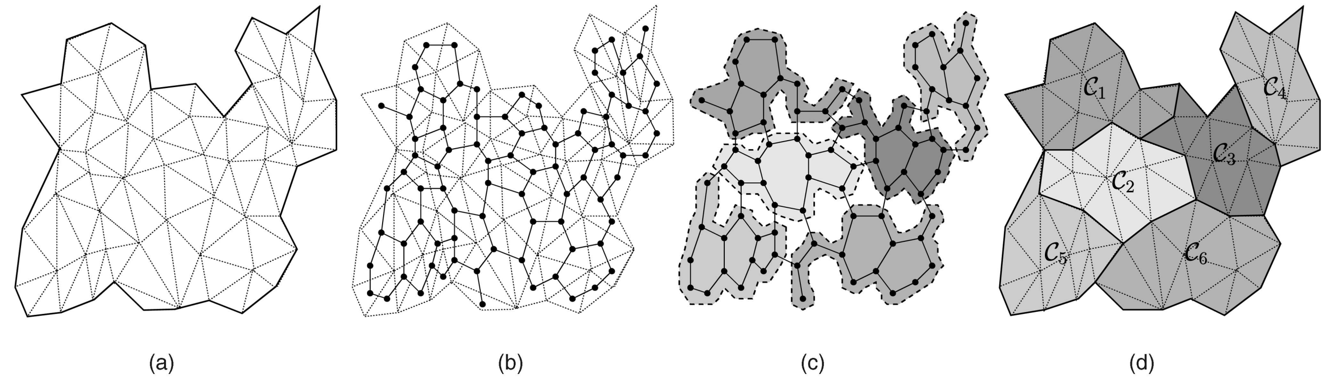
After the screen workload is computed, the screen is partitioned into subscreens such that the estimated rendering loads of subscreens are similar. The number of subscreens is chosen equal to the number of processors so that each processor is assigned the task of rendering one of the subscreens. In the literature, several screen partitioning techniques exist. Quad trees, recursive bisection, and jagged partitioning are among such techniques [27], [34]. In these techniques, the subscreens are always isothetic rectangles. This restriction decreases the flexibility in partitioning and prevents getting further performance.

In this work, for implementation efficiency in screen partitioning, an $M \times M$ coarse mesh, which forms M^2 square pixel blocks, is imposed on the screen. An individual pixel block constitutes an atomic rendering task, assigned to a single processor. The set of pixel blocks assigned to a processor forms a subscreen for that processor. In the extreme case of using a too-fine mesh, a single pixel corresponds to a single pixel block. This allows the partitioning algorithm to have the highest flexibility in determining subscreen boundaries. However, the increasing preprocessing overhead makes this approach practically infeasible. On the contrary, the use of a too-coarse mesh may restrict the solution space of the partitioning algorithm and prevent having a satisfactory load balance. A better approach is to trade off between the preprocessing overhead and the size of the solution space by varying M according to the current parallelization and visualization parameters.

2.2 Cell Clustering

Scan converting all front-facing faces for calculation of the screen workload is a costly operation. To reduce the scan conversion cost, in a view-independent preprocessing phase, we apply a top-down, graph-partitioning-based clustering on the data. The motivation behind this clustering is grouping close tetrahedral cells to form cell clusters with small surface areas so that the total surface area to be scan converted during the workload calculations is smaller. The clustering is independent of visualization parameters and is performed just once at the very beginning of the series of visualization instances. Hence, the preprocessing overhead introduced is almost negligible. The proposed parallel DVR algorithm and models work on cell clusters throughout the succeeding view-dependent preprocessing and data replication phases instead of working on individual cells.

In our graph-partitioning-based clustering approach, cells correspond to tasks to be partitioned, and cell clusters



(a) (b) (c) (d)

Fig. 2. (a) A tetrahedral data set. (b) The graph representation of the data set. (c) A partition obtained by 6-way graph partitioning. (d) The resulting set of six cell clusters.

correspond to parts to be formed. In the clustering graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, each vertex in $\mathcal{V}$ represents a tetrahedral cell. An edge in $\mathcal{E}$ exists between two vertices if and only if a face is shared by the cells corresponding to those vertices. Vertices and edges are associated with weights. As the weight of each vertex, a unit cost of 1 is assigned. The area of a face shared between two neighbor cells is assigned as the weight of the respective edge connecting the vertices corresponding to those two cells.

C-way partitioning [24] of the clustering graph $\mathcal{G}$ creates a mutually disjoint and exhaustive set $\{\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_C\}$ of C nonempty cell clusters. In partitioning, since part weights are balanced, clusters contain almost equal number of cells and, hence, their communication costs will be similar in data replication. Minimizing the weighted edge cut corresponds to minimizing the total surface area of cell clusters. This clustering scheme, illustrated in Fig. 2, aims to minimize both the interaction between adjacent cell clusters and the average-case interaction between cell clusters and the screen. This means smaller projected areas for cell clusters and, hence, less scan conversion cost in workload calculations.

In this approach, the total number C of generated clusters must be chosen carefully. In one extreme, C can be chosen to be equal to the number of processors. In such a case, the solution space of the partitioning algorithm is severely restricted. On the other extreme, each cluster can be made up of a single tetrahedral cell, in which case we face with an extremely high preprocessing overhead. In this work, C is chosen empirically.

During the view-dependent screen workload calculations at the beginning of each visualization instance, the rendering load of a cell cluster $\mathcal{C}$ is estimated as the sum of the projected areas of all front-facing faces ($\mathcal{F}_\mathcal{C}$) in the cell cluster and is calculated as

$$CCload(\mathcal{C}) = \sum_{f \in \mathcal{F}_\mathcal{C}} a_f, \tag{1}$$

where the projected area of a face f is $a_f = |x_1(y_2 - y_3) + x_2(y_3 - y_1) + x_3(y_1 - y_2)|$. Here, x_i and y_j are the coordinates (in normalized projection coordinate system) of vertices of face f. To determine the pixel blocks whose sampling loads are affected, each cell cluster's projected area is computed by scan converting projected areas of front-facing faces on the surface of the cell cluster. To calculate the screen workload, the estimated rendering load of each cell cluster (1) is distributed evenly among the pixel blocks that are overlapped by the projected area of the cell cluster. In this approach, since each pixel block affected by the cell cluster is assigned equal rendering load, estimation errors are introduced. Also, since cell clusters are replicated as a whole, communication volume slightly increases. However, cell clustering brings the benefit of reduced preprocessing cost during the screen workload calculations. Furthermore, in the implementation, it simplifies housekeeping, decreases the number of iterations in some loops, and simplifies some data structures.

2.3 Remapping and Data Replication

As visualization parameters change, the rendering load distribution on the screen and, hence, on processors change. In adaptive IS-parallel DVR, the screen is repartitioned at the beginning of each visualization instance, and pixels are remapped to processors for load rebalancing. Since OS primitives need to be shared among processors, they must be replicated via communication between processors. For an efficient parallelization, novel remapping models are needed. These models should rebalance the load distribution in the parallel system while minimizing the communication overhead due to data replication.

In the literature, several graph-partitioning-based remapping models exist for the problems in other contexts. These models may be classified as scratch-remap [36], [43] or diffusion-based [37], [42], [43], [46]. Scratch-remap models work in two phases. In the first phase, tasks are partitioned into parts, which have similar computational loads. In the second phase, parts are mapped to processors such that the data migration overhead is as low as possible. Diffusion-based models move tasks from heavily loaded to lightly loaded processors and interleave minimization of the migration overhead with load balancing.

3 Screen Partitioning and Remapping Models

3.1 Hypergraph Partitioning Problem

A hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{N})$ consists of a set of vertices $\mathcal{V}$ and a set of nets $\mathcal{N}$ [5]. Each net n_j in $\mathcal{N}$ connects a subset of vertices in $\mathcal{V}$, which are said to be the pins of n_j. Each vertex v_i has a

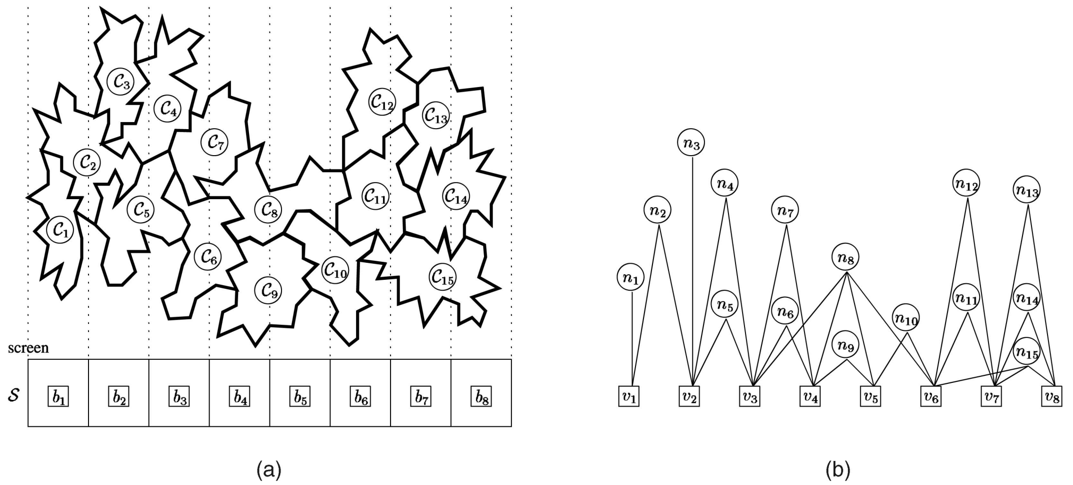


Fig. 3. (a) A sample visualization instance with 15 cell clusters and eight pixel blocks. (b) The interaction hypergraph $\mathcal{H}_\mathrm{I}$ representing the interaction between the cell clusters and pixel blocks in (a).

weight w_i, and each net n_j has a cost c_j. $\Pi = \{\mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_K\}$ is a K-way vertex partition if each part $\mathcal{V}_k$ is nonempty, parts are pairwise disjoint, and the union of parts gives $\mathcal{V}$. In Π, a net is said to connect a part if it has at least one pin in that part. The connectivity set Λ_j of a net n_j is the set of parts connected by n_j. The connectivity $\lambda_j = |\Lambda_j|$ of a net n_j is equal to the number of parts connected by n_j. If $\lambda_j = 1$, then n_j is an internal net. If $\lambda_j > 1$, then n_j is an external net and is said to be cut. In Π, the weight W_k of a part $\mathcal{V}_k$ is equal to the sum of the weights of vertices in $\mathcal{V}_k$, i.e.,

$$W_k = \sum_{v_i \in \mathcal{V}_k} w_i. \tag{2}$$

The K-way hypergraph partitioning problem [1] is defined as finding a vertex partition Π for a given hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{N})$ such that part weights are balanced while a cost defined on nets is optimized. In this work, the connectivity-1 metric

$$\chi(\Pi) = \sum_{n_j \in \mathcal{N}} c_j(\lambda_j - 1) \tag{3}$$

is used as the cost to be minimized. In this metric, which is frequently used in VLSI [16], [28] and recently used in scientific computing [3], [11], [45] communities, each net n_j contributes $c_j(\lambda_j - 1)$ to the cost $\chi(\Pi)$ of a partition Π.

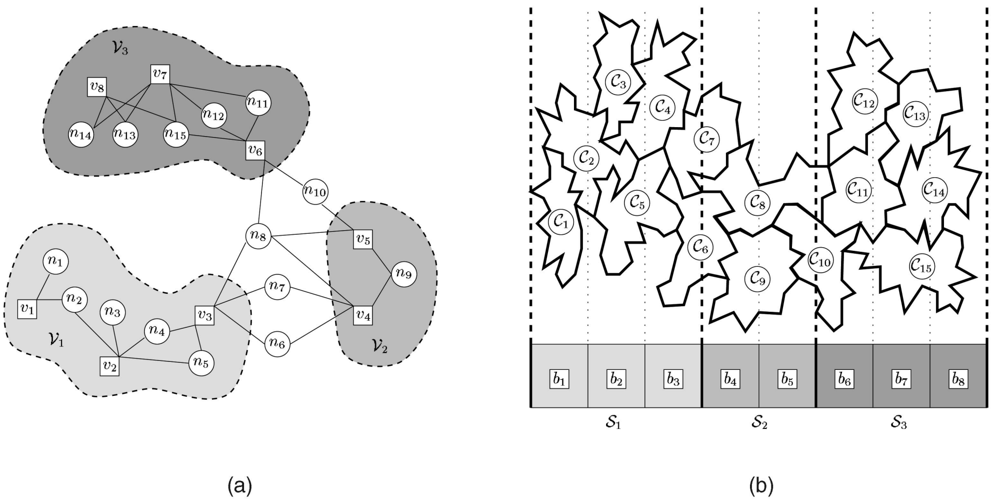
3.2 Adaptive Screen Partitioning Model

We model the computational structure of a visualization instance as a hypergraph and formulate the screen partitioning problem in adaptive IS-parallel DVR as a hypergraph partitioning problem. In the proposed model, an interaction hypergraph $\mathcal{H}_\mathrm{I} = (\mathcal{V}, \mathcal{N})$ represents the interaction between OS primitives (cell clusters) and IS primitives (pixel blocks). In $\mathcal{H}_\mathrm{I}$, a vertex v_i in vertex set $\mathcal{V}$ represents a pixel block b_i in set $\mathcal{S}$ of pixel blocks. As the weight w_i of a vertex v_i, the rendering load $PBload(b_i)$, estimated during the screen workload calculations for the corresponding pixel block b_i, is assigned. A net n_j in net set $\mathcal{N}$ represents a cell cluster $\mathcal{C}_j$. Vertex v_i is a pin of a net n_j if the projected area of cell cluster $\mathcal{C}_j$ overlaps pixel block b_i. As the cost c_j of a net n_j, the storage cost $Cost(\mathcal{C}_j)$ of the corresponding cell cluster $\mathcal{C}_j$ is assigned. Here, $Cost(\mathcal{C}_j)$ is the number of bytes needed to store (or send) the C_j cluster's data.

Fig. 3a illustrates a sample visualization instance. To simplify the drawing and ease understanding, 3D cell clusters are illustrated as 2D regions. Similarly, the 2D screen is replaced with a single row of pixel blocks. The dotted vertical lines show the view volume boundaries of pixel blocks, assuming parallel projection. Throughout the examples, unit rendering loads and storage costs are assumed for pixel blocks and cell clusters, respectively. Fig. 3b shows the interaction hypergraph $\mathcal{H}_\mathrm{I}$ constructed to represent the sample interaction of Fig. 3a. In $\mathcal{H}_\mathrm{I}$, nets and vertices are represented by circles and squares, respectively. In Fig. 3b, for example, vertices v_1 and v_2 are the pins of net n_2 since the projected area of cell cluster $\mathcal{C}_2$ overlaps both pixel blocks b_1 and b_2.

After constructing $\mathcal{H}_\mathrm{I}$, the screen partitioning problem reduces to the hypergraph partitioning problem of finding a vertex partition $\Pi = \{\mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_K\}$, where each part $\mathcal{V}_k$ corresponds to a subscreen $\mathcal{S}_k$ to be rendered by a single processor. In the proposed model, a vertex partition Π is obtained by applying K-way hypergraph partitioning on $\mathcal{H}_\mathrm{I}$. As a result, since the weights of the parts in Π are balanced, the screen is partitioned into K subscreens $\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_K$, which have similar rendering loads. Hence, after the subscreens are assigned to processors, each processor performs almost the same amount of rendering.

In a partition Π, if a net n_j has a pin on a part $\mathcal{V}_k$ (i.e., $\mathcal{V}_k \in \Lambda_j$), then cell cluster $\mathcal{C}_j$ is needed in rendering at least one pixel block in subscreen $\mathcal{S}_k$ and, hence, must be replicated on the processor responsible from $\mathcal{S}_k$. Each cell cluster $\mathcal{C}_j$ is replicated on λ_j different processors, incurring $\lambda_j c_j$ bytes of replication in the parallel system. Hence, the total connectivity cost $\chi'(\Pi) = \sum_{n_j \in \mathcal{N}} c_j \lambda_j$ exactly corresponds to the total amount of cell cluster replication. By minimizing $\chi'(\Pi)$, the proposed model correctly minimizes this amount. Due to (4), there is a constant factor CF between the total connectivity cost $\chi'(\Pi)$ and the conventional connectivity-1 cost $\chi(\Pi)$ of a partition Π, i.e.,



(a)

(b)

Fig. 4. (a) A 3-way vertex partition Π of $\mathcal{H}_\mathrm{I}$ in Fig. 3b. (b) The corresponding screen partition.

$$\chi'(\Pi) = \sum_{n_j \in \mathcal{N}} c_j \lambda_j = \sum_{n_j \in \mathcal{N}} c_j(\lambda_j - 1) + \sum_{n_j \in \mathcal{N}} c_j = \chi(\Pi) + CF. \tag{4}$$

Therefore, minimizing $\chi(\Pi)$ (3) during the partitioning also minimizes $\chi'(\Pi)$, enabling the use of existing hypergraph partitioning tools [10], [25] without any modification.

Depending on the parallel DVR framework employed, some cell clusters may already have a copy on one or more processors in the parallel system. If a cell cluster $\mathcal{C}_j$ is already replicated on a processor where it is needed, then no communication is necessary for transferring $\mathcal{C}_j$ to that processor. Hence, the total replication amount $\chi'(\Pi)$ forms an upper bound on the total volume of communication, whose worst case occurs when no cell clusters have a copy in any of the processors where they must be replicated. As a result, minimizing the total connectivity cost $\chi'(\Pi)$ also corresponds to minimizing the upper bound on the total volume of communication. In the case that cell clusters are not stored within the parallel system, but retrieved from a central data server outside the parallel system, the model exactly minimizes the total volume of communication.

Fig. 4a shows a 3-way vertex partition Π found for a 3-processor system by applying hypergraph partitioning on $\mathcal{H}_\mathrm{I}$ of Fig. 3b. In Π, cut net n_8 has all three vertex parts in its connectivity set $\Lambda_8 = \{\mathcal{V}_1, \mathcal{V}_2, \mathcal{V}_3\}$. This means that cell cluster $\mathcal{C}_8$ is needed in rendering all three subscreens and, hence, it must be replicated on all processors. Similarly, $\lambda_6 = \lambda_7 = \lambda_{10} = 2$ for cut nets n_6, n_7, and n_{10} and, hence, cell clusters $\mathcal{C}_6$, $\mathcal{C}_7$, and $\mathcal{C}_{10}$ are each replicated on two processors. All of the remaining 11 nets are internal and, hence, replicated on a single processor. Therefore, the total replication amount is equal to $\chi'(\Pi) = 1 \times 3 + 3 \times 2 + 11 \times 1 = 20$. Fig. 4b illustrates subscreens $\mathcal{S}_1$, $\mathcal{S}_2$, and $\mathcal{S}_3$, formed according to vertex partition Π.

3.3 Remapping of Pixel Blocks

After the screen is partitioned and subscreens are found, a one-to-one subscreen-to-processor mapping $\mathcal{M}_\mathcal{S}$ must be created in order to assign each subscreen $\mathcal{S}_\ell$ to a processor $P_k = \mathcal{M}_\mathcal{S}(\mathcal{S}_\ell)$. This process remaps all pixel blocks in a subscreen to a processor for rendering. The many-to-one remapping $\mathcal{M}_b$ indicates the assignment of a pixel block b_i to a processor $P_k = \mathcal{M}_b(b_i)$. A subscreen-to-processor mapping $\mathcal{M}_\mathcal{S}$ can be created arbitrarily (e.g., $P_k = \mathcal{M}_\mathcal{S}(\mathcal{S}_k)$). Using $\mathcal{M}_\mathcal{S}$, $\mathcal{M}_b$ can be obtained as

$$P_k = \mathcal{M}_b(b_i) \Leftrightarrow b_i \in \mathcal{S}_\ell \wedge P_k = \mathcal{M}_\mathcal{S}(\mathcal{S}_\ell). \tag{5}$$

A vertex partition Π and a mapping $\mathcal{M}_\mathcal{S}$ together induce a replication pattern $\mathcal{R}_\mathcal{C}$ for cell clusters. A cell cluster $\mathcal{C}_j$ is replicated on a set $\mathcal{R}_\mathcal{C}(\mathcal{C}_j)$ of processors as

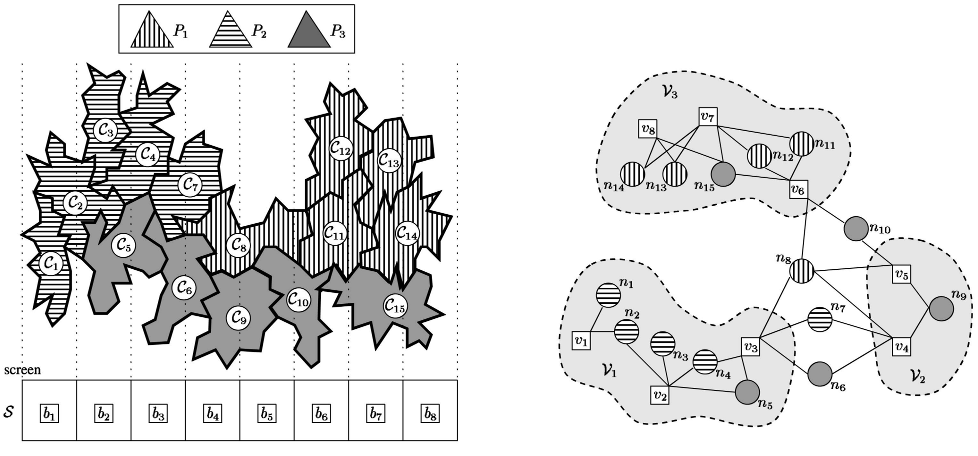
$$\mathcal{R}_\mathcal{C}(\mathcal{C}_j) = \{P_k : \exists \mathcal{V}_\ell, \mathcal{V}_\ell \in \Lambda_j \wedge P_k = \mathcal{M}_\mathcal{S}(\mathcal{S}_\ell)\}. \tag{6}$$

In our parallel DVR framework, each processor statically keeps a subset of cell clusters throughout the visualization. That is, at the beginning of a visualization instance, a single copy of each cell cluster $\mathcal{C}_j$ is available only on its home processor $P_k = Home(\mathcal{C}_j)$. Home processors are responsible from temporarily replicating their cell clusters on the processors that need them. In the following sections, we propose two remapping models that aim to minimize the total volume of communication within this framework.

3.3.1 *Two-Phase Remapping Model*

The two-phase model [8] has two consecutive phases. The first phase produces K subscreens, using partition Π found by K-way partitioning of $\mathcal{H}_\mathrm{I}$, as described in Section 3.2. The objective of this phase is to minimize the upper bound on the total volume of communication. The second phase assigns the subscreens formed in the first phase to processors by finding a mapping $\mathcal{M}_\mathcal{S}$ that achieves the maximum saving in the total communication volume relative to the upper bound. Without the second phase, each subscreen $\mathcal{S}_\ell$ may be assigned to a processor P_k arbitrarily, as mentioned in Section 3.3. However, this may lead to a communication volume as high as the upper bound set by the first phase.

Fig. 5a shows an initial processor mapping for cell clusters. In the figure, the fill pattern of a cell cluster $\mathcal{C}_j$ indicates its home processor $Home(\mathcal{C}_j)$. Processors P_1, P_2, and P_3 initially store cell clusters filled with vertical lines, horizontal lines, and color, respectively. Fig. 5b shows a 3-way vertex partition



(a)

(b)

Fig. 5. (a) An initial, static cluster-to-processor mapping. (b) A 3-way vertex partition Π of $\mathcal{H}_{\text{I}}$.

Π found by the first phase. In this example, consider the trivial $\mathcal{M}(\mathcal{S}_1) = P_1$, $\mathcal{M}(\mathcal{S}_2) = P_2$, $\mathcal{M}(\mathcal{S}_3) = P_3$ mapping. With this mapping, processors P_1, P_2, and P_3 need eight, five, and seven cell clusters, but store only one, one, and two of the cell clusters they need, respectively. Hence, the total communication volume incurred by this mapping is $(8-1)+(5-1)+(7-2)=16$. However, the $\mathcal{M}(\mathcal{S}_1)=P_3$, $\mathcal{M}(\mathcal{S}_2)=P_2$, and $\mathcal{M}(\mathcal{S}_3) = P_1$ mapping incurs a total communication volume of only $(8-2)+(5-1)+(7-5)=12$. This decrease in the communication volume is mostly because subscreen $\mathcal{S}_3$ is assigned to processor P_1, which already stores most of the cell clusters needed by subscreen $\mathcal{S}_3$.

Taking this observation into account, we formulate the problem of finding the best subscreen-to-processor mapping, which achieves the highest saving in the total volume of communication, as a maximum weight bipartite matching problem. In this second phase, the K subscreens, obtained using vertex partition Π of the first phase, and the K processors in the parallel system form the two partite vertex sets $\{s_1, s_2, \ldots, s_K\}$ and $\{p_1, p_2, \ldots, p_K\}$ of a bipartite graph $\mathcal{B}$. That is, each subscreen vertex s_ℓ and processor vertex p_k represents subscreen $\mathcal{S}_\ell$ and processor P_k, respectively. A cell cluster $\mathcal{C}_j$ incurs an edge $e_{\ell k}$ between vertices s_ℓ and p_k with weight $Cost(\mathcal{C}_j)$ if $P_k = Home(\mathcal{C}_j)$ and $\mathcal{C}_j$ is needed by subscreen $\mathcal{S}_\ell$. Multiple edges between the same pair of vertices are contracted into a single edge, whose weight is equal to the sum of the weights of each contracted edge.

In this model, finding the maximum-weighted matching in $\mathcal{B}$ corresponds to finding a subscreen-to-processor mapping that achieves the highest saving in the total communication volume relative to the upper bound set by the first phase. Each edge $e_{\ell k}$ in the maximum-weighted matching assigns subscreen $\mathcal{S}_\ell$ to processor P_k, generating a subscreen-to-processor mapping $\mathcal{M}_\mathcal{S}$. The subscreen-to-processor mapping found by the second phase is an optimum solution, which minimizes the total volume of communication for the given initial cluster-to-processor mapping and the screen partition supplied by the first phase. Using the subscreen-to-processor mapping $\mathcal{M}_\mathcal{S}$ in (5) and (6), the remapping $\mathcal{M}_b$ of pixel blocks and the replication pattern $\mathcal{R}_\mathcal{C}$ of cell clusters can be calculated.

Fig. 6a shows bipartite graph $\mathcal{B}$ constructed for the sample case of Fig. 5. In the figure, bold edges indicate the maximum-weighted matching, composed of edges e_{12}, e_{23}, and e_{31} with weights 5, 3, and 5, respectively. The subscreen-to-processor mapping corresponding to this matching is $\mathcal{M}(\mathcal{S}_1) = P_2$, $\mathcal{M}(\mathcal{S}_2) = P_3$, and $\mathcal{M}(\mathcal{S}_3) = P_1$. With this mapping, the total volume of communication is $(8+5+7)-(5+3+5)=7$ with a saving of 13 over the upper bound 20, set by the first phase. Fig. 6b shows the remapping of pixel blocks to processors. Replicated cell clusters are illustrated by the square-filled pattern.

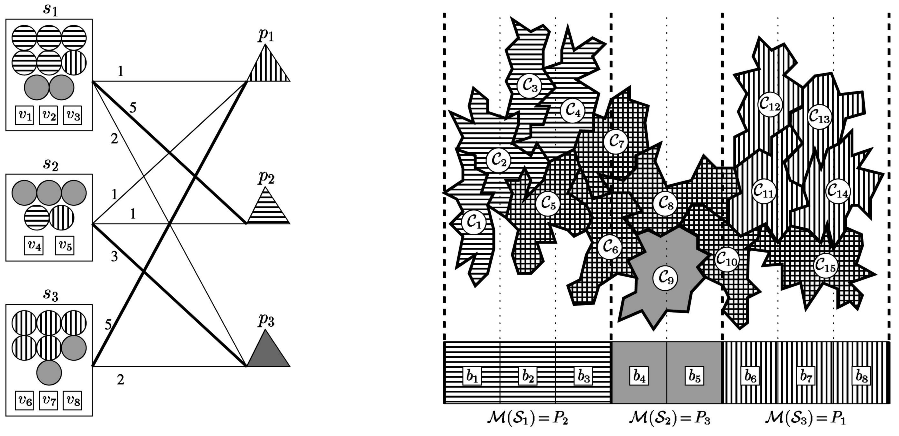
3.3.2 One-Phase Remapping Model

An important point not considered by the first-phase of the two-phase model is that, in our framework, each cell cluster $\mathcal{C}_j$ is originally owned by a home processor $P_k = Home(\mathcal{C}_j)$ and no communication is necessary to replicate $\mathcal{C}_j$ on P_k. Consider net n_9 in Fig. 5b. If $\mathcal{S}_2$ is assigned to P_1, $\mathcal{C}_9$ must be transferred from its home processor P_3 to P_1 introducing some communication overhead. However, if $\mathcal{S}_2$ is assigned to P_3, no data transfer is necessary for replication at P_3 since P_3 already has $\mathcal{C}_9$ in its memory.

In order to accurately model the total volume of communication within our framework, the initial cluster-to-processor mapping must be supplied into the model. In the one-phase model, we use a remapping hypergraph $\tilde{\mathcal{H}}_{\text{R}} = (\tilde{\mathcal{V}}, \mathcal{N})$, which is obtained by augmenting the interaction hypergraph $\mathcal{H}_{\text{I}}$, proposed earlier in Section 3.2, with some vertex and pin additions. Vertex set $\tilde{\mathcal{V}}$ of the remapping hypergraph $\tilde{\mathcal{H}}_{\text{R}}$ is formed by introducing a set $\mathcal{P} = \{p_1, p_2, \ldots, p_K\}$ of K processor vertices into $\mathcal{H}_{\text{I}}$, that is, $\tilde{\mathcal{V}} = \mathcal{V} \cup \mathcal{P}$. Each processor vertex p_k represents a processor P_k belonging to the parallel system and has no weight. Also, new pins are added to the pin set of $\mathcal{H}_{\text{I}}$ such that a processor vertex p_k is a pin of a net n_j if cell cluster $\mathcal{C}_j$ is initially assigned to processor P_k, that is, $Home(\mathcal{C}_j) = P_k$.

In the proposed model, a K-way vertex partition $\tilde{\Pi} = \{\tilde{\mathcal{V}}_1, \tilde{\mathcal{V}}_2, \ldots, \tilde{\mathcal{V}}_K\}$ of $\tilde{\mathcal{H}}_{\text{R}}$ is said to be feasible if it satisfies the mapping constraint

$$\left|\tilde{\mathcal{V}}_\ell \bigcap \mathcal{P}\right| = 1 \text{ , for } \ell = 1, 2, \ldots, K, \tag{7}$$


s_1
p_1
v_1 v_2 v_3
1
5
2
s_2
p_2
1
1
3
v_4 v_5
s_3
p_3
5
2
v_6 v_7 v_8
$\mathcal{C}_1$ $\mathcal{C}_2$ $\mathcal{C}_3$ $\mathcal{C}_4$ $\mathcal{C}_5$ $\mathcal{C}_6$ $\mathcal{C}_7$ $\mathcal{C}_8$ $\mathcal{C}_9$ $\mathcal{C}_{10}$ $\mathcal{C}_{11}$ $\mathcal{C}_{12}$ $\mathcal{C}_{13}$ $\mathcal{C}_{14}$ $\mathcal{C}_{15}$
b_1 b_2 b_3 b_4 b_5 b_6 b_7 b_8
$\mathcal{M}(\mathcal{S}_1) = P_2$
$\mathcal{M}(\mathcal{S}_2) = P_3$
$\mathcal{M}(\mathcal{S}_3) = P_1$

(a)

(b)

Fig. 6. (a) Bipartite graph $\mathcal{B}$ created using the initial cluster-to-processor mapping and vertex partition Π in Fig. 5. (b) Mapping of subscreens to processors, using the maximum-weighted matching in (a).

that is, each part $\tilde{\mathcal{V}}_\ell$ contains exactly one processor vertex p_k. A feasible partition $\tilde{\Pi}$ induces a remapping $\mathcal{M}_b$ for pixel blocks such that all pixel blocks represented by the nonprocessor vertices in a part are remapped to the processor represented by the unique processor vertex in that part. That is, a pixel block b_i, whose corresponding vertex v_i is in $\tilde{\mathcal{V}}_\ell$, is remapped to processor P_k if processor vertex p_k is in $\tilde{\mathcal{V}}_\ell$.

Another point omitted by the two-phase model is that communication overheads of processors vary during the data replication and some processors spend more time on communication. Taking this fact into consideration, the one-phase model aims to balance the estimated time for incoming data communication plus the time for local rendering of each processor. In this model, each vertex v_i is assigned a weight w_i which is equal to the estimated time $PBload(b_i) \times t_r$ for rendering pixel block b_i. Here, t_r is the time cost for taking a single sample within a data cell. As the cost c_j of a net n_j, the estimated communication time $Cost(\mathcal{C}_j) \times t_c$ of cell cluster $\mathcal{C}_j$ is assigned. Here, t_c is the per-byte cost for receiving a cell cluster, unpacking it, and creating the necessary data structures.

In this model, we modify the conventional part weight definition (2) and define the weight W'_k of a part $\tilde{\mathcal{V}}_k$ as the sum of the weights of vertices within $\tilde{\mathcal{V}}_k$ plus the sum of the costs of cut nets that connect $\tilde{\mathcal{V}}_k$ but not processor vertex p_k, i.e.,

$$W'_k = \sum_{v_i \in \tilde{\mathcal{V}}_k} w_i + \sum_{\tilde{\mathcal{V}}_k \in \Lambda_j \wedge p_k \notin n_j} c_j, \tag{8}$$

where the second summation term is the incoming message volume overhead of processor P_k. Note that we prefer to balance this overhead since, in our framework, outgoing message volume overheads of processors are already balanced.

After this setting, the remapping problem reduces to the problem of finding a feasible K-way partition $\tilde{\Pi} = \{\tilde{\mathcal{V}}_1, \tilde{\mathcal{V}}_2, \ldots, \tilde{\mathcal{V}}_K\}$ of $\tilde{\mathcal{H}}_\mathrm{R}$, satisfying the mapping constraint. Maintaining the balance among parts corresponds to maintaining the time balance among processors during the replication plus local rendering phases. In $\tilde{\Pi}$, consider a net n_j which connects processor vertex p_k. In this model, net n_j indicates that processor P_k should replicate cell cluster $\mathcal{C}_j$ on all processors responsible from each subscreen corresponding to a vertex part in Λ_j, excluding the processor itself, i.e., processor P_k. Since cell cluster $\mathcal{C}_j$ must be replicated on $\lambda_j - 1$ processors, the communication volume incurred by net n_j is $c_j(\lambda_j - 1)$. Note that internal nets incur no communication. Hence, by minimizing the cost $\chi(\tilde{\Pi})$ (3) of partition $\tilde{\Pi}$, the model exactly minimizes the total volume of communication.

Fig. 7 shows the remapping hypergraph $\tilde{\mathcal{H}}_\mathrm{R}$, constructed for a 3-processor system by augmenting the interaction hypergraph $\mathcal{H}_\mathrm{I}$ of Fig. 5b. In $\tilde{\mathcal{H}}_\mathrm{R}$, triangles represent processor vertices, corresponding to processors. A dotted line, connecting a processor vertex p_k and a net n_j, indicates that $P_k = Home(\mathcal{C}_j)$. Fig. 7 also shows a 3-way vertex partition $\tilde{\Pi}$ of $\tilde{\mathcal{H}}_\mathrm{R}$, where vertex parts $\tilde{\mathcal{V}}_1$, $\tilde{\mathcal{V}}_2$, and $\tilde{\mathcal{V}}_3$ contain processor vertices p_2, p_3, and p_1, respectively. In Fig. 7, consider cut net n_8 with connectivity set $\Lambda_8 = \{\tilde{\mathcal{V}}_1, \tilde{\mathcal{V}}_2, \tilde{\mathcal{V}}_3\}$. Processor vertex p_1, corresponding to home processor $P_1 = Home(\mathcal{C}_8)$ of cell cluster $\mathcal{C}_8$, is in vertex part $\tilde{\mathcal{V}}_3$. Hence, processor P_1 is responsible from replicating cell cluster $\mathcal{C}_8$ on processors P_2 and P_3, determined by processor vertices p_2 and p_3 in the other two vertex parts $\tilde{\mathcal{V}}_1$ and $\tilde{\mathcal{V}}_2$. There are five cut nets n_5, n_6, n_7,

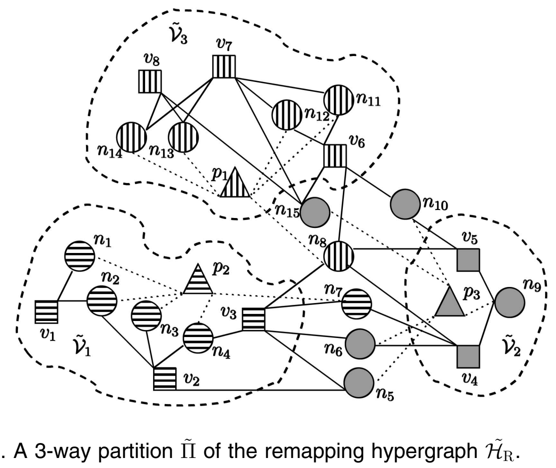


Fig. 7. A 3-way partition $\tilde{\Pi}$ of the remapping hypergraph $\tilde{\mathcal{H}}_\mathrm{R}$.

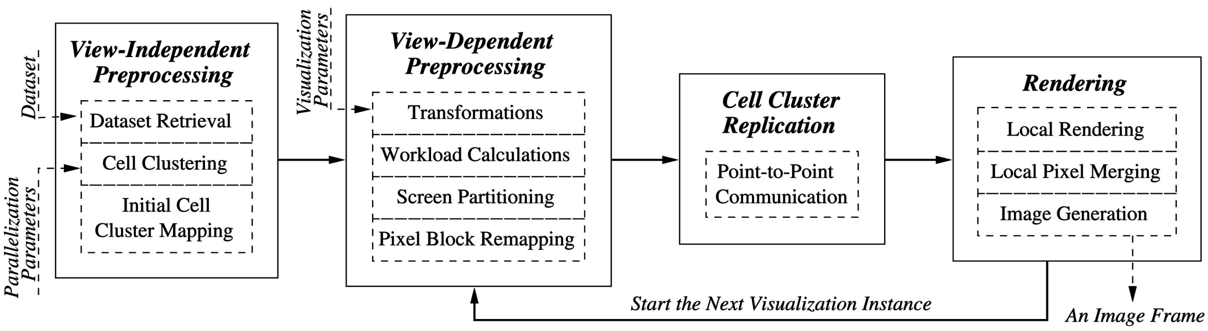


Fig. 8. The proposed adaptive IS-parallel DVR algorithm.

n_{10}, and n_{15} with connectivity 2, each incurring a communication cost of 1. The other nine nets are internal and incur no communication. Hence, the total communication volume is accurately calculated as $\chi(\tilde{\Pi}) = 9 \times 0 + 5 \times 1 + 1 \times 2 = 7$.

Existing hypergraph partitioning tools can be enhanced to maintain the mapping (7) and balancing constraints in the model. The mapping constraint can also be maintained by using the state-of-the-art hypergraph partitioning tools that support the fixed vertices feature [10], [45]. This widely used feature [2] allows prespecified vertices to be fixed to given parts and can be exploited to fix each vertex p_k to a part $\tilde{\mathcal{V}}_k$, for $k = 1, 2, \ldots, K$.

4 IS-Parallel DVR Algorithm

The proposed parallel DVR algorithm (Fig. 8) starts with view-independent preprocessing. This phase is followed by three consecutive phases, repeated for each visualization instance: view-dependent preprocessing, cell cluster replication, and rendering.

4.1 View-Independent Preprocessing

This phase, performed just once at the very beginning of the whole visualization process, carries out the view-independent operations, which include reading the data set from the disk, clustering data cells, and mapping cell clusters to processors. Since most scientific simulations are carried out on parallel systems, we assume that each local disk stores a contiguous portion of the data. Hence, processors read subvolumes in parallel.

After reading their data, each processor concurrently creates the view-independent clustering graph of its local data using the adjacency information between cells. Then, the clustering scheme of Section 2.2 is applied on the local graphs, and each processor obtains a set of cell clusters. Since the volume is currently stored in a distributed manner, creation and partitioning of a global visualization graph may be expensive. Hence, a local cell clustering scheme, which reduces the overhead of clustering, is preferred. We use the state-of-the-art graph partitioning tool MeTiS [23] for partitioning the clustering graphs.

After cell clustering, an initial cluster-to-processor mapping is found. This mapping is important in that all following remapping phases use this initial data mapping. Even if a cell cluster may be temporarily replicated on other processors after remapping, it is statically owned by only its home processor. This static owner keeps the cell cluster throughout the whole visualization process. The reason for this static assignment scheme is the drastic variation in preprocessing costs of cell clusters, which requires balancing the preprocessing overhead of processors. During the initial cell cluster distribution step, cell clusters are assigned to processors such that processors have roughly equal scan conversion costs. The best-fit-decreasing heuristic used in solving the K-feasible bin-packing problem [22] is adapted to obtain such an initial distribution. Cell clusters are assigned to K processors in decreasing scan-conversion cost order, where best-fit criterion corresponds to assigning a cell cluster to a processor which currently has the minimum total scan-conversion cost.

4.2 View-Dependent Preprocessing

This phase contains the steps that try to adapt the computational structure according to changing view-dependent visualization parameters: calculation of the screen workload, partitioning of the screen, and remapping of pixel blocks. During the screen workload calculations, the interaction between the volume and the screen is computed, and the rendering load distribution on the screen is estimated. That is, the interaction between cell clusters and pixel blocks is found, and the rendering loads of pixel blocks are computed.

The screen partitioning and remapping steps use the proposed models. The interaction between a processor's local data and the screen is stored as a local hypergraph on the processor. Since each processor owns a portion of the whole volume, only the local hypergraphs can be created. These hypergraphs are then merged into a global hypergraph, which represents the interaction of the whole volume with the screen. For this purpose, an all-to-all broadcast operation, in which each processor sends its local hypergraph to others, is performed among processors. By combining the common vertices in local hypergraphs, a global hypergraph, which is replicated on all processors, is obtained. During the global hypergraph creation, the pixel blocks having no sampling load are discarded from the hypergraph. The fixed vertices in the one-phase model are also added at this step.

Finally, a pixel-to-processor remapping is found using one of the proposed remapping models. In the implementation, the sequential hypergraph partitioning tool PaToH [10] is used for partitioning the global hypergraph. Since this hypergraph is small in size, the multilevel paradigm is abandoned and the flat hypergraph is partitioned without

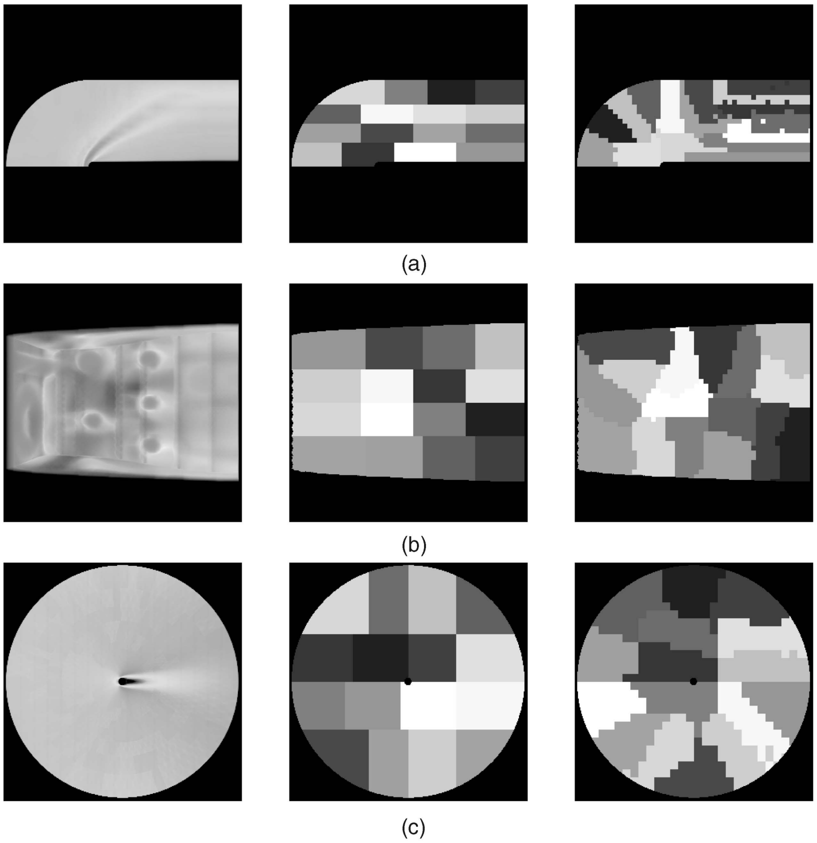


Fig. 9. Example renderings of the data sets and the 16-way screen partitions produced by the jagged-partitioning-based and hypergraph-partitioning-based screen partitioning models. (a) Blunt fin (40,960 vertices, 187,395 cells), (b) combustion chamber (47,025 vertices, 215,040 cells), and (c) oxygen post (109744 vertices, 513,375 cells).

further coarsening. This considerably decreases the preprocessing overhead due to hypergraph partitioning. The solution qualities are not affected much since we run the partitioner at each processor with a different seed and pick the best solution (i.e., the lowest imbalance rate or the smallest total communication volume) for remapping. In the two-phase model, a maximum-weighted matching is obtained using the Kuhn-Munkres algorithm [15].

4.3 Cell Cluster Replication

Before the rendering starts, cell clusters are temporarily replicated in the parallel system according to the replication pattern $\mathcal{R}_{\mathcal{C}}$, induced by the pixel-to-processor remapping. The replication is performed by sending cell clusters from their home processors to the processors where they are needed via point-to-point communication between processors.

4.4 Rendering

After cell clusters are replicated, processors are ready to locally render their assigned pixel blocks in parallel. A ray is shot from each pixel covered by the projected areas of front-facing external faces of cell clusters only if the pixel belongs to the subscreen assigned to the processor. The rays are followed through the volume by utilizing the adjacency information stored in cells and cell clusters, eliminating the need to scan convert all front-facing faces on surfaces of cell clusters. Although it is possible to have nonconvex cell clusters as a result of the clustering algorithm, this does not cause an increase in the number of ray segments created. Existence of such nonconvexities is eliminated due to data replication and, hence, processors act as if rendering a whole and convex subvolume.

However, because of the nonconvexities in the nature of the volumetric data, the use of ray buffers is still required. The generated ray segments are accumulated in the corresponding ray buffers. For each pixel, a separate ray buffer is kept. The accumulated color and opacity values are inserted into their corresponding ray buffers in the sorted order of their increasing z coordinates. Later, the values in ray buffers are composited using the traditional composition formulas in a separate local pixel merging phase. Since, at this stage, all processors have a subimage, an all-to-one communication operation is performed, and the whole and final image for the current visualization instance is generated in one of the processors. After the rendering, each processor deallocates the memory reserved for the temporarily replicated cell clusters for which it is not a home processor.

5 Experimental Results

Experiments are conducted on three data sets (*Blunt Fin*, *Combustion Chamber*, and *Oxygen Post*), obtained from NASA Ames Research Center [35]. These data sets are the results of CFD simulations and are originally curvilinear. The unstructured data sets used in the experiments are obtained using the tetrahedralization techniques described in [20] and [44]. Properties of the data sets are summarized in the caption of Fig. 9, which displays our renderings and

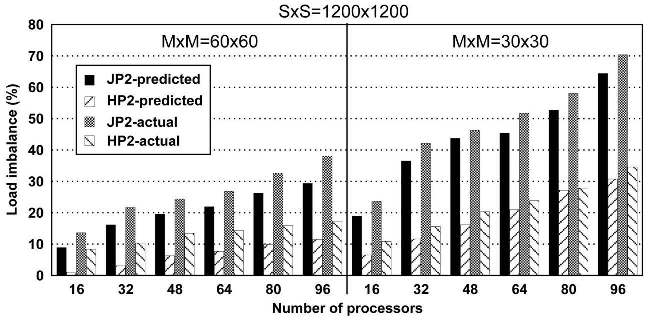


Fig. 10. Averages of the predicted and actual sampling load imbalance values.

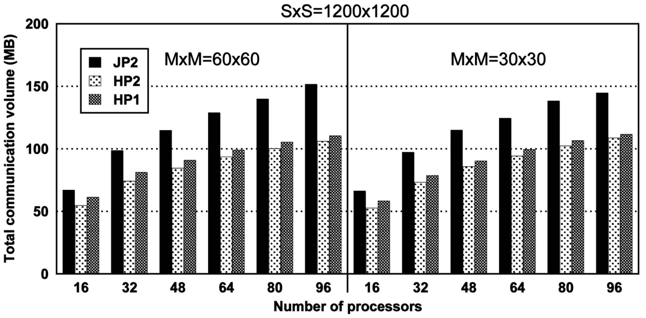


Fig. 11. Averages of the total communication volumes in cell cluster replication.

the screen partitions produced by different models. In each row, the first image is the rendering obtained using the standard viewing parameters. The second and third images illustrate the 16-way screen partitions produced by the jagged-partitioning-based model [27] and the proposed hypergraph-partitioning-based model, respectively. In these images, each color represents a subscreen, rendered by a distinct processor.

The rendering platform is a 32-node PC cluster interconnected by a Gigabit Ethernet switch. Each node contains an Intel Pentium IV 2.6 GHz processor, 1 GB of RAM, and runs Debian/GNU Linux. The DVR algorithms are implemented in C using LAM/MPI [7].

In the experiments, each of the three data sets is rendered using five different viewing parameter sets. Hence, the values reported for an experiment represent the averages of the values obtained from 15 different executions of the parallel DVR algorithm. The viewing parameter sets contain different view-point coordinates and viewing directions. These values are selected such that different computational characteristics of the data sets are reflected as much as possible. In each experiment, processors are assigned $C = 10$ cell clusters, which is an empirically found number. As mentioned, to make the view-dependent preprocessing overhead affordable, coarse meshes of varying sizes are imposed on the screen. Three different remapping models are compared: jagged-partitioning-based (JP2), two-phase hypergraph-partitioning-based (HP2), and one-phase hypergraph-partitioning-based (HP1) models. The JP2 model is implemented as a two-phase model, in which the jagged partitioning algorithm is used in the first phase for screen partitioning while the matching algorithm is used in the second phase for subscreen-to-processor matching, similar to the second phase of the HP2 model. Hence, it is an enhanced version of the model in [27].

Two sets of experiments are conducted. The first set of experiments tests solution qualities of the remapping models in load balancing and minimization of the total communication volume. These experiments are carried out at large numbers of virtual processors by assigning more than one executable to available processors. In the second set of experiments, practical aspects of our parallel implementation are investigated. Execution time of a single visualization instance and view-dependent preprocessing time are dissected into their components, and speedup values are recorded at the available numbers of processors.

5.1 Experiments on Remapping Quality

These experiments are conducted on 16, 32, 48, 64, 80, and 96 virtual processors using a screen resolution of $S \times S = 1,200 \times 1,200$. Two coarse mesh resolutions, $M \times M = 30 \times 30$ and $M \times M = 60 \times 60$, are tried. Fig. 10 shows the predicted and actual load imbalances in sampling amounts of processors for JP2 and HP2, respectively. The predicted imbalance values are the ones expected by the partitioning algorithm. The actual imbalance values are the sampling imbalance values observed in parallel rendering. No results are displayed for HP1 since this model tries to directly balance processors' total rendering time including the communication overhead. According to Fig. 10, the actual values are always higher than the predicted values in both models. This is due to the estimation errors made in screen workload calculations. As the number of processors increases, the predicted values get closer to the actual values. This is because of the increase in the workload estimation quality, which is caused by the increase in the number of cell clusters and, hence, the decrease in cell cluster volumes. In general, HP2 performs significantly better than JP2 in terms of load balancing. For example, with a mesh resolution of $M \times M = 60 \times 60$ and 96 virtual processors, JP2 results in a load imbalance of 38.1 percent. With the same parameters, the load imbalance for HP2 is 17.3 percent. As expected, the imbalance values almost linearly increase with increasing number of processors. When the mesh resolution is decreased from $M \times M = 60 \times 60$ to $M \times M = 30 \times 30$, both models perform worse in load balancing due to the decrease in the number of pixel blocks and, hence, the size of the solution space.

Fig. 11 displays the total volume of communication in cell cluster replication for varying numbers of processors and mesh resolutions. With a mesh resolution of $M \times M = 60 \times 60$, using 96 virtual processors, HP2 and HP1 result in around 30 percent and 27 percent less total communication volume than JP2, respectively. When the number of processors is increased from 16 to 96, the volume almost doubles. This points out the importance of minimizing this overhead at large numbers of processors. If the mesh resolution is reduced to $M \times M = 30 \times 30$, there occurs a slight decrease in the total volume of communication. This is due to the decrease in the total length of subscreen boundaries and, hence, the amount of overlaps between cell clusters and subscreen boundaries. Since coarse mesh resolution affects both the load imbalance

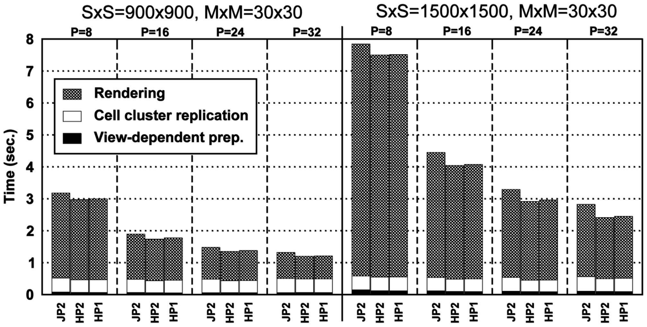


Fig. 12. Dissection of the average execution time of a single visualization instance.

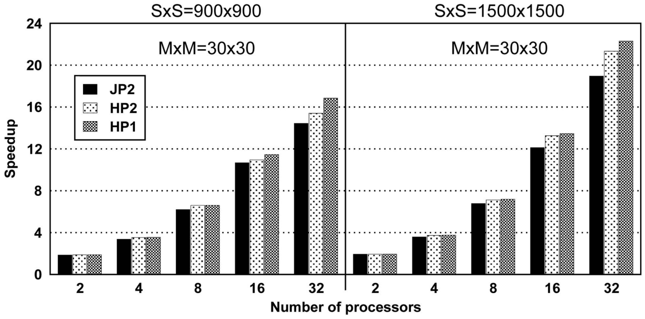


Fig. 14. Average speedups achieved by JP2, HP2, and HP1 in parallel rendering.

and communication volume, it can be used to trade off between these two parallelization overheads. In general, JP2 incurs the highest total volume of communication, while HP2 is the best at minimizing this overhead. Although HP1 accurately calculates the total volume of communication, it produces results inferior to HP2. This is basically due to the fact that the recursive bisection paradigm employed in PaToH is not well-suited to handle a hypergraph with fixed vertices.

5.2 Experiments on Parallel Performance

Experiments verifying the practical performance of the models are carried out on the available numbers of processors 8, 16, 24, and 32. In the figures related with time dissection of different phases, averages of the maximum execution times of processors in each phase are shown. In Fig. 12, the average parallel execution time for a single visualization instance is dissected into three components as view-dependent preprocessing, cell cluster replication, and rendering. The two cases examined are $S \times S = 900 \times 900$ and $S \times S = 1,500 \times 1,500$ with $M \times M = 30 \times 30$. According to Fig. 12, view-dependent preprocessing and rendering times increase with increasing screen resolution. The cell cluster replication time is not affected much from the variation in the screen resolution. The rendering time falls with increasing number of processors, whereas the replication time remains almost the same. At 32 processors, for the $S \times S = 900 \times 900$ resolution case, the replication time takes more than one third of the total visualization time. This indicates that the replication step has the potential to form a bottleneck on the scalability at large numbers of processors.

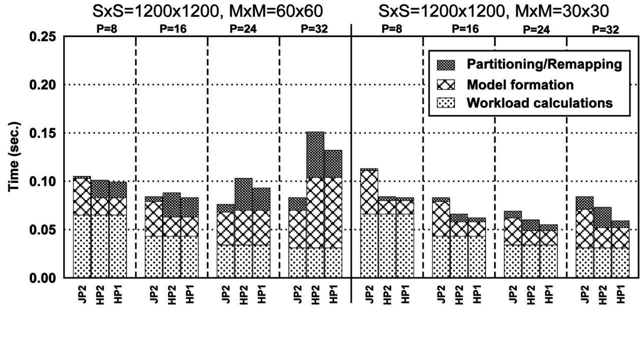


Fig. 13. Dissection of the average view-dependent preprocessing time.

In Fig. 13, the view-dependent preprocessing time is dissected into three components as screen workload calculations, model formation, and partitioning/remapping. In HP2 and HP1, the model formation step represents creation of the global hypergraph from local hypergraphs via communication among processors. In JP2, this step represents the distributed global sum operation on local screen workloads of processors. With increasing number of processors, the duration of screen workload calculations tends to decrease since the total surface area to be scan converted per processor gets smaller, whereas partitioning/remapping times increase and are affected from the coarse mesh resolution. Decreasing the mesh resolution from $M \times M = 60 \times 60$ to $M \times M = 30 \times 30$ decreases the number of pixel blocks and, hence, the partitioning/remapping time. The partitioning time for HP1 is slightly less than that of HP2 since the partitioning heuristics in HP1 converge earlier.

Fig. 14 shows the speedups achieved at 2, 4, 8, 16, and 32 processors. On 32 processors, with a screen resolution of $S \times S = 900 \times 900$ and a coarse mesh resolution of $M \times M = 30 \times 30$, speedups are 14.44, 15.41, and 16.85 for JP2, HP2, and HP1, respectively. At the same number of processors and coarse mesh resolution, with a screen resolution of $S \times S = 1,500 \times 1,500$, speedups are respectively 18.96, 21.34, and 22.30. HP1 is able to render an image with a resolution of $S \times S = 900 \times 900$ in 1.135 seconds on the average, i.e., 14.3 percent faster than JP2. Moreover, it is observed that the increasing screen resolution and number of processors favor the proposed models. It should be noted that HP1 achieves better speedups than HP2 although the sum of the execution times for individual phases of HP1 are higher than that of HP2 (Fig. 12). This is because HP1 tries to assign less communication volume overhead to computationally loaded processors and vice versa. The speedup gap between the HP-based models and JP2 are less than the ones suggested by the theoretical results. This is mainly due to the implicit tendency of JP2 toward creating screen partitions that induce low concurrent communication volume.

5.3 Comparison with an OS-Parallel DVR Algorithm

In this section, we compare the performance of HP1 with our recently proposed adaptive, OS-parallel DVR algorithm (OS) [4]. In this model, the computational structure in the data space is represented as a graph, where the clusters of cells correspond to vertices and faces shared between cell clusters correspond to edges. The remapping problem in OS parallelization is formulated as a graph partitioning

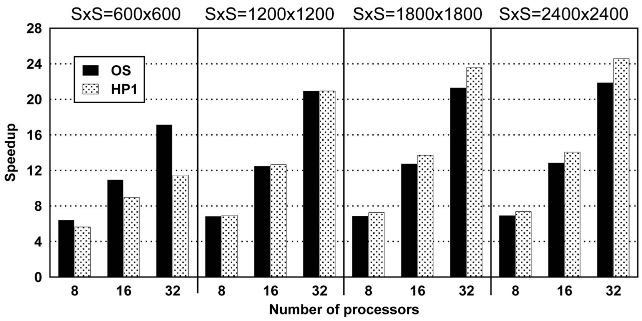


Fig. 15. Average speedups achieved by OS and HP1 in parallel rendering.

problem by introducing a set of fixed processor vertices into the graph. Our enhanced version of MeTiS is used to minimize the communication volume overheads in data remapping and global pixel merging while balancing the rendering loads of processors. The details can be found in [4]. Fig. 15 provides the speedups achieved by OS and HP1. In all executions of HP1, coarse mesh resolution is $M \times M = 30 \times 30$. According to Fig. 15, for low screen resolutions, OS achieves better speedups than HP1. For example, at a resolution of $S \times S = 600 \times 600$ with 32 processors, speedups are 17.47 and 11.48 for OS and HP1, respectively. At $S \times S = 1,200 \times 1,200$, both algorithms display a similar performance. As the resolution is further increased, HP1 begins to achieve better speedups. For example, at a resolution of $S \times S = 2,400 \times 2,400$ with 32 processors, speedups are 21.86 and 24.57 for OS and HP1, respectively. The scalability problem of OS at high screen resolutions is due to the global pixel merging overhead. This overhead, which proportionally increases with the screen resolution, is not present in HP1. We report further experiments and observations in [9].

6 Conclusion

The experiments show that, compared to the previous models, the HP-based remapping models yield superior speedup values by obtaining better load balance and incurring less total communication volume. We believe that as new partitioning heuristics are developed and existing hypergraph partitioning tools are improved, solution qualities of the proposed models will also improve. We should also note that the final target in parallel DVR is a hybrid, adaptive algorithm in which both IS and OS will be partitioned for higher scalability. In this respect, the proposed work is a good frontier for this hybrid algorithm.

Acknowledgments

This work is partially supported by the Scientific and Technological Research Council of Turkey under projects EEEAG-103E028 and EEEAG-105E065.

References

[1] C.J. Alpert and A.B. Kahng, "Recent Directions in Netlist Partitioning: A Survey," *VLSI J.*, vol. 19, nos. 1-2, pp. 1-81, 1995.

[2] C.J. Alpert, A.E. Caldwell, A.B. Kahng, and I.L. Markov, "Hypergraph Partitioning with Fixed Vertices," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems,* vol. 19, no. 2, pp. 267-272, 2000.

[3] C. Aykanat, A. Pinar, and Ü.V. Çatalyürek, "Permuting Sparse Rectangular Matrices into Block-Diagonal Form," *SIAM J. Scientific Computing,* vol. 25, no. 6, pp. 1860-1879, 2004.

[4] C. Aykanat, B.B. Cambazoglu, F. Findik, and T.M. Kurç, "Adaptive Decomposition and Remapping Algorithms for Object-Space-Parallel Direct Volume Rendering of Unstructured Grids," *J. Parallel and Distributed Computing,* in press.

[5] C. Berge, *Graphs and Hypergraphs.* North-Holland, 1973.

[6] H. Berk, C. Aykanat, and U. Güdükbay, "Direct Volume Rendering of Unstructured Grids," *Computers & Graphics,* vol. 27, no. 3, pp. 387-406, 2003.

[7] G. Burns, R. Daoud, and J. Vaigl, "LAM: An Open Cluster Environment for MPI," *Proc. Supercomputing Symp. '94,* pp. 379-386, 1994.

[8] B.B. Cambazoglu and C. Aykanat, "Image-Space-Parallel Direct Volume Rendering on a Cluster of PCs," *Proc. 18th Int'l Symp. Computer and Information Sciences,* pp. 457-464, 2003.

[9] B.B. Cambazoglu and C. Aykanat, "Hypergraph-Partitioning-Based Remapping Models for Image-Space-Parallel Direct Volume Rendering of Unstructured Grids," Technical Report, BU-CE-0503, Dept. of Computer Eng., Bilkent Univ., 2005.

[10] Ü.V. Çatalyürek and C. Aykanat, "PaToH: Partitioning Tool for Hypergraphs," technical report, Dept. of Computer Eng., Bilkent Univ., 1999.

[11] Ü.V. Çatalyürek and C. Aykanat, "Hypergraph-Partitioning-Based Decomposition for Parallel Sparse-Matrix Vector Multiplication," *IEEE Trans. Parallel and Distributed Systems,* vol. 10, no. 7, pp. 673-693, July 1999.

[12] J. Challinger, "Parallel Volume Rendering for Curvilinear Volumes," *Proc. IEEE Scalable High Performance Computing Conf.,* pp. 14-21, 1992.

[13] J. Challinger, "Scalable Parallel Volume Raycasting for Nonrectilinear Computational Grids," *Proc. IEEE/ACM Parallel Rendering Symp.,* pp. 81-88, 1993.

[14] J. Challinger, "Scalable Parallel Direct Volume Rendering for Nonrectilinear Computational Grids," PhD thesis, Univ. of California, 1993.

[15] G. Chartrand and O.R. Oellermann, *Applied and Algorithmic Graph Theory.* McGraw-Hill, 1993.

[16] A. Dasdan and C. Aykanat, "Two Novel Multiway Circuit Partitioning Algorithms Using Relaxed Locking," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems,* vol. 16, no. 2, pp. 169-178, 1997.

[17] T.T. Elvins, "A Survey of Algorithms for Volume Visualization," *ACM SIGGRAPH Computer Graphics,* vol. 26, no 3, pp. 194-201, 1992.

[18] R. Farias, J. Mitchell, and C.T. Silva, "ZSWEEP: An Efficient and Exact Projection Algorithm for Unstructured Volume Rendering," *Proc. ACM/IEEE Volume Visualization and Graphics Symp.,* pp. 91-99, 2000.

[19] R. Farias and C.T. Silva, "Parallelizing the ZSWEEP Algorithm for Distributed-Shared Memory Architectures," *Proc. Int'l Volume Graphics Workshop '01,* pp. 181-192, 2001.

[20] M.P. Garrity, "Ray-Tracing Irregular Volume Data," *ACM SIGGRAPH Computer Graphics,* vol. 24, no. 5, pp. 35-40, 1990.

[21] C. Hofsetz and K.-L. Ma, "Multi-Threaded Rendering Unstructured-Grid Volume Data on the SGI Origin 2000," *Proc. Third Eurographics Workshop Parallel Graphics and Visualization,* pp. 91-99, 2000.

[22] E. Horowitz and S. Sahni, *Fundamentals of Computer Algorithms.* Potomac: Computer Science Press, 1978.

[23] G. Karypis and V. Kumar, "MeTiS: A Software Package for Partitioning Unstructured Graphs, Partitioning Meshes and Computing Fill-Reducing Orderings of Sparse Matrices," technical report, Dept. of Computer Science, Univ. of Minnesota, 1998.

[24] G. Karypis and V. Kumar, "Multilevel k-Way Partitioning Scheme for Irregular Graphs," *J. Parallel and Distributed Computing,* vol. 48, no. 1, pp. 96-129, 1998.

[25] G. Karypis and V. Kumar, "hMETIS: A Hypergraph Partitioning Package," technical report, Dept. of Computer Science, Univ. of Minnesota, 1998.

[26] K. Koyamada, "Fast Traversal of Irregular Volumes," *Visual Computing, Integrating Computer Graphics with Computer Vision,* T.L. Kunii, ed., pp. 295-312, Springer-Verlag, 1992.

[27] H. Kutluca, T.M. Kurç, and C. Aykanat, "Image-Space Decomposition Algorithms for Sort-First Parallel Volume Rendering of Unstructured Grids," *J. Supercomputing,* vol. 15, no. 1, pp. 51-93, 2000.

[28] T. Lengauer, *Combinatorial Algorithms for Integrated Circuit Layout.* John Wiley and Sons, 1990.

[29] M. Levoy, "Display of Surfaces from Volume Data," *IEEE Computer Graphics and Implementations,* vol. 8, no. 3, pp. 29-37, 1988.

[30] W.-S. Lin, R.W.H. Lau, K. Hwang, X. Lin, and P.Y.S. Cheung, "Adaptive Parallel Rendering on Multiprocessors and Workstation Clusters," *IEEE Trans. Parallel and Distributed Systems,* vol. 12, no. 3, pp. 241-258, Mar. 2001.

[31] K.-L. Ma, "Parallel Volume Ray-Casting for Unstructured-Grid Data on Distributed Memory Multicomputers," *Proc. Parallel Rendering Symp.,* pp. 23-30, 1995.

[32] K.-L. Ma and T.W. Crockett, "A Scalable Parallel Cell-Projection Volume Rendering Algorithm for Three-Dimensional Unstructured Data," *Proc. Parallel Rendering Symp.,* pp. 95-104, 1997.

[33] S. Molnar, M. Cox, D. Ellsworth, and H. Fuchs, "A Sorting Classification of Parallel Rendering," *IEEE Computer Graphics and Applications,* vol. 14, no. 4, pp. 23-32, 1994.

[34] C. Mueller, "The Sort-First Rendering Architecture for High-Performance Graphics," *Proc. Symp. Interactive 3D Graphics,* pp. 75-84, 1995.

[35] NASA Data Set Archive, http://www.nas.nasa.gov/Research/Datasets/datasets.html, 2004.

[36] L. Oliker and R. Biswas, "PLUM: Parallel Load Balancing for Adaptive Unstructured Meshes," *J. Parallel and Distributed Computing,* vol. 52, no. 2, pp. 150-177, 1998.

[37] C.-W. Ou and S. Ranka, "Parallel Incremental Graph Partitioning," *IEEE Trans. Parallel and Distributed Systems,* vol. 8, no. 8, pp. 884-896, 1997.

[38] M.E. Palmer and S. Taylor, "Rotation Invariant Partitioning for Concurrent Scientific Visualization," *Proc. Parallel Computational Fluid Dynamics,* 1994.

[39] R. Samanta, T. Funkhouser, K. Li, and J.P. Singh, "Sort-First Parallel Rendering with a Cluster of PCs," *Proc. SIGGRAPH Technical Sketches,* 2000.

[40] R. Samanta, T. Funkhouser, K. Li, and J.P. Singh, "Hybrid Sort-First and Sort-Last Parallel Rendering with a Cluster of PCs," *Proc. SIGGRAPH/Eurographics Workshop Graphics Hardware,* pp. 99-108, 2000.

[41] R. Samanta, T. Funkhouser, and K. Li, "Parallel Rendering with K-Way Replication," *Proc. IEEE Symp. Parallel and Large-Data Visualization and Graphics,* pp. 75-84, 2001.

[42] K. Schloegel, G. Karypis, and V. Kumar, "Multilevel Diffusion Schemes for Repartitioning of Adaptive Meshes," *J. Parallel and Distributed Computing,* vol. 47, no. 2, pp. 109-124, 1997.

[43] K. Schloegel, G. Karypis, and V. Kumar, "Wavefront Diffusion and LMSR: Algorithms for Dynamic Repartitioning of Adaptive Meshes," *IEEE Trans. Parallel and Distributed Systems,* vol. 12, no. 5, pp. 451-466, May 2001.

[44] P. Shirley and A. Tuchman, "A Polygonal Approximation to Direct Scalar Volume Rendering," *ACM SIGGRAPH Computer Graphics,* vol. 24, no. 5, pp. 63-70, 1990.

[45] B. Ucar and C. Aykanat, "Encapsulating Multiple Communication-Cost Metrics in Partitioning Sparse Rectangular Matrices for Parallel Matrix-Vector Multiplies," *SIAM J. Scientific Computing,* vol. 25, no. 6, pp. 1837-1859, 2004.

[46] C. Walshaw, M. Cross, and M.G. Everett, "Parallel Dynamic Graph Partitioning for Adaptive Unstructured Meshes," *J. Parallel and Distributed Computing,* vol. 47, no. 2, pp. 102-108, 1997.

[47] J. Wilhelms, A.V. Gelder, P. Tarantino, and J. Gibbs, "Hierarchical and Parallelizable Direct Volume Rendering for Irregular and Multiple Grids," *Proc. IEEE Visualization Conf. '96,* pp. 57-64, 1996.

[48] P.L. Williams, "Interactive Direct Volume Rendering of Curvilinear and Unstructured Data," PhD thesis, Univ. of Illinois at Urbana-Champaign, 1992.

[49] C.M. Wittenbrink, "Survey of Parallel Volume Rendering Algorithms," *Proc. Int'l Conf. Parallel and Distributed Processing Techniques and Applications,* pp. 1329-1336, 1998.

Berkant Barla Cambazoglu graduated from Bursa Erkek Lisesi. He received the BS, MS, and PhD degrees, all in computer engineering, from the Computer Engineering Department of Bilkent University in 1997, 2000, and 2006, respectively. He has worked in two research projects funded by The Scientific and Technological Research Council of Turkey and a project funded by the European Union Sixth Framework Program. His research interests include parallel computing, scientific visualization, information retrieval, data mining, and grid computing.

Cevdet Aykanat received the BS and MS degrees from the Middle East Technical University, Ankara, Turkey, both in electrical engineering, and the PhD degree from The Ohio State University, Columbus, in electrical and computer engineering. He was a Fulbright scholar during his PhD studies. He worked at the Intel Supercomputer Systems Division, Beaverton, Oregon, as a research associate. Since 1989, he has been affiliated with the Department of Computer Engineering, Bilkent University, Ankara, Turkey, where he is currently a professor. His research interests mainly include parallel computing, parallel scientific computing and its combinatorial aspects, parallel computer graphics applications, parallel data mining, graph and hypergraph-partitioning, load balancing, neural network algorithms, high-performance information retrieval systems, parallel and distributed Web crawling, parallel and distributed databases, and grid computing. He has (co)authored about 40 technical papers published in academic journals indexed in SCI. He is the recipient of the 1995 Young Investigator Award of The Scientific and Technological Research Council of Turkey. He is a member of the ACM, the IEEE, and the IEEE Computer Society. He has been recently appointed as a member of IFIP Working Group 10.3 (Concurrent Systems) and INTAS Council of Scientists.