Available online at www.sciencedirect.com

ScienceDirect

ELSEVIER

J. Parallel Distrib. Comput. 67 (2006) 77–99

Journal of Parallel and Distributed Computing

www.elsevier.com/locate/jpdc

Adaptive decomposition and remapping algorithms for object-space-parallel direct volume rendering of unstructured grids[☆]

Cevdet Aykanat[a,*], B. Barla Cambazoglu[a], Ferit Findik[b], Tahsin Kurc[c]

[a]Computer Engineering Department, Bilkent University, TR 06800 Bilkent, Ankara, Turkey
[b]Microsoft Corporation, Redmond, WA 98052-6399, USA
[c]The Ohio State University, Biomedical Informatics Department, Columbus, OH 43210, USA

Received 4 December 2005; received in revised form 31 March 2006; accepted 6 May 2006
Available online 24 July 2006

Abstract

Object space (OS) parallelization of an efficient direct volume rendering algorithm for unstructured grids on distributed-memory architectures is investigated. The adaptive OS decomposition problem is modeled as a graph partitioning (GP) problem using an efficient and highly accurate estimation scheme for view-dependent node and edge weighting. In the proposed model, minimizing the cutsize corresponds to minimizing the parallelization overhead due to the data communication and redundant computation/storage while maintaining the GP balance constraint corresponds to maintaining the computational load balance in parallel rendering. A GP-based, view-independent cell clustering scheme is introduced to induce more tractable view-dependent computational graphs for successive visualizations. As another contribution, a graph-theoretical remapping model is proposed as a solution to the general remapping problem and is used in minimization of the cell-data migration overhead. The remapping tool RM-MeTiS is developed by modifying the GP tool MeTiS and is used in partitioning the remapping graphs. Experiments are conducted using benchmark datasets on a 28-node PC cluster to evaluate the performance of the proposed models.

Keywords: Direct volume rendering; Object space parallelization; Adaptive decomposition; Unstructured grids; Graph partitioning; Remapping

1. Introduction

The increasing complexity of scientific and engineering simulations necessitates more powerful tools for interpretation of the acquired results. *Scientific visualization* algorithms are utilized for detailed interpretation of the resulting datasets to reach useful conclusions. *Volume rendering* is a very important branch of scientific visualization, which makes it possible for scientists to visualize 3D volumetric datasets.

The data used in volume rendering is in the form of a grid superimposed on a volume. The nodes (data points) of the grid contain the scalar values that represent the simulation results. Volumetric grids can be classified as *structured* and *unstructured*. Structured grids are topologically equivalent to integer lattices and hence can easily be represented by a 3D array. In unstructured grids, distribution of the data points does not follow a regular pattern, and there may be voids in the grid. These grids are represented by a list of cells in which each cell contains pointers to its data points. The cell-to-cell connectivity information is provided explicitly. With advances in generating high-quality adaptive meshes, unstructured grids became popular in simulating the scientific and engineering problems that have complex geometries.

Volume rendering methods can be classified as *direct* and *indirect*. Direct methods differ from indirect methods in that they do not use any intermediate representation of the data. They are more general, flexible and have the potential to provide more complete information about the data being visualized. However, direct methods are slow due to massive computations, and visualizing the vast amounts of volumetric data produced by simulations requires large computer memory. Hence, direct volume rendering (DVR) is a good candidate for parallelization on distributed-memory architectures. In addition, most simulations

[☆] This work is partially supported by The Scientific and Technical Research Council of Turkey under Grant EEEAG-103E028.

[*] Corresponding author. Fax: +90 312 266 4047.
E-mail addresses: aykanat@cs.bilkent.edu.tr (C. Aykanat), berkant@cs.bilkent.edu.tr (B.B. Cambazoglu), feritf@microsoft.com (F. Findik), kurc-1@medctr.osu.edu (T. Kurc).

0743-7315/$ - see front matter © 2006 Elsevier Inc. All rights reserved.
doi:10.1016/j.jpdc.2006.05.005

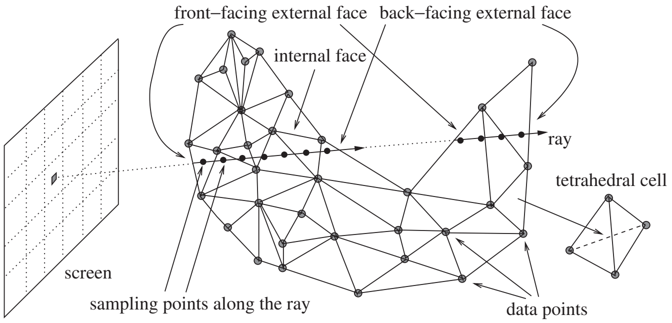
front–facing external face
back–facing external face
internal face
ray
tetrahedral cell
screen
sampling points along the ray
data points

Fig. 1. Ray-casting-based DVR of an unstructured dataset.

are conducted on general-purpose multicomputers. Visualizing the results on the parallel machine where the simulations are done avoids the overhead of transporting large amounts of data.

Research on DVR of unstructured grids started in the early 1990s [1,8,10,11,14,15,17,26,27,43,44,47,50,53]. DVR methods can be classified as *projection-based* and *ray-casting-based*. In projection-based methods, cells are projected onto the screen, in visibility order, to find their contributions and composite them. In ray-casting-based methods, for each pixel on the screen, a ray is cast and followed through the volume in front-to-back order. Samples are computed along the ray and are composited to generate the color of the pixel. For non-convex datasets, the rays may enter and exit the volume more than once. The parts of the rays that lie inside the volume, which in fact determine the contributions of the dataset to the screen, are called *ray segments*. Fig. 1 displays some concepts in ray-casting-based DVR. Ray casting is a desirable choice for DVR due to its capability of rendering non-convex and cyclic grids and its power of generating high-quality images. In this work, Koyamada's [26] algorithm, being one of the outstanding algorithms in ray casting, is selected for parallelization. Some basic concepts in DVR and an overview of Koyamada's DVR algorithm is provided in Appendix A.

1.1. Approaches and issues in parallel DVR

A DVR application contains two interacting domains: *object space* (OS) and *image space* (IS). The OS is a 3D domain (volume) containing the data to be visualized. The IS is a 2D domain (screen) containing the pixels from which rays are shot into the 3D object domain. Based on these domains, there are basically two approaches in parallel DVR: IS and OS parallelism. In IS parallelism, the screen is decomposed into regions, and each region is assigned to a separate processor. The cells are redistributed among the processors such that each processor has all the cells whose projection areas intersect the screen region assigned to it. Then, each processor performs local rendering operations to generate the complete image for its local screen region. In OS parallelism, the object domain is decomposed into disjoint subvolumes, and each subvolume is concurrently rendered by a separate processor. At the end of this local rendering phase, full screen but partial images are created at each processor. In the pixel merging phase, the partial images are composited over the interconnection network. OS parallelism is a promising approach offering excellent scalability in terms of the number of cells it can handle.

An important factor in decomposition is spatial coherency. There are two closely interrelated types of spatial coherency: *IS coherency* and *OS coherency*. IS coherency (pixel-to-pixel coherency) relies on the observation that the rays shot from nearby pixels of the screen are likely to pass through the same and/or nearby cells of the volume. OS coherency (cell-to-cell coherency) relies on the observation that nearby cells contribute to the same and/or nearby pixels. Disturbing the spatial coherency due to decomposition incurs parallelization *overheads* in the forms of communication, redundant computation, and redundant storage. For example, in OS decomposition, assigning the neighbor cells which contribute to the same pixel(s) to separate processors incurs communication in the pixel merging phase. Furthermore, the state-of-the-art DVR algorithms try to exploit the spatial coherency as much as possible to speed up rendering operations. Disturbing the spatial coherency utilized by the underlying sequential DVR algorithm may incur redundant computation. Hence, decomposition algorithms should try to preserve the spatial coherency as much as possible to minimize the total parallelization overhead.

Both IS- and OS-parallel approaches can be classified as *static*, *dynamic*, and *adaptive*. The static approach is a view-independent scheme, where the decomposition remains static for multiple visualizations of the same dataset for different viewing parameters. This approach has the advantage of avoiding the decomposition overhead for each visualization instance. But, it fails to maintain the load balance and spatial coherency together. In the dynamic approach, atomic rendering tasks are assigned to processors on a demand-driven basis. Although the dynamic approach solves the load balancing problem in a natural way, it suffers from disturbing the spatial coherency since neighbor pixels and/or cells may be processed by different processors. Furthermore, each task assignment incurs communication on distributed-memory architectures. Adaptive decomposition is a view-dependent scheme in which the decomposition and task assignment are adapted according to the changes in parameters of successive visualizations. The adaptive approach is promising because it explicitly handles both the load balancing and spatial coherency problems.

1.2. Previous work on parallel DVR

Several works exist on parallel DVR of unstructured grids on shared-memory architectures [5,6,48,49]. Challinger [5,6] presented IS parallelization of a hybrid DVR algorithm [7]. Wilhelms et al. [48] presented IS parallelization of a hierarchical DVR algorithm for irregular and multiple grids. Williams [49] presented OS parallelization of a projection-based DVR algorithm. The adaptive approach was investigated in IS parallelization for distributed-memory architectures [3,28,38]. Cambazoglu and Aykanat [3] developed an adaptive, IS-parallel DVR algorithm based on hypergraph partitioning. Kutluca et al. [28] presented a comparison of 12 adaptive IS decomposition algorithms. Palmer and Taylor [38] presented adaptive IS parallelization of a ray-casting-based DVR algorithm. Two recent works exist for distributed-shared-memory architectures [12,19]. Farias and Silva [12] presented parallelization of their ZSWEEP algorithm [10]. Hofsetz and Ma [19] presented multithreaded parallelization of a projection-based DVR algorithm. Several recent works exist on parallel polygon rendering [29,34,39,40].

OS parallelization on distributed-memory architectures was investigated in two works [31,32], where the static task assignment approach was adopted. In the former work [31], Ma presented parallelization of the ray-casting-based DVR algorithm of Garrity [14]. Ma formulated the static OS decomposition problem as a *graph partitioning* (GP) problem. In this model, nodes of the graph represent the cells, and edges represent the connectivity between the cells. Thus, minimizing the cutsize in partitioning is an effort towards preserving the OS coherency. Unit node and edge weighting is used so that the GP tool Chaco [18] generates subvolumes containing equal number of cells. But, due to the large cell-size variation in unstructured grids, having subvolumes with equal number of cells is not adequate to maintain the load balance. A similar situation holds for the unit edge weighting scheme due to the large face area variation in unstructured grids.

In the latter work [32], which is reelaborated in [33], Ma and Crockett adopted scattered cell assignment to achieve static load balancing in OS parallelization of a projection-based DVR algorithm. In the scattered cell assignment approach, assignment of neighbor cells to different processors results in a total loss of spatial coherency. This may cause a considerable performance degradation in local rendering computations if the underlying sequential DVR algorithm exploits the spatial coherency. For example, Koyamada's ray-casting-based DVR algorithm, which is the underlying DVR algorithm in our work, exploits OS coherency during the ray segment traversal through connectivity information. The DVR algorithm used in [32] exploits neither OS nor IS coherency, so its performance is not affected by the lack of spatial coherency in the local rendering phase. However, the ray segments generated for each local cell of each processor must be saved until the pixel merging phase, and each such ray segment incurs an extra volume of communication in the data migration phase. Ma and Crockett [32] proposed a depth ordering for local rendering and PM computations by partitioning the volume along orthogonal coordinate planes through the use of a hierarchical spatial data structure. This computational ordering through spatial partitioning restores some amount of spatial coherency, which is totally destroyed due to the scattered assignment, and also reduces the memory requirement due to the storage of the generated ray segments. In order to decrease the communication overhead in the PM phase, the local rendering and pixel merging phases are multiplexed and communication is overlapped with local rendering computations.

Wittenbrink [51] investigated parallelization of the projection-based DVR algorithm of Shirley and Tuchman [43] on PixelFlow [9], which is a special-purpose graphics architecture providing hardware for both IS and OS parallelism for polygon rendering. Special-purpose architectures are out of the scope of this paper. In this paper, we investigate OS parallelization for general-purpose distributed-memory architectures assuming that the visualization process is performed on the parallel machine where the simulations are done. A survey of parallel volume rendering algorithms can be found in [52].

1.3. Contributions

In this work, we propose a novel GP-based approach, which consists of a view-independent and a view-dependent step, for adaptive OS decomposition. The objective of the view-independent step is to minimize the view-dependent decomposition overhead to make adaptive decomposition affordable. For the view-independent step, we propose an efficient top-down cell clustering scheme, formulated as a GP problem, to generate the atomic tasks, i.e., clusters of connected cells. We approximate the average-case computational structure for multiple visualizations of a dataset $\mathcal{V}$ by a computational graph $\mathcal{G}_{\mathcal{V}}$. The nodes and edges of $\mathcal{G}_{\mathcal{V}}$, which represent the cell-to-cell connectivity information of a given volumetric dataset $\mathcal{V}$, are weighted with the volumes and areas of the respective cells and faces in $\mathcal{V}$. Partitioning this weighted graph induces cell clusters of roughly equal volume while minimizing the sum of the boundary areas between cluster pairs.

The view-dependent OS decomposition problem is modeled as K-way partitioning of the coarse computational graph obtained by contracting $\mathcal{G}_{\mathcal{V}}$ according to the clustering solution found in the view-independent step. A highly accurate and efficient estimation scheme is proposed for view-dependent node and edge weighting of the coarse computational graph, where the weight of a node accounts for the computational load of the respective cell cluster and the weight of an edge accounts for the number of common rays intersecting the respective pair of cell clusters. In this model, maintaining the balance among part sizes corresponds to maintaining the computational load balance in the local rendering phase, and minimizing the cutsize corresponds to minimizing the total number of local ray segments to be generated due to the virtual non-convexities introduced by the decomposition. Thus, minimizing the cutsize corresponds to minimizing the total overhead due to the redundant computation and storage during the local rendering phase and the total communication in the pixel merging phase. The consistency of this correspondence can be maintained by an efficient implementation of the local rendering phase, which considers the set of local cell clusters of each processor as a single subvolume.

In adaptive OS decomposition, each new decomposition and mapping may incur an excessive amount of cluster data migration due to the differences in the new and previous mappings of cell clusters. Therefore, an adaptive decomposition model should also consider minimization of this data redistribution overhead as well as minimization of the communication volume and the amount of redundant computation to incur because of assigning interacting clusters to different processors. This problem constitutes a typical case of a general problem known as the *remapping* problem. In this work, we also propose a novel

Table 1
The summary of important abbreviations and symbols

bf face/ff face	Back-facing/front-facing face
NPCS	The normalized projection coordinate system: $\mathcal{V}$ transformed into the IS
p-node/t-node	Processor/task node in a remapping graph
pt-edge/tt-edge	Processor-to-task/task-to-task edge in a remapping graph
WCS	The world coordinate system: $\mathcal{V}$ defined in the OS
$\mathcal{C}=\{\mathcal{C}_1,\ldots,\mathcal{C}_\alpha\}$	An α-way clustering of the cells of $\mathcal{V}$
$\mathcal{C}_i$	The subvolume of $\mathcal{V}$ determined by the set of cells in the ith cluster of $\mathcal{C}$
$\tilde{\mathcal{E}}_{\text{pt}}/\tilde{\mathcal{E}}_{\text{tt}}$	The set of pt-edges/tt-edges in a remapping graph
γ^v	The volume of a unit cube of the WCS in the NPCS for a given v
$\mathcal{G}=(\mathcal{N},\mathcal{E},w)$	The computational graph
$(\mathcal{G},\mathcal{M},\mathcal{T})$	The remapping 3-tuple
$\tilde{\mathcal{G}}=(\tilde{\mathcal{N}},\tilde{\mathcal{E}},\tilde{w})$	The remapping graph
$\tilde{\mathcal{G}}_\ell$	The ℓth level coarser remapping graph obtained by the coarsening phase of RM-MeTiS
$\tilde{\mathcal{G}}_m$	The coarsest remapping graph obtained by the coarsening phase of RM-MeTiS
$\mathcal{G}_\mathcal{V}=(\mathcal{N}_\mathcal{V},\mathcal{E}_\mathcal{V})$	The graph representing the cell-to-cell connectivity information of $\mathcal{V}$
$\mathcal{G}_{\mathcal{V}_k}$	The graph representing the cell-to-cell connectivity information of a subvolume $\mathcal{V}_k$ of $\mathcal{V}$
$\mathcal{G}_\mathcal{V}^v=(\mathcal{N}_\mathcal{V},\mathcal{E}_\mathcal{V},w_\mathcal{V}^v)$	The fine visualization graph representing the computational structure of rendering $(\mathcal{V},v)$
$\mathcal{G}_\mathcal{V}^{\text{avg}}=(\mathcal{N}_\mathcal{V},\mathcal{E}_\mathcal{V},w_\mathcal{V}^{\text{avg}})$	The clustering graph representing an average-case computational structure for $\mathcal{V}$
$\mathcal{G}_\mathcal{C}=(\mathcal{N}_\mathcal{C},\mathcal{E}_\mathcal{C})$	The coarse graph representing the cluster-to-cluster connectivity information of $\mathcal{V}$
$\mathcal{G}_\mathcal{C}^v=(\mathcal{N}_\mathcal{C},\mathcal{E}_\mathcal{C},w_\mathcal{C}^v)$	The coarse visualization graph representing the computational structure of rendering $(\mathcal{V},v)$ according to cell clustering $\mathcal{C}$
$(\mathcal{G}_\mathcal{C}^v,\mathcal{M}_\mathcal{C},\mathcal{T}_\mathcal{C})$	The remapping 3-tuple for adaptive OS decomposition
$\tilde{\mathcal{G}}_\mathcal{C}^v=(\tilde{\mathcal{N}}_\mathcal{C},\tilde{\mathcal{E}}_\mathcal{C},\tilde{w}_\mathcal{C}^v)$	The coarse remapping graph for adaptive OS decomposition
I_i^v	The number of ray–face intersections performed in subvolume $\mathcal{C}_i$ of $\mathcal{V}$ for a given v
K	The number of processors
$\mathcal{M}/\mathcal{M}_\mathcal{C}$	The current task-to-processor/cluster-to-processor mapping functions
$\tilde{\mathcal{M}}/\tilde{\mathcal{M}}_\mathcal{C}$	The task-to-processor/cluster-to-processor mapping functions obtained after remapping
$\tilde{\mathcal{N}}_p/\tilde{\mathcal{N}}_t$	The set of p-nodes/t-nodes in a remapping graph
$\Pi=\{\mathcal{P}_1,\ldots,\mathcal{P}_K\}$	A K-way partition of a graph $\mathcal{G}/\mathcal{G}_\mathcal{V}$
$\tilde{\Pi}=\{\tilde{\mathcal{P}}_1,\ldots,\tilde{\mathcal{P}}_K\}$	A K-way partition of a remapping graph $\tilde{\mathcal{G}}/\tilde{\mathcal{G}}_\mathcal{C}^v$ satisfying the mapping constraint (Eq. (3))
$\tilde{\Pi}_\ell$	A K-way partition of the ℓth level coarser remapping graph $\tilde{\mathcal{G}}_\ell$ obtained during the uncoarsening phase of RM-MeTiS
$\mathcal{P}_k$	The set of nodes in the kth part of Π (it determines a subvolume $\mathcal{V}_k$ of $\mathcal{V}$)
$\tilde{\mathcal{P}}_k$	The kth part of $\tilde{\Pi}$ containing a single p-node and a set of t-nodes
R_i^v	The number of ray segments generated for subvolume $\mathcal{C}_i$ of $\mathcal{V}$ for a given v
S_i^v	The number of samples taken in subvolume $\mathcal{C}_i$ of $\mathcal{V}$ for a given v
$t_I/t_R/t_S$	The time cost of a ray–face intersection/ray-segment generation/sampling operation
$\mathcal{T}/\mathcal{T}_\mathcal{C}$	The task/cluster migration cost function
T_i^v	The sequential rendering time of a subvolume $\mathcal{C}_i$ of $\mathcal{V}$ for a given v
$(\mathcal{V},v)$	A visualization instance: rendering a volumetric dataset $\mathcal{V}$ for a given viewing parameter set v
$1/\Delta z$	The sampling rate in equidistant sampling
(z,s)	The depth (z) and scalar (s) values computed at a ray–face intersection point

GP-based model as a solution to the general remapping problem. The proposed model generates a remapping graph, augmenting the computational graph by adding processor nodes and processor-to-task edges. In the literature, the idea of making such augmentations appears in a different manner in the context of the task assignment problem in heterogeneous distributed systems [25,30,45]. In this work, a remapping tool, herein referred to as ReMapping-MeTiS (RM-MeTiS), is also developed for partitioning the remapping graph by modifying the GP tool MeTiS [23] and is used in adaptive OS decomposition.

The remapping problem has received close attention in the literature. In general, scratch-remap [36,42] and diffusion-based [37,41,42,46] approaches are followed as GP-based solutions to this problem. Oliker and Biswas [36] proposed a scheme in which the workload graph is repartitioned from scratch using MeTiS, and the parts are assigned to processors in a separate step to minimize the data movement. Two improvements for the scratch-remap scheme are proposed by Schloegel et al. [42]. Ou and Ranka [37] developed a method which optimally minimizes the one-norm of the diffusion solution using linear programming. Walshaw et al. [46] proposed an iterative optimization technique which incrementally modifies the existing mapping to construct a new mapping. Schloegel et al. [41] performed diffusion of tasks in a multilevel framework with two algorithms that try to minimize data movement without significantly compromising the cutsize. In [42], the same authors proposed improvements over their diffusion-based remapping algorithms.

The rest of the paper is organized as follows. Table 1 displays some abbreviations and the notation used in the paper. The proposed adaptive OS decomposition approach is discussed in Section 2. Section 3 presents the proposed remapping model. Our remapping tool RM-MeTiS is presented in Section 4. The

OS-parallel DVR algorithm is discussed in Section 5. Section 6 gives the experimental results obtained on a 28-node PC cluster. Finally, conclusions are presented in Section 7.

2. Creating view-dependent coarse-grain visualization graph

2.1. Fine-grain decomposition model

The cell-to-cell connectivity information of a volumetric dataset $\mathcal{V}$ is represented as an undirected graph $\mathcal{G}_{\mathcal{V}} = (\mathcal{N}_{\mathcal{V}}, \mathcal{E}_{\mathcal{V}})$. The nodes of $\mathcal{G}_{\mathcal{V}}$ represent the cells of $\mathcal{V}$, and there exists an edge between two nodes if the respective cells share a face. Only internal faces incur edges in $\mathcal{G}_{\mathcal{V}}$. In a partition of a graph, an edge is said to be *cut* if its nodes are in different parts, and *uncut* otherwise. Consider a cut edge $e_{ij} \in \mathcal{E}_{\mathcal{V}}$ corresponding to internal face f_{ij} shared between the pair of cells c_i and c_j which are mapped to separate processors. In the tetrahedral cell model, the set of rays intersecting internal face f_{ij} determines the set of common rays intersecting both c_i and c_j. Without loss of generality, assume that c_j is behind c_i, in front-to-back visibility ordering, for a given viewing parameter set v. This also denotes that f_{ij} is a back-facing (bf) face of c_i, whereas it is a front-facing (ff) face of c_j. Since the composition operation is *associative*, the processor holding c_j can generate ray segments for the rays intersecting ff face f_{ij} of c_j and initiate the traversal and composition of these ray segments without waiting for the composition results of the respective rays from the processor holding c_i. However, these partial composition results should be merged according to the visibility order. Hence, each cut edge e_{ij} in a partition of $\mathcal{G}_{\mathcal{V}}$ incurs communication because of the merging operations, and the volume of communication is proportional to the number of rays intersecting f_{ij}. Each cut edge e_{ij} also disturbs the OS coherency utilized by Koyamada's sequential algorithm. In this algorithm, for any ray intersecting face f_{ij}, the exit-point (z, s) values computed for cell c_i are directly used as the entry-point (z, s) values for cell c_j for sampling and composition operations. So, cut edge e_{ij} incurs the redundant computation of the (z, s) values for c_j from scratch for each ray intersecting f_{ij} during the ray-segment generation. Furthermore, cut edge e_{ij} incurs the redundant storage of the ray segments generated and composited for ff face f_{ij} of c_j in the processor holding c_j. The amounts of both types of redundancies are proportional to the number of rays intersecting f_{ij}.

The computational structure of rendering a visualization instance $(\mathcal{V}, v)$ can be represented by a weighted graph $\mathcal{G}^v_{\mathcal{V}} = (\mathcal{N}_{\mathcal{V}}, \mathcal{E}_{\mathcal{V}}, w^v_{\mathcal{V}})$, herein referred to as the *visualization graph*. $w^v_{\mathcal{V}}$ denotes the weighting function to be defined on the nodes and edges of $\mathcal{G}_{\mathcal{V}}$ to create $\mathcal{G}^v_{\mathcal{V}}$ for a given v. Each node of $\mathcal{G}^v_{\mathcal{V}}$ should be assigned a weight proportional to the rendering computations associated with the respective cell. Each edge of $\mathcal{G}^v_{\mathcal{V}}$ should be assigned a weight proportional to the number of rays intersecting the respective internal face. Thus, an OS decomposition of a visualization instance $(\mathcal{V}, v)$ can be obtained by K-way partitioning of visualization graph $\mathcal{G}^v_{\mathcal{V}}$. Each part $\mathcal{P}_k$ in a K-way partition $\Pi = \{\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_K\}$ of $\mathcal{G}^v_{\mathcal{V}}$ corresponds to a subvolume $\mathcal{V}_k$ to be rendered simultaneously and independently by a distinct processor P_k, for $k = 1, 2, \ldots, K$. Maintaining the balance constraint in GP corresponds to maintaining the computational load balance among the processors during the local rendering calculations. The cutsize of a partition Π is exactly equal to the increase (ΔR) in the number of ray segments generated, processed, and stored because of the OS decomposition. That is, the cutsize is equal to $\Delta R = R' - R$, where R' and R denote the total number of ray segments generated by the parallel and sequential DVR algorithms, respectively. Hence, the cutsize plus R determines the worst-case communication volume to incur during the pixel merging phase. The worst case occurs if the ray segment(s) generated for each active pixel are merged by a processor different than the processor(s) which generated those ray segments. Since R is a constant value for a given visualization instance, minimizing the cutsize corresponds to minimizing both the upper bound on the total volume of communication and the total amount of redundant computation and storage.

We identify two types of computational interactions between the cells: *internal* and *external* interactions. An internal interaction occurs between two neighbor cells if they share an internal face. An external interaction occurs between two external cells if the projection areas of their ff external faces overlap. If an internal interaction exists between two cells, then this interaction exists throughout the visualization, but its magnitude is subject to change with the varying viewing parameters. However, in the case of external interactions, both interactions and their magnitudes are subject to change with the varying viewing parameters. In the proposed model, only the internal interactions induce edges in $\mathcal{G}^v_{\mathcal{V}}$, where the edge weights represent the magnitudes of the interactions between the respective cells for the given v. Since external interactions exist only in non-convex meshes, the proposed model provides a full coverage for cell-to-cell interactions. In meshes having a high rate of convexity, the total amount of external interactions is much less than that of internal interactions. So, omission of external interactions in the model is not expected to degrade the quality of decompositions in such datasets.

2.2. View-independent cell clustering for coarsening

OS decomposition is performed at the beginning of each visualization instance. So, this overhead must be minimized as much as possible to make it affordable. We apply a view-independent clustering on cells of $\mathcal{V}$ to induce more tractable visualization graph instances for the following view-dependent OS decompositions. That is, instead of individual cells, the cell clusters constitute the atomic tasks for the rendering computations. $\mathcal{V}$ is decomposed into a set $\mathcal{C} = \{\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_\alpha\}$ of α disjoint cell clusters. The total number of clusters should be both considerably smaller than the number of cells and sufficiently larger than K to enable a large search space for the GP tool during the view-dependent OS decomposition.

A view-independent graph $\mathcal{G}^{\text{avg}}_{\mathcal{V}} = (\mathcal{N}_{\mathcal{V}}, \mathcal{E}_{\mathcal{V}}, w^{\text{avg}}_{\mathcal{V}})$ is constructed for top-down clustering through GP. Here, $w^{\text{avg}}_{\mathcal{V}}$

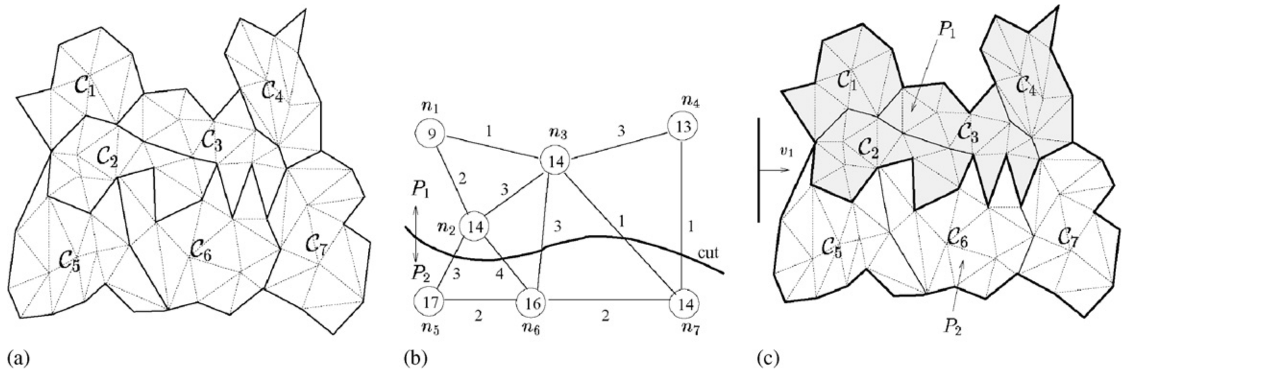

Fig. 2. (a) A 7-way clustering $\mathcal{C} = \{\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_7\}$ of a sample dataset $\mathcal{V}$ containing 97 cells. (b) A coarse visualization graph $\mathcal{G}_\mathcal{C}$ obtained by contracting $\mathcal{G}_\mathcal{V}$ according to clustering $\mathcal{C}$ and a 2-way partition $\Pi = \{\mathcal{P}_1 = \{\mathcal{C}_1, \mathcal{C}_2, \mathcal{C}_3, \mathcal{C}_4\}, \mathcal{P}_2 = \{\mathcal{C}_5, \mathcal{C}_6, \mathcal{C}_7\}\}$ of $\mathcal{G}_\mathcal{C}$ for v_1. Numbers inside the circles show the cell counts that the respective clusters have. Numbers beside the edges show the number of pseudo-boundary faces shared by the respective pairs of cell clusters. (c) The decomposition and mapping of $\mathcal{V}$ according to Π.

denotes the weighting function defined on the nodes and edges of $\mathcal{G}_\mathcal{V}$ for estimating an average-case computational structure for multiple visualizations of $\mathcal{V}$. Each node of $\mathcal{G}_\mathcal{V}^{\text{avg}}$ is assigned a weight proportional to the volume of the respective cell. The rationale is that the number of samples to be taken in a cell is proportional to its volume. Furthermore, more rays are likely to hit a cell with a large volume. Thus, the volume of a cell in the world coordinate system (WCS) approximates a view-independent load for the cell, relative to other cells. Similarly, more rays are likely to intersect a face with a large area. So, the area of a face approximates the amount of interaction between the two cells sharing that face. Hence, each edge of $\mathcal{G}_\mathcal{V}^{\text{avg}}$ is weighted with the area (in the WCS) of the respective face. The α-way partitioning of $\mathcal{G}_\mathcal{V}^{\text{avg}}$ serves our purpose for view-independent α-way cell clustering by inducing cell clusters of approximately equal volume while minimizing the sum of the boundary areas between neighbor clusters.

Fig. 2(a) illustrates a 7-way clustering of a sample dataset $\mathcal{V}$. In Fig. 2(a), due to the 2D representation, each triangle represents a tetrahedral cell, and each triangle edge represents the respective face of the cell. In a cluster, a face is called a *pseudo-boundary* face if it is shared by two cells belonging to two different clusters. In Fig. 2(a), the dotted lines represent internal faces of a cluster, and the solid lines determine cluster boundaries. Each solid line shared by two triangles represents a pseudo-boundary face.

Graph $\mathcal{G}_\mathcal{V}$ representing $\mathcal{V}$ is contracted according to clustering $\mathcal{C}$ to obtain a coarser graph representation $\mathcal{G}_\mathcal{C} = (\mathcal{N}_\mathcal{C}, \mathcal{E}_\mathcal{C})$ for $\mathcal{V}$. In $\mathcal{G}_\mathcal{C}$, $\mathcal{N}_\mathcal{C} = \mathcal{C}$, and there exists an edge e_{ij} between the nodes representing clusters $\mathcal{C}_i$ and $\mathcal{C}_j$ if and only if these clusters share at least one pseudo-boundary face. Graph $\mathcal{G}_\mathcal{C}$ represents the cluster-to-cluster connectivity information of $\mathcal{V}$ for the given clustering $\mathcal{C}$. Fig. 2(b) illustrates the contraction of $\mathcal{G}_\mathcal{V}$ according to the 7-way clustering $\mathcal{C}$ given in Fig. 2(a) to obtain $\mathcal{G}_\mathcal{C}$ with seven nodes and 11 edges.

A pseudo-boundary face may become a *boundary* face after a decomposition if it is shared by two cells belonging to two different clusters which are mapped to different processors. In Fig. 2(b), the cut edges (n_2, n_5), (n_2, n_6), (n_3, n_6), (n_3, n_7), and (n_4, n_7) respectively, represent 3, 4, 3, 1, and 1 pseudo-boundary faces, which become boundary faces after the decomposition. These boundary faces represented by the cut edges incur redundant computation and storage during the local rendering phase and communication during the pixel merging phase. Fig. 2(c) displays the 2-way decomposition and mapping of $\mathcal{V}$ induced by bipartition Π given in Fig. 2(b). In Fig. 2(c), the shaded and unshaded cell clusters show the sets of clusters $\mathcal{P}_1$ and $\mathcal{P}_2$ assigned to processors P_1 and P_2, respectively. The bold lines determine the boundaries of the local subvolumes formed by local cell clusters, and each bold line segment represents a boundary face.

2.3. View-dependent node and edge weighting

The computational structure of each successive visualization instance $(\mathcal{V}, v)$ is represented by a coarse visualization graph $\mathcal{G}_\mathcal{C}^v = (\mathcal{N}_\mathcal{C}, \mathcal{E}_\mathcal{C}, w_\mathcal{C}^v)$, where $w_\mathcal{C}^v$ denotes the weighting function to be computed for the nodes and edges of $\mathcal{G}_\mathcal{C}$ to construct $\mathcal{G}_\mathcal{C}^v$. Clearly, the topology of $\mathcal{G}_\mathcal{C}^v$ does not change with changing v. To simplify the discussion, we assume that the entire volume is visible on the screen. Extensions to the proposed weight estimation schemes when the volume is partially visible are presented in Appendix B.

2.3.1. Node weight estimation

The rendering time T_i^v of a cell cluster $\mathcal{C}_i$ of $\mathcal{V}$ for a given v can be dissected into three major components: *ray-segment generation*, *ray–face intersection*, and *sampling* times. Ray-segment generation involves scan converting ff external and ff pseudo-boundary faces of $\mathcal{C}_i$ to compute the intersections of ray segments with these faces and the (z, s) values at the intersection points. Ray–face intersection involves finding the intersections of ray segments with the bf faces of $\mathcal{C}_i$ and computing the scalar values and gradients at the intersections. Sampling involves computing the scalar values at sampling points inside $\mathcal{C}_i$, mapping the scalar values to colors and opacities, and compositing these. Hence,

$$w(n_i) = T_i^v = R_i^v t_R + I_i^v t_I + S_i^v t_S, \tag{1}$$

where R_i^v, I_i^v, and S_i^v denote the numbers of ray segments generated for $\mathcal{C}_i$, ray–face intersections performed, and samples taken in $\mathcal{C}_i$, respectively. In Eq. (1), t_R, t_I, and t_S represent the unit costs of respective computations. These unit costs cannot be determined by measurement due to the highly interleaved execution manner of the respective types of computations. Hence, we estimated these unit costs statistically using the *least-squares approximation* method. Our experimental analysis in Section 6.2 shows that the average error in estimating the rendering time of a subvolume/volume using Eq. (1) with correct R^v, I^v, and S^v counts is below 4%. Hence, the computational load estimation to determine the weight $w(n_i)$ of a node n_i of $\mathcal{G}_C^v$ reduces to estimation of the I_i^v, S_i^v, and R_i^v counts associated with the rendering of $\mathcal{C}_i$ for a given v.

I_i^v can be calculated by summing the number of pixels covered by the projection areas of bf faces in $\mathcal{C}_i$. But, this exact computation scheme requires scan conversion of all bf faces and is a costly operation. Hence, we approximate the pixel coverage count of a face *f* by its projection area a_f in the normalized projection coordinate system (NPCS) as

$$a_f = x_1(y_2 - y_3) + x_2(y_3 - y_1) + x_3(y_1 - y_2), \qquad (2)$$

where x_i and y_i denote the *x* and *y* coordinates (in the NPCS) of the *i*th node of face *f*, respectively. The per face errors due to the discretization of the projection screen are not reflected to the total estimate for I_i^v due to the summation of floating-point area values of faces with contiguous projection areas.

In mid-point sampling, S_i^v is simply equal to I_i^v since each ray–face intersection incurs a single sampling. In equidistant sampling, S_i^v can be estimated by multiplying the volume of a cluster in the NPCS by the fixed sampling rate $1/\Delta z$. Because, the volume of the cluster in the NPCS denotes the number of unit cubes in that volume, and the number of samples to be taken in each unit cube in the NPCS is equal to $1/\Delta z$. As the NPCS is dependent on v, a straightforward implementation requires the computation and summation of the volumes (in the NPCS) of the individual cells of the cluster for each visualization instance. However, we adopt a much more efficient scheme which exploits the idea that there exists a constant scaling between any volume in the view-independent WCS and the view-dependent NPCS for a given v in parallel projection. This scale factor γ^v can be easily determined by computing the volume of a unit cube of the WCS in the NPCS for a given v. The total volume of each cluster in the WCS is computed only once so that estimation of S_i^v for a cluster $\mathcal{C}_i$ reduces to multiplying its volume by $\gamma^v/\Delta z$. The estimation errors due to the discretization of the sampling volume are just at boundary surfaces of clusters.

R_i^v can be approximated by the sum of the projection areas of ff external and ff pseudo-boundary faces of $\mathcal{C}_i$ using Eq. (2). However, as it will be described in Section 5.3, our local rendering scheme considers the whole set of local clusters of a processor as a single subvolume during the rendering, so ray segments are generated only for the ff external and ff boundary faces of the subvolume. As the ray segment generation cost of a cluster depends on the partitioning of $\mathcal{G}_C^v$, incorporation of this cost to an existing GP tool is quite difficult. Furthermore, partitioning of $\mathcal{G}_C^v$ involves minimization of the number of ray-segment generations due to boundary faces. Hence, the ray-segment generation cost is ignored in node weighting of $\mathcal{G}_C^v$.

2.3.2. Edge weight estimation

The weight $w(n_i, n_j)$ of an edge e_{ij}, which indicates the amount of interaction between clusters $\mathcal{C}_i$ and $\mathcal{C}_j$, is computed as follows. Each face shared by two neighbor cells in clusters $\mathcal{C}_i$ and $\mathcal{C}_j$ contributes its pixel coverage count to edge weight $w(n_i, n_j)$. The pixel coverage counts of these pseudo-boundary faces between $\mathcal{C}_i$ and $\mathcal{C}_j$ are approximated by the projection areas of these faces, which are computed using Eq. (2). The source of errors in this approximation is due to the discretization of the projection screen similar to that of ray–face intersection estimation. If clusters $\mathcal{C}_i$ and $\mathcal{C}_j$ are mapped to different processors, then these pseudo-boundary faces become boundary faces of the local subvolumes of the respective processors, thus incurring computation and storage overheads due to ray-segment generation and communication overhead due to pixel merging. The magnitudes of these overheads are proportional to edge weight $w(n_i, n_j)$.

3. Remapping model for adaptive decomposition

3.1. General remapping model

In the remapping problem, the computational structure of an application changes from one instance of the computation to another. The task mapping should be adapted in accordance with the changes in the computational structure, and this necessitates migration of computational tasks together with their associated data structures. The objective in each remapping step is to preserve the load balance while minimizing the total overhead due to both task migration and the interactions between the tasks mapped to different processors. A remapping problem instance is defined by a 3-tuple $(\mathcal{G}, \mathcal{M}, \mathcal{T})$. Here, $\mathcal{G} = (\mathcal{N}, \mathcal{E}, w)$ denotes the computational graph [4] representing the modified computational structure of the application. The nodes of this graph represent atomic computations. The weight $w(n_i)$ of node n_i denotes the computational load of the respective task. The weight $w(n_i, n_j)$ of edge e_{ij} represents the amount of communication and redundant computation to incur when the tasks represented by n_i and n_j are mapped to different processors. $\mathcal{M}$ denotes the current *K*-way task-to-processor mapping function, where $\mathcal{M}(n_i) = k$ means that the respective task currently resides in processor P_k. $\mathcal{T}$ denotes the task migration cost function, where $\mathcal{T}(n_i)$ is the cost of migrating the respective task from its current processor $P_{\mathcal{M}(n_i)}$ to another processor due to remapping.

In the proposed model, we construct a *remapping graph* $\tilde{\mathcal{G}} = (\tilde{\mathcal{N}}, \tilde{\mathcal{E}}, \tilde{w})$ by augmenting computational graph $\mathcal{G}$, adding a node p_k for each processor P_k. That is, $\tilde{\mathcal{N}} = \tilde{\mathcal{N}}_p \cup \tilde{\mathcal{N}}_t$, where $\tilde{\mathcal{N}}_p = \{p_1, p_2, \ldots, p_K\}$ and $\tilde{\mathcal{N}}_t = \mathcal{N}$. The added and original nodes are called as *processor-nodes* (p-nodes) and *task-nodes* (t-nodes), respectively. To obtain the edge set $\tilde{\mathcal{E}}$ of $\tilde{\mathcal{G}}$, we augment edge set $\mathcal{E}$ by connecting each p-node to the t-nodes

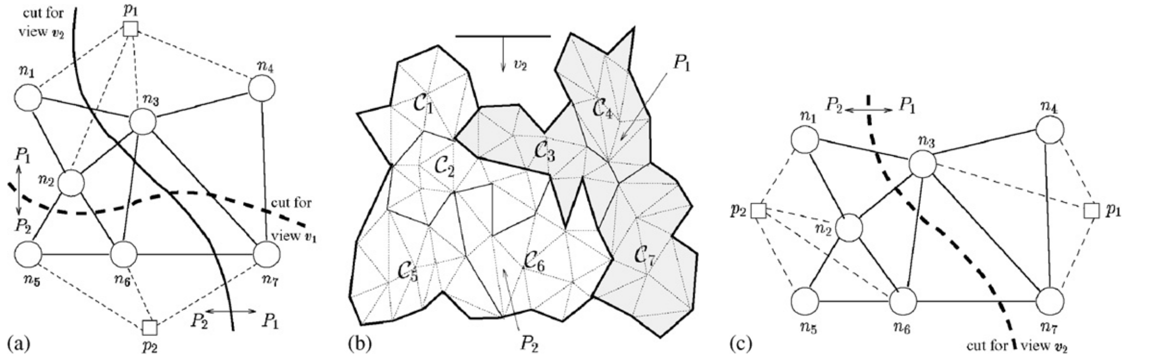

Fig. 3. (a) The remapping graph $\tilde{\mathcal{G}}_{\mathcal{C}}^{v}$ constructed after rendering visualization instance $(\mathcal{V}, v_1)$ in Fig. 2(c). The circles and squares represent the t-nodes and p-nodes, respectively. The dashed bold curve shows the cut, which represents bipartition $\Pi = \{\{p_1, n_1, n_2, n_3, n_4\}, \{p_2, n_5, n_6, n_7\}\}$ for $(\mathcal{V}, v_1)$. The solid bold curve shows the cut, which represents repartitioning $\tilde{\Pi} = \{\{p_1, n_3, n_4, n_7\}, \{p_2, n_1, n_2, n_5, n_6\}\}$ of $\tilde{\mathcal{G}}_{\mathcal{C}}^{v}$ for $(\mathcal{V}, v_2)$. (b) The 2-way decomposition and mapping of $\mathcal{V}$ induced by $\tilde{\Pi}$ for rendering of $(\mathcal{V}, v_2)$. (c) The remapping graph $\tilde{\mathcal{G}}_{\mathcal{C}}^{v}$ to be constructed after rendering $(\mathcal{V}, v_2)$ for the next visualization instance $(\mathcal{V}, v_3)$.

that correspond to the tasks residing in the respective processor according to the current mapping $\mathcal{M}$. That is, $\tilde{\mathcal{E}} = \tilde{\mathcal{E}}_{\text{pt}} \cup \tilde{\mathcal{E}}_{\text{tt}}$, where $\tilde{\mathcal{E}}_{\text{pt}} = \{(n_i, p_k) : n_i \in \tilde{\mathcal{N}}_t, p_k \in \tilde{\mathcal{N}}_p, \mathcal{M}(n_i) = k\}$ and $\tilde{\mathcal{E}}_{\text{tt}} = \mathcal{E}$. The added and original edges are called *processor-to-task* edges (pt-edges) and *task-to-task* edges (tt-edges), respectively. In node weighting, t-node weights remain the same, and p-nodes are assigned zero weights. In edge weighting, tt-edge weights remain the same, and each pt-edge is assigned the task migration cost of the respective task.

A K-way partition $\tilde{\Pi} = \{\tilde{\mathcal{P}}_1, \tilde{\mathcal{P}}_2, \ldots, \tilde{\mathcal{P}}_K\}$ of $\tilde{\mathcal{G}}$ is said to be *feasible* if it satisfies the *mapping constraint*,

$$|\tilde{\mathcal{P}}_k \cap \tilde{\mathcal{N}}_p| = 1 \quad \text{for } k = 1, 2, \ldots, K, \tag{3}$$

i.e., each part $\tilde{\mathcal{P}}_k$ of $\tilde{\Pi}$ contains exactly one p-node. Then, a feasible partition $\tilde{\Pi}$ of $\tilde{\mathcal{G}}$ induces remapping $\tilde{\mathcal{M}}$ in which tasks (t-nodes) in each part are all assigned to the same processor corresponding to the unique p-node in that part. That is, $\tilde{\mathcal{M}}(n_i) = k$ if both t-node n_i and p-node p_k are in the same part of $\tilde{\Pi}$.

Consequently, the remapping problem can be formulated as finding a K-way partition of $\tilde{\mathcal{G}}$ satisfying the mapping constraint (Eq. (3)). Maintaining the GP balance corresponds to maintaining the computational load balance. Minimizing the cutsize corresponds to minimizing the total overhead due to the task migration and the interactions between the tasks mapped to different processors. A cut tt-edge e_{ij} incurs communication between processors $P_{\tilde{\mathcal{M}}(n_i)}$ and $P_{\tilde{\mathcal{M}}(n_j)}$ due to the interactions between the tasks corresponding to t-nodes n_i and n_j. The cut tt-edges may also incur redundant computation. A cut pt-edge e_{ik} incurs communication due to the migration of the task corresponding to t-node n_i from processor P_k to $P_{\tilde{\mathcal{M}}(n_i)}$. The task migration cost function $\mathcal{T}$ must have an appropriate scaling between the weights of tt-edges and pt-edges.

3.2. Remapping model in OS-parallel DVR

Each visualization instance $(\mathcal{V}, v)$ is identified by a 3-tuple $(\mathcal{G}_{\mathcal{C}}^{v}, \mathcal{M}_{\mathcal{C}}, \mathcal{T}_{\mathcal{C}})$. $\mathcal{G}_{\mathcal{C}}^{v}$ represents the computational structure of $(\mathcal{V}, v)$. $\mathcal{M}_{\mathcal{C}}$ is the current cluster-to-processor mapping. $\mathcal{T}_{\mathcal{C}}$ is constructed through a scaling which considers only the communication volume overhead. That is, $\mathcal{T}_{\mathcal{C}}(n_i)$ is taken to be equal to the ratio of the data size of cluster $\mathcal{C}_i$ to the data size of a single ray segment. In the remapping graph $\tilde{\mathcal{G}}_{\mathcal{C}}^{v}$, tt-edges represent the interactions between clusters as in $\mathcal{G}_{\mathcal{C}}^{v}$, and pt-edges represent the current processor mapping of clusters. $\mathcal{T}_{\mathcal{C}}$ determines the weights of pt-edges. So, K-way partitioning of $\tilde{\mathcal{G}}_{\mathcal{C}}^{v}$ according to the mapping constraint (Eq. (3)) induces a remapping for the clusters of $\mathcal{V}$ with the desired properties for the new v.

Fig. 3(a) illustrates the remapping graph $\tilde{\mathcal{G}}_{\mathcal{C}}^{v}$ for the case in Fig. 2. Partition Π determines the current mapping $\mathcal{M}_{\mathcal{C}}(\mathcal{C}_1) = \mathcal{M}_{\mathcal{C}}(\mathcal{C}_2) = \mathcal{M}_{\mathcal{C}}(\mathcal{C}_3) = \mathcal{M}_{\mathcal{C}}(\mathcal{C}_4) = 1$ and $\mathcal{M}_{\mathcal{C}}(\mathcal{C}_5) = \mathcal{M}_{\mathcal{C}}(\mathcal{C}_6) = \mathcal{M}_{\mathcal{C}}(\mathcal{C}_7) = 2$ for $(\mathcal{V}, v_1)$. There are no cut pt-edges in Π as expected. $\tilde{\Pi}$ in Fig. 3(a) determines the remapping $\tilde{\mathcal{M}}_{\mathcal{C}}(\mathcal{C}_3) = \tilde{\mathcal{M}}_{\mathcal{C}}(\mathcal{C}_4) = \tilde{\mathcal{M}}_{\mathcal{C}}(\mathcal{C}_7) = 1$ and $\tilde{\mathcal{M}}_{\mathcal{C}}(\mathcal{C}_1) = \tilde{\mathcal{M}}_{\mathcal{C}}(\mathcal{C}_2) = \tilde{\mathcal{M}}_{\mathcal{C}}(\mathcal{C}_5) = \tilde{\mathcal{M}}_{\mathcal{C}}(\mathcal{C}_6) = 2$ for $(\mathcal{V}, v_2)$ as shown in Fig. 3(b). Cut pt-edges (p_1, n_1) and (p_1, n_2) in $\tilde{\Pi}$ incur migration of clusters $\mathcal{C}_1$ and $\mathcal{C}_2$ from processor P_1 to P_2 before the rendering of $(\mathcal{V}, v_2)$. Cut pt-edge (p_2, n_7) incurs migration of $\mathcal{C}_7$ from P_2 to P_1. Cut tt-edges (n_1, n_3), (n_2, n_3), (n_3, n_6), and (n_6, n_7) represent the sets of boundary faces shared between the respective cluster pairs that will incur communication and redundant computation during the rendering of $(\mathcal{V}, v_2)$.

4. RM-MeTiS: MeTiS-based remapping heuristic

We exploit the state-of-the-art GP tool MeTiS [23] (kMeTiS option) for partitioning the remapping graph $\tilde{\mathcal{G}}$. This section presents our modifications and enhancements to the MeTiS package that make it support the mapping constraint (Eq. (3)). MeTiS consists of three phases: *multilevel coarsening*, *initial partitioning* (referred to as *initial remapping* in RM-MeTiS), and *multilevel refinement*.

4.1. Multilevel coarsening

In this phase, the given graph $\tilde{\mathcal{G}} = \tilde{\mathcal{G}}_0$ is coarsened into a sequence of smaller graphs $\tilde{\mathcal{G}}_1, \tilde{\mathcal{G}}_2, \ldots, \tilde{\mathcal{G}}_m$ until the coarsest graph $\tilde{\mathcal{G}}_m$ becomes sufficiently small. This coarsening is achieved by coalescing disjoint node pairs of graph $\tilde{\mathcal{G}}_\ell$ into

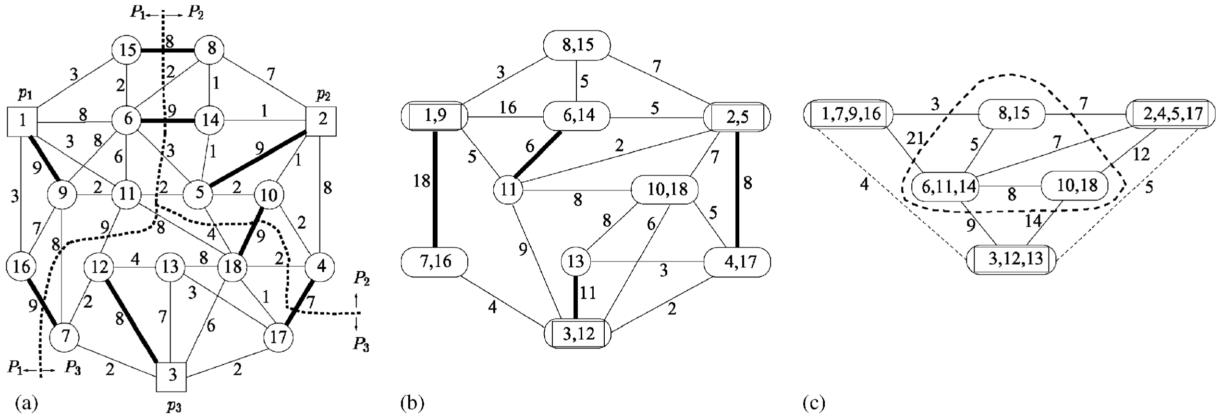

Fig. 4. 2-level coarsening of a sample remapping graph for a 3-processor system. (a) The original remapping graph $\tilde{\mathcal{G}} = \tilde{\mathcal{G}}_0$ with the current mapping $\Pi = \{\{p_1, n_6, n_9, n_{11}, n_{15}, n_{16}\}, \{p_2, n_4, n_5, n_8, n_{10}, n_{14}\}, \{p_3, n_7, n_{12}, n_{13}, n_{17}, n_{18}\}\}$ and the randomized heavy-edge matching (the set of bold edges) in $\tilde{\mathcal{G}}_0$. (b) The coarser remapping graph $\tilde{\mathcal{G}}_1$ and the matching in $\tilde{\mathcal{G}}_1$. (c) The coarsest remapping graph $\tilde{\mathcal{G}}_2$.

supernodes of the next level graph $\tilde{\mathcal{G}}_{\ell+1}$ through various randomized *matching* schemes in which nodes are visited in a random order. Among various matching criteria, the *heavy-edge matching scheme* is selected for RM-MeTiS. In heavy-edge matching, a node n_i is matched with a node n_j if the weight of the edge between these two nodes is maximum over all valid edges incident to node n_i.

At each level ℓ, $\tilde{\mathcal{G}}_\ell$ effectively induces an $|\tilde{\mathcal{N}}_\ell|$-way partition of $\tilde{\mathcal{G}}_0$. RM-MeTiS maintains the mapping constraint in a relatively relaxed manner such that each supernode of $\tilde{\mathcal{G}}_\ell$ contains at most one p-node of $\tilde{\mathcal{G}}_0$. Two unmatched nodes are considered for matching only if at least one of them is a t-node. Matching two t-nodes in $\tilde{\mathcal{G}}_\ell$ creates a t-supernode in $\tilde{\mathcal{G}}_{\ell+1}$, whereas matching a t-node with a p-node creates a p-supernode. So, the constituent nodes of a t-supernode are all t-nodes, whereas the constituent nodes of a p-supernode are all t-nodes except one p-node. There exist exactly K p-supernodes in $\tilde{\mathcal{G}}_\ell$ at each level ℓ.

Fig. 4 shows 2-level coarsening of a sample remapping graph $\tilde{\mathcal{G}}$ containing 15 t-nodes and three p-nodes. In Fig. 4(a), the circles and squares denote t-nodes and p-nodes, respectively. In Figs. 4(b) and (c), the circles/ellipses represent t-supernodes, and each ellipse with a rectangle inside represents a p-supernode. The numbers inside the circles/ellipses represent the indices of the nodes constituting the respective nodes/supernodes. The numbers beside the edges denote their weights. For simplicity, unit weights are assumed for t-nodes. Recall that p-nodes have zero weights. Hence, the current 3-way mapping, shown by the dashed curve in Fig. 4(a), achieves perfect load balance by assigning five t-nodes to each processor. In Figs. 4(a) and (b), decimal ordering is assumed to be the random node visit order used during the matching. In Fig. 4(b), the node visit order is determined by the index of the smallest-indexed node in a supernode. The matching shown in Fig. 4(a) contains eight edges of $\tilde{\mathcal{G}}_0$, where nodes n_{11} and n_{13} remain unmatched. Fig. 4(b) shows the coarser graph $\tilde{\mathcal{G}}_1$ obtained by contracting $\tilde{\mathcal{G}}_0$ according to the matching in Fig. 4(a).

In Fig. 4(c), the two dashed edges incident to p-supernode $[p_3, n_{12}, n_{13}]$ denote the two pp-edges of $\tilde{\mathcal{G}}_2$. Although $\tilde{\mathcal{G}}_0$ does not contain any pp-edges, coarser graphs may contain such edges because of merging two adjacent t-nodes with different p-nodes. These edges are not considered during the coarsening phase since two p-supernodes cannot be matched due to the mapping constraint. They are not considered during the initial partitioning and refinement phases since p-supernodes are not allowed to move as described in the following sections. The pp-edges always remain on the cut during the initial partitioning phase and continue to remain on the cut during the uncoarsening phase until they disappear due to node expansions.

4.2. Initial remapping

In RM-MeTiS, the objective of the initial partitioning phase is to find a K-way partition of $\tilde{\mathcal{G}}_m$ satisfying the mapping and GP balance constraints. We adopt a direct K-way initial partitioning scheme instead of the recursive bisection scheme of pMeTiS [22] adopted in kMeTiS [21] since it is harder to maintain the mapping constraint during the recursive bisection and the intrinsic features of $\tilde{\mathcal{G}}_m$ can be efficiently exploited in direct K-way initial partitioning. The *greedy graph growing partitioning* (GGGP) algorithm is extended for K-way initial partitioning. GGGP starts from an initial bipartition, where a randomly selected node and all remaining nodes form the *growing* and *shrinking* parts, respectively. Then, the boundary nodes of the shrinking part are moved to the growing part according to their Fiduccia–Mattheyses (FM) [13] gains until the balance constraint is satisfied. The FM gain of a move is the change in the cutsize if the move is realized.

Extension of GGGP to K-way partitioning requires selection of $K-1$ such nodes for determining the growing parts. Fortunately, the property that $\tilde{\mathcal{G}}_m$ contains exactly K p-supernodes and $|\tilde{\mathcal{N}}_m| - K$ t-supernodes can be considered as inducing a $(K+1)$-way initial partition of $\tilde{\mathcal{G}}_m$ such that each p-supernode constitutes a p-part and all t-supernodes constitute a single t-part. The K p-parts are the natural candidates for $K-1$ growing parts, and the remaining p-part together with the t-part constitute the shrinking part. So, the problem is the selection of a good shrinking part, which is, in fact, finding a good p-part

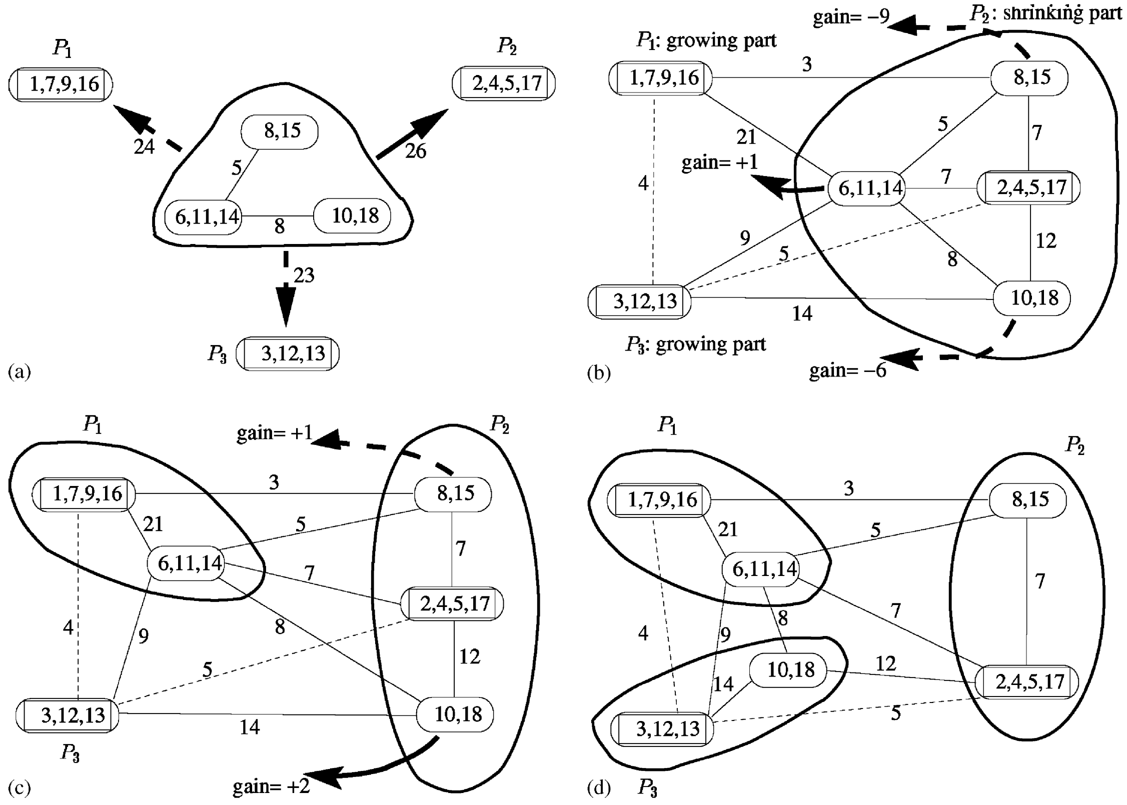

Fig. 5. The initial remapping phase. (a) The attraction values for the t-part. (b) The t-part is assigned to the most attractive p-part P_2 constituting the shrinking part. t-supernode $\{n_6, n_{11}, n_{14}\}$ with the maximum move gain of $+1$ moves from P_2 to P_1, reducing the cutsize from 56 to 55. (c) t-supernode $\{n_{10}, n_{18}\}$ with the maximum move gain of $+2$ moves from P_2 to P_3, reducing the cutsize to 53. (d) Remapping $\tilde{\Pi}_2 = \{\{[p_1, n_7, n_9, n_{16}], [n_6, n_{11}, n_{14}]\}, \{[p_2, n_4, n_5, n_{17}], [n_8, n_{15}]\}, \{[p_3, n_{12}, n_{13}], [n_{10}, n_{18}]\}\}$ on $\tilde{\mathcal{G}}_2$.

assignment for all t-supernodes. In RM-MeTiS, all t-supernodes are assigned to the most attractive p-part. The attraction of a p-part is defined as the sum of all pt-edges between the respective p-supernode and all t-supernodes.

Each t-supernode in the shrinking part is associated with $K-1$ moves since it can move to $K-1$ different growing parts. So, $K-1$ FM move gains for each boundary t-supernode are computed, and those t-supernodes are inserted into a max-heap priority queue according to their maximum move gains. At each step, the move with the maximum gain is realized if the size of the destination growing part does not exceed the maximum allowed part size after the move. If that move violates the size constraint at the destination part, the new maximum move gain of the same t-supernode to the remaining shrinking parts is computed and reinserted into the priority queue. If all moves associated with a t-supernode violate the maximum part-size constraint, the node is not considered for further moves and remains in the shrinking part. After each move, the maximum move gains of the t-supernodes in the shrinking part which are adjacent to the moved t-supernode are updated. These moves are realized even if their gains are negative until the size of the shrinking part drops below the maximum allowed part size. Then, only the moves with positive gains are permitted so that the algorithm terminates when either a move with a negative gain is encountered or the weight of the shrinking part decreases below the minimum allowed part size. Fig. 5 illustrates the algorithm on the partition obtained in Fig. 4(c) for the coarsest graph $\tilde{\mathcal{G}}_2$. In Fig. 5, the imbalance ratio is assumed to be 10% so that the maximum and minimum part sizes allowed are 6 and 4, respectively.

4.3. Multilevel refinement

At each level ℓ, for $\ell = m, \ldots, 1$, partition $\tilde{\Pi}_\ell$ found on $\tilde{\mathcal{G}}_\ell$ is projected back to a partition $\tilde{\Pi}_{\ell-1}$ on $\tilde{\mathcal{G}}_{\ell-1}$ by assigning the constituent nodes of each supernode of $\tilde{\mathcal{G}}_\ell$ to the part of their supernode. Obviously, $\tilde{\Pi}_{\ell-1}$ has the same cutsize with $\tilde{\Pi}_\ell$. As the finer graph $\tilde{\mathcal{G}}_{\ell-1}$ has more freedom in possible node moves, $\tilde{\Pi}_{\ell-1}$ is refined using an iterative improvement heuristic based on boundary node moves. In RM-MeTiS, a modified version of the *global Kernighan–Lin refinement* (GKLR) algorithm [24] is used. GKLR iterates a number of passes until a locally optimum solution is found. All nodes are *unlocked* at the beginning of each pass. At each step in a pass, the move with the maximum FM gain (even if it is negative) which does not disturb the balance constraint is tentatively performed. The node associated with the move is locked so that it cannot move again during the same pass. At the end of a pass, the moves in the prefix subsequence of moves with the maximum gain sum are realized if the maximum prefix sum is positive, and the next pass starts from this resulting partition. Allowing moves with negative gains brings hill climbing ability to the algorithm.

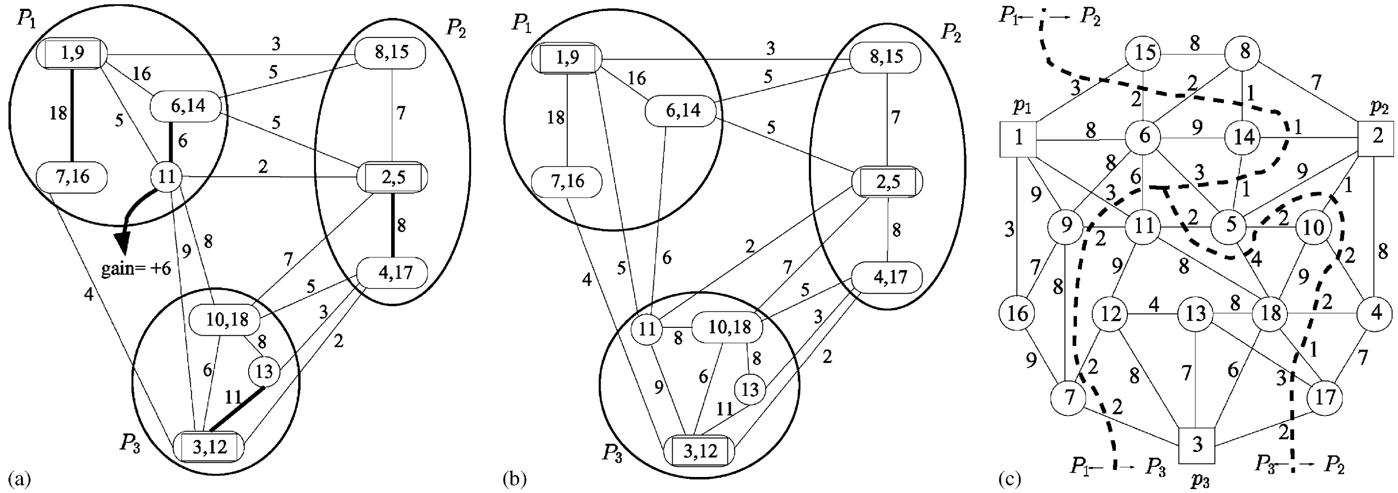

Fig. 6. The multilevel refinement phase. (a) Projection of $\tilde{\Pi}_2$ on $\tilde{\mathcal{G}}_2$ to a partition $\tilde{\Pi}_1$ on $\tilde{\mathcal{G}}_1$. t-node n_{11} with the maximum move gain of +6 moves from part P_1 to part P_3, reducing the cutsize from 53 to 47. (b) The refined remapping $\tilde{\Pi}_1 = \{\{[p_1, n_9], [n_7, n_{16}], [n_6, n_{14}]\}, \{[p_2, n_5], [n_4, n_{17}], [n_8, n_{15}]\}, \{[p_3, n_{12}], [n_{10}, n_{18}], n_{11}, n_{13}\}\}$ found on $\tilde{\mathcal{G}}_1$. (c) Projection of $\tilde{\Pi}_1$ to $\tilde{\Pi}_0 = \{\{p_1, n_6, n_7, n_9, n_{14}, n_{16}\}, \{p_2, n_4, n_5, n_8, n_{15}, n_{17}\}, \{p_3, n_{10}, n_{11}, n_{12}, n_{13}, n_{18}\}\}$ on the original graph $\tilde{\mathcal{G}}_0 = \tilde{\mathcal{G}}$.

The locking mechanism employed in GKLR is efficiently exploited in RM-MeTiS to maintain the feasibility of all partitions obtained during the uncoarsening phase by simply keeping all p-supernodes always in a locked state. As partition $\tilde{\Pi}_m$ obtained for the coarsest graph $\tilde{\mathcal{G}}_m$ is a feasible partition satisfying the mapping constraint, this simple locking mechanism on p-supernodes maintains the feasibility of further refinements by preventing the p-supernodes from moving to other parts. Fig. 6 illustrates the uncoarsening phase. Remapping $\tilde{\Pi}_0$ obtained in Fig. 6(c) for $\tilde{\mathcal{G}}$ reduces the cutsize of the previous mapping Π given in Fig. 4(a) from 80 to 47 while maintaining the perfect balance. Remapping $\tilde{\Pi}_0$ achieves this cutsize reduction by decreasing the tt-edge component of the cutsize from 80 to 35 through migration of the six tasks corresponding to nodes $n_7, n_{10}, n_{11}, n_{14}, n_{15}$, and n_{17} with a total migration cost of $2 + 1 + 3 + 1 + 3 + 2 = 12$.

5. Parallel DVR algorithm

Our OS-parallel DVR algorithm consists of four consecutive phases: *clustering*, *remapping*, *local rendering*, and *pixel merging*. The clustering phase is executed only once at the very beginning. The last three phases are effectively executed for each successive visualization instance.

5.1. Clustering phase

Since $\mathcal{V}$ is produced in a distributed manner by a parallel simulation application, construction of the global graph $\mathcal{G}_{\mathcal{V}}$ for partitioning requires a high volume of communication. So, we adopt a local clustering scheme concurrently performed at each processor to reduce the view-independent preprocessing overhead. In this scheme, each processor P_k constructs its local portion $\mathcal{G}_{\mathcal{V}_k}$ of $\mathcal{G}_{\mathcal{V}}$. Then, each P_k computes the view-independent weights for the nodes and edges of its local $\mathcal{G}_{\mathcal{V}_k}$ to construct its local clustering graph $\mathcal{G}_{\mathcal{V}_k}^{\text{avg}}$ according to the weighting scheme in Section 2.2. Finally, each P_k performs α_k-way partitioning of its local $\mathcal{G}_{\mathcal{V}_k}^{\text{avg}}$ using MeTiS. The local number α_k of the resulting clusters in each P_k is selected to be proportional to the total node weight in its local $\mathcal{G}_{\mathcal{V}_k}^{\text{avg}}$ such that $\sum_{k=1}^{K} \alpha_k = \alpha$. This scheme is an effort towards reducing the variation in cluster weights for better performance in partitioning the coarse remapping graphs.

After the local clustering, each processor constructs a *ClusterData* structure for each of its local cell clusters. The two major components of the *ClusterData* structure are the *VtxArray* and *CellArray* structures (see Appendix A), which store the tetrahedral cell data associated with the respective cluster. Local cell and node indexing is utilized within the *CellArray* and *VtxArray* structures of each cell cluster. To maintain the connectivity information between the cell clusters, within this local indexing scheme, the four global neighbor-cell identifiers of each cell in the *CellArray* structures are replaced by the global cluster identifiers and local indices of the respective neighbor cells. Each *ClusterData* structure also maintains the total volume (in the WCS) of the respective local cell cluster, computed as the sum of the volumes of the constituent cells of the cell cluster. As described in Section 2.3, this view-independent volume information is utilized in estimating the view-dependent computational weights of the cell clusters during the remapping phase.

The final step of this phase is the construction of the view-independent portions of the view-dependent remapping graph $\tilde{\mathcal{G}}_{\mathcal{C}}^{v}$ determined by the 3-tuple $(\mathcal{G}_{\mathcal{C}}^{v}, \mathcal{M}_{\mathcal{C}}, \mathcal{T}_{\mathcal{C}})$ to reduce the overhead of the repeated remapping phases. Recall that the topology of $\mathcal{G}_{\mathcal{C}}^{v}$ does not change with changing v and is identical to that of the view-independent coarse graph $\mathcal{G}_{\mathcal{C}}$. So, each processor concurrently constructs the adjacency list representation of its local portion of $\mathcal{G}_{\mathcal{C}}$ by contracting its local $\mathcal{G}_{\mathcal{V}_k}$ according to $\mathcal{C}$. Then, a copy of the topology of the global graph $\mathcal{G}_{\mathcal{C}}$ is replicated at each processor by an all-to-all broadcast [16] operation. Another view-independent component of the remapping tuple is the cluster migration cost function $\mathcal{T}_{\mathcal{C}}$. Each processor computes the sizes of the *ClusterData* structures of its local clusters to determine their migration costs.

5.2. Remapping phase

This phase consists of three steps: *graph update*, *graph partitioning*, and *cluster migration*. In the graph-update step, each processor concurrently computes the node and edge weights of its local portion of $\mathcal{G}_{\mathcal{C}}^{v}$ for a given v according to the weight estimation schemes described in Section 2.3. Then, an all-to-all broadcast operation is performed on these local node and edge weights so that each processor gathers a copy of the weighted global graph $\mathcal{G}_{\mathcal{C}}^{v}$. Finally, processors complete $\tilde{\mathcal{G}}_{\mathcal{C}}^{v}$ by adding the weighted pt-edges using the current cluster mapping $\mathcal{M}_{\mathcal{C}}$. In the GP step, all processors concurrently execute RM-MeTiS for K-way partitioning of their copies of remapping graph $\tilde{\mathcal{G}}_{\mathcal{C}}^{v}$ using different random seeds. Then, the best partition is determined by performing a global-minimum operation on the cutsize values of the local partitioning solutions, and the processor which produced the best solution broadcasts its part vector, which corresponds to the new mapping $\tilde{\mathcal{M}}_{\mathcal{C}}$. This scheme is an effort towards avoiding the worst-case behavior of RM-MeTiS, which is a randomized algorithm. In the cluster migration step, the *ClusterData* structures of the clusters of which new mappings differ from their current mappings migrate to their new home processors.

5.3. Local rendering phase

In GP, minimizing the cutsize is equivalent to maximizing the sum of the weights of uncut edges. The uncut tt-edges in a partition of the remapping graph represent the pseudo-boundary faces which are shared between two local clusters of a processor. Hence, after the remapping, each processor has a set of highly interacting clusters for local rendering. Thus, rendering the local clusters independently by considering them as individual subvolumes incurs substantial amount of redundant computation and storage because of the ray segments to be generated for the ff pseudo-boundary faces. Our implementation considers the local clusters of each processor as a single local subvolume to be rendered by Koyamada's sequential algorithm.

In the local rendering phase, processors concurrently traverse the local *CellArray* structures and check each face *f* of each local cell c_i to determine whether it is an external or a boundary face. A face *f* is identified as an external face if cell c_i does not have a neighbor cell through face *f*. A face *f* is identified as a boundary face if cell c_i is neighbor to a cell c_j in cluster C_q through face *f* such that C_q is a non-local cluster of P_k. Each of the ff external and ff boundary faces is scan converted, and a ray segment is generated for each pixel covered by the face. Each ray segment is followed through the local subvolume using Koyamada's algorithm until it exits from a bf external face or a bf boundary face, and then it is inserted into the list of the respective pixel in the local *RayBuffer* structure, in sorted order, according to its exit z value. The efficient ray traversal scheme of Koyamada's algorithm is not disturbed even if two successive cells on the route of a ray belong to two different local clusters since *ClusterData* structures preserve the connectivity information between the cells despite the clustering. For example, in Fig. 7, ray r_2, which passes through four local clusters, is processed by processor P_1 as a single ray segment as in the sequential algorithm.

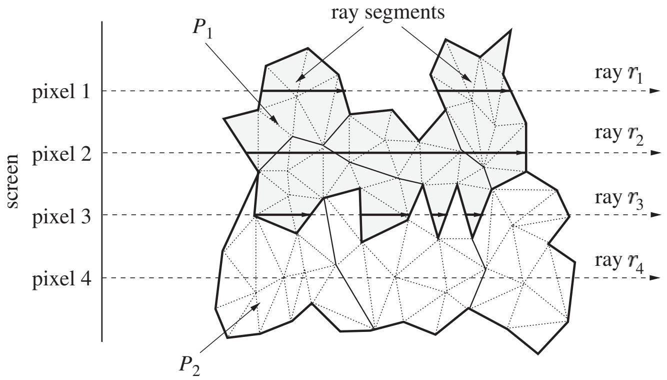

Fig. 7. The local rendering of the shaded subvolume (consisting of four cell clusters) by processor P_1.

Our implementation of Koyamada's algorithm requires a *RayBuffer* structure since multiple ray segments may be generated for pixels in non-convex datasets. However, even if a dataset is convex, OS decomposition may generate local subvolumes with virtual non-convexities despite the explicit effort towards minimization in remapping. These virtual non-convexities increase the number of local ray segments. For example, in Fig. 7, processor P_1 generates two ray segments for pixel 1 due to the non-convexity of the volume as in the sequential algorithm. However, although the sequential algorithm generates only one ray segment for pixel 3, P_1 and P_2, respectively, generate four and five local ray segments due to the virtual non-convexities.

5.4. Pixel merging phase

Since the composited ray segments residing in the local *RayBuffer* structures of different processors may contribute to the same pixels, a global merge operation is needed to obtain the final image. For example, in Fig. 7, the ray segments generated by processors P_1 and P_2 for pixel 3 must be gathered and composited to determine the final color of pixel 3. The pixel merging phase consists of three steps: *pixel assignment* (PA), *ray-segment migration* (RSM), and *local pixel merging* (LPM).

5.4.1. Pixel assignment step

In the PA step, the screen is partitioned into K subregions such that each processor is held responsible for producing the final image of a distinct subregion through LPM. The problem in this step can be considered as the independent task assignment problem where merging of the ray segments belonging to individual pixels constitute the independent atomic tasks. The objective is to minimize the communication overhead due to RSM and to maintain the load balance in the following LPM step. The overhead of this step must be kept minimal because of the fine granularity of the pixel merging operations. For that reason, a 2D coarse mesh is imposed on the screen and mesh cells are used as atomic pixel merging tasks.

The following definitions are provided for the sake of the clarity of the presentation. For a mesh cell m, Ψ_m denotes the subset of processors which generated ray segments for the pixels in m during the local rendering phase. The local ray segment count r_{km} of a processor P_k for m denotes the number of ray segments generated by P_k for m. The global ray segment count $R_m = \sum_{P_k \in \Psi_m} r_{km}$ of m denotes the total number of ray segments produced by all processors for m. The computational load of a mesh cell m during the LPM step is proportional to R_m. In this work, three PA schemes are tried: *scattered assignment* (SA), *minimum-communication assignment* (MCA), and *balanced-load minimized-communication assignment* (BLMCA).

5.4.1.1. Scattered assignment (SA). In this scheme, mesh cells are assigned to processors in a scattered fashion for load balancing in the LPM step. The performance of this scheme depends on the assumption that the neighbor mesh cells are likely to have equal workload since they have similar views of the volume. The main advantage of this scheme is the simplicity. The major deficiency of this scheme is the lack of an explicit minimization effort for the communication overhead of the RSM step. However, the SA scheme can be expected to achieve a balance on the communication requirements of individual processors, thus reducing the concurrent communication volume during the RSM step.

5.4.1.2. Minimum-communication assignment (MCA). In the MCA scheme, it is assumed that the bottleneck in global pixel merging is the RSM step rather than the LPM step. Hence, the total volume of communication in the RSM step is minimized while totally ignoring the load balancing for the LPM step. The MCA scheme is based on the following simple observation. Assigning a mesh cell m to a processor P_k incurs the migration of the respective ray segments from each processor $P_\ell \in (\Psi_m - \{P_k\})$ to P_k, while avoiding the migration of the respective local ray segments of processor P_k. That is, this assignment incurs a communication volume of $R_m - r_{km}$. Hence, the greedy assignment of each mesh cell m to the processor $P_q \in \Psi_m$, which has the maximum local ray segment count for m, produces an assignment with a globally minimum communication volume.

The MCA scheme is performed in parallel as follows. Each processor concurrently traverses its local *RayBuffer* structure to construct a local workload (WL) matrix, whose size is equal to the coarse mesh size such that each entry of the WL matrix contains the local ray segment count for the respective mesh cell. Then, a global-maximum operation is performed on the local WL matrices through a fold and expand communication step so that, for each mesh cell, the processor with the maximum local ray segment count is determined, and the mesh cell is assigned to that processor.

5.4.1.3. Balanced-load minimized-communication assignment (BLMCA). This scheme considers both the volume of communication in the RSM step and the load balance in the LPM step. The MCA scheme constitutes an optimal solution to the former objective, whereas the latter one corresponds to the K-way number partitioning problem, which is NP-hard. The numbers in the number partitioning problem correspond to the global ray segment counts of the mesh cells. The best-fit-decreasing (BFD) heuristic used in solving the K-feasible bin-packing problem [20] can be effectively used for the solution of the number partitioning problem and hence the pure load balancing problem. In the BFD-based load balancing, mesh cells are considered for assignment in decreasing order of their global ray segment counts. The best-fit criterion is the assignment of the mesh cells to the minimally loaded bins (processors), and the bin capacity is the maximum allowable part size.

The BLMCA scheme incorporates the greedy assignment criterion of the MCA scheme to the BFD-based load balancing heuristic by replacing its best-fit criterion by the criterion of the MCA scheme for feasible assignments. The proposed heuristic starts with initializing the workloads of all processors to zero. Then, a mesh cell m, considered in the sorted order, is assigned to processor P_k having the maximum local ray-segment count r_{km} for m if the current load of P_k remains below the maximum part size after the assignment. Otherwise, the mesh cell m is assigned to the minimally loaded processor, where this assignment criterion corresponds to the best-fit criterion used in the BFD-based pure load balancing heuristic. After the assignment, the load of the respective processor is incremented by R_m. A good property of the BLMCA heuristic is that the maximum amount of communication volume that can be avoided by the assignment of a heavily loaded mesh cell is likely to be larger than that of a lightly loaded mesh cell. Hence, delaying the assignment of lightly loaded mesh cells minimizes the deviation from both the minimal communication cost to be found by the MCA algorithm and the load balance quality to be found by the pure BFD-based load balancing heuristic. The parallelization of the BLMCA scheme is similar to that of the MCA scheme with an additional global-sum operation carried out together with the global-max operation on the local WL matrices.

5.5. Ray-segment migration step

In the RSM step, processors concurrently traverse their local *RayBuffer* structures and packetize the ray segment lists that belong to the mesh cells assigned to other processors into the respective send buffers. Then, these buffers are sent to the respective processors so that each processor gathers all the ray segments belonging to the pixels assigned to itself.

5.6. Local pixel merging step

In LPM, each processor concurrently composites the gathered ray segments for each of its assigned pixels in the visibility order. This composition operation for each local pixel involves merging of k sorted ray-segment lists, where k denotes the number of distinct processors that generated ray segments for that particular pixel. A binary heap is used for K-way merging. At the end of this step, each processor holds the final color of each pixel assigned to itself.

6. Experimental results

The sequential and parallel DVR algorithms are developed on a 28-node PC cluster interconnected by a Gigabit Ethernet switch. Each node of the cluster contains an Intel Pentium IV 2.6 GHz processor with 1 GB of RAM and runs the Debian/GNU Linux operating system. The parallel DVR algorithms are implemented in C using the LAM/MPI library [2] as the communication interface.

Table 2 summarizes the properties of the volumetric datasets used in the experiments. These datasets, obtained from NASA Ames Research Center [35], are commonly used by the researchers in the volume rendering field. All datasets are originally curvilinear in structure, and they represent the results of some CFD simulations. The raw datasets consist of hexahedral cells, and they are converted into the unstructured tetrahedral data format by dividing each hexahedral cell into five tetrahedral cells [14,43]. In Table 2, the datasets are listed in the increasing order of both the number of nodes and cells. This order is maintained in all the following tables and figures. In Table 2, each COV value represents the *coefficient of variation* of cell sizes in terms of the cell volumes of the respective dataset. The COV value of a dataset can be considered as an indication of the level of irregularity in the dataset such that larger COV values mean more irregular datasets. Fig. 8 displays the rendered images of the datasets.

As these datasets are the results of CFD simulations, a parallel CFD simulation on K processors can be modeled by decomposing each dataset into K subvolumes by partitioning its associated CFD computational graph using MeTiS and assigning the data structures associated with each subvolume to a distinct processor. In CFD applications, both the computational costs of cells and the amount of interactions through the shared faces between neighbor cells are equal. Thus, the computational graph has unit node and edge weights, and its topology is identical to that of the view-independent visualization graph $\mathcal{G}_{\mathcal{V}}$. In all of our experiments, the execution of the proposed parallel DVR algorithm starts from such initial decompositions.

Seven different viewing directions, including the standard viewing direction, are used in the experiments to measure the average-case performance. In the standard view, the datasets are viewed along the z-axis in the positive z direction in the WCS. The other six views are obtained by successively rotating the volume 30° around all axes. The experiments are carried out for three different screen sizes: 400×400, 600×600, and 900×900. The window on the view plane is selected such that the whole volume is visible through the window, and a thin margin exists between the borders of the window and the projected volume. As RM-MeTiS is a randomized algorithm, the parallel DVR algorithm is executed 10 times for each parallel visualization instance, and the average performance results are displayed in the tables and figures.

6.1. Clustering

Table 3 illustrates the variation of the average performance of the parallel DVR algorithm with the number α of clusters to be generated in the clustering phase. The parallel rendering time T_{par} of a visualization instance is expected to decrease with increasing α because of the expected increase in the quality of the partitions to be found by RM-MeTiS due to the increase in the size of the solution space. On the contrary, T_{par} is expected to increase with increasing α because of the increase in the view-dependent remapping overhead. Hence, T_{par} is expected to decrease with increasing α until a turnover value after which T_{par} begins to increase. This behavior can easily be seen in Table 3. As also seen in the table, $\alpha_{\mathrm{avg}}=\alpha/K$ values leading to the best parallel performance decrease with increasing K as expected. In all of the following experiments, $\alpha = 1200$ is used in clustering, and hence we obtain $\alpha_{\mathrm{avg}} = 300, 150, 100, 75, 60, 50, 43$ for $K = 4, 8, 12, 16, 20, 24, 28$ processors, respectively.

Fig. 9 displays the view-independent clustering time T_{CL} as a percent of the parallel rendering time. As seen in Fig. 9, the percent clustering overhead $\%T_{\mathrm{CL}}$ is at most 60% of a single parallel rendering time. As T_{CL} is independent of the screen size, $\%T_{\mathrm{CL}}$ decreases with increasing screen size for a fixed (dataset, K) pair and it is always below 20% for the screen size 900 × 900. Moreover, $\%T_{\mathrm{CL}}$ remains almost constant with increasing K for a fixed (dataset, screen size) pair. The results show that the local clustering scheme proposed in Section 2.2 makes the view-independent clustering overhead negligible after only a few successive parallel visualizations. Hence, the performance results are obtained by considering the view-dependent parallel execution times, which involve remapping, local rendering, and pixel merging phases.

6.2. Adaptive decomposition and remapping

Table 4 illustrates the performance of the proposed estimation schemes used for the node and edge weighting of the view-dependent coarse visualization graph $\mathcal{G}_C^v$. As seen in Table 4, estimation errors in both ray–face intersection counts (I) and sampling counts (S) are extremely low, where $\%S^{\mathrm{err}}$ values are drastically lower than $\%I^{\mathrm{err}}$ values as expected. Because, in estimation of the I count of a cluster, the discretization errors in estimation of the pixel counts of the faces that have overlapping projection areas accumulate, whereas the discretization errors in estimation of the S count is just on the boundary surfaces of clusters. The percent errors in estimation of local rendering times are considerably larger than both $\%I^{\mathrm{err}}$ and $\%S^{\mathrm{err}}$ values, but $\%T_{\mathrm{LR}}^{\mathrm{err}}$ values still remain around 3%. This experimental finding stems from the additional errors introduced due to estimation of unit computational costs t_I and t_S. Fortunately, the estimation errors in T_{LR} are not reflected to the estimated load imbalance values because of the cancelation effects. Note that $\%I^{\mathrm{err}}$ also determines the estimation error in edge weighting of graph $\mathcal{G}_C^v$ since estimation of I in node weighting and estimation of edge weights both involve the same kind of face area computations.

Table 5 displays the performance results for the static decomposition, adaptive repartitioning (RP), and adaptive repartitioing with remapping (RM) schemes on 28 processors. As seen in Table 5, the proposed remapping scheme yields the best speedup in all cases. The static scheme [31] already

Table 2
The characteristics of the volumetric datasets used in the experiments

Datasets	# of nodes	# of cells	# of internal faces	# of external faces	COV
The blunt fin (BF)	40,960	187,395	368,032	13,516	5.50
The combustion chamber (CC)	47,025	215,040	422,272	15,616	0.42
The oxygen post (OP)	109,744	513,375	1,012,912	27,676	4.26

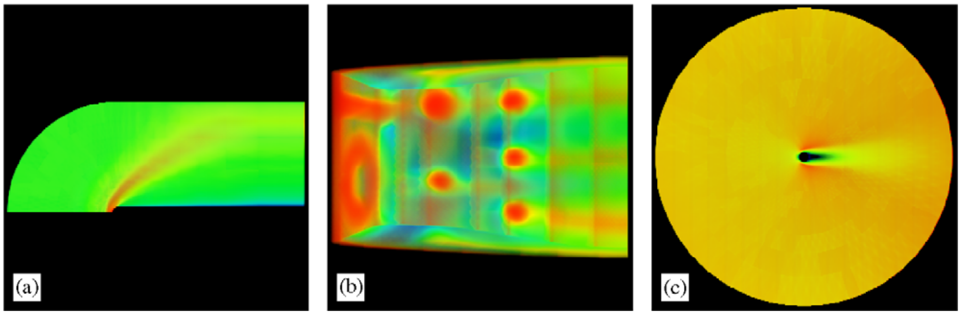

Fig. 8. The rendered images of the volumetric datasets used in the performance analysis: (a) the blunt fin, (b) the combustion chamber, and (c) the oxygen post.

Table 3
The variation of the average parallel rendering time with $\alpha_{avg} = \alpha/K$

K	α_{avg}				
	25	50	100	200	300
4	1.000	0.939	0.922	0.915	0.914
8	1.000	0.974	0.972	0.977	0.978
16	1.000	0.984	0.989	1.008	1.032
24	1.000	0.999	1.019	1.070	1.117

The view-dependent parallel execution times including remapping, local rendering, and pixel merging times are normalized with respect to those of the $\alpha_{avg} = 25$ case. The averaging is over all datasets and the medium screen size 600 × 600.

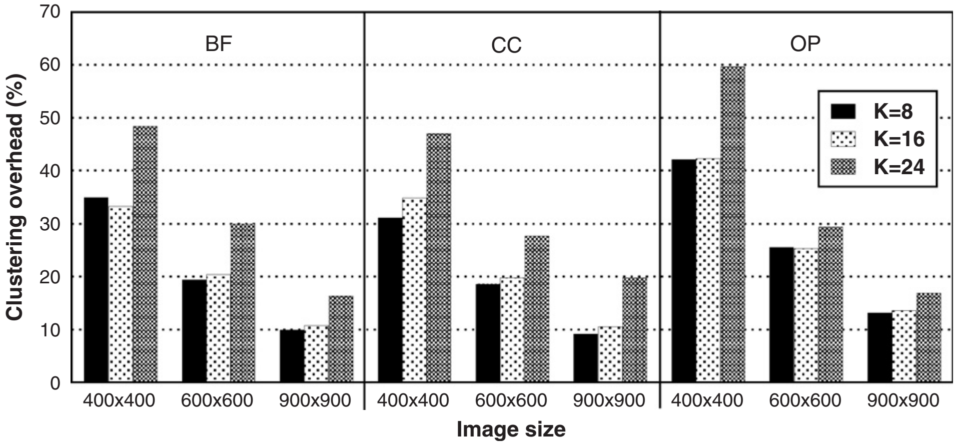

Fig. 9. The clustering overhead as a percent of the view-dependent parallel rendering time.

achieves good speedup values for the CC dataset, which has almost equal-sized cells. However, it results in drastically small speedup values as low as 3.26 on the BF and OP datasets (with high COV values) because of the extremely large load imbalance values reaching as high as 636%. This shows that, as the cell-size variation is inherently high in unstructured grids, adaptive decomposition is crucial for parallel DVR of unstructured grids.

The performance results of the RP scheme are also displayed in Table 5 to justify the need for considering the cluster

Table 4
Performance of the estimation schemes proposed for view-dependent node/edge weighting

Datasets	Screen sizes	% Error in estimates			% Load imbalance	
		$\%I^{\mathrm{err}}$	$\%S^{\mathrm{err}}$	$\%T_{\mathrm{LR}}^{\mathrm{err}}$	$\%LI_{\mathrm{e}}$	$\%LI_{\mathrm{m}}$
BF	400 × 400	1.316	0.016	3.329	4.753	8.694
	600 × 600	1.330	0.008	3.383	4.719	7.177
	900 × 900	1.335	0.005	3.499	4.919	6.743
CC	400 × 400	1.441	0.007	3.022	3.258	5.354
	600 × 600	1.447	0.003	3.053	2.916	4.833
	900 × 900	1.449	0.002	3.144	2.613	4.604
OP	400 × 400	1.128	0.023	3.280	3.968	8.613
	600 × 600	1.131	0.019	2.915	3.589	7.413
	900 × 900	1.135	0.017	2.809	3.239	5.792

$\%I^{\mathrm{err}}$, $\%S^{\mathrm{err}}$, and $\%T_{\mathrm{LR}}^{\mathrm{err}}$ denote the average percent errors in estimation of ray–face intersection counts, sampling counts, and local rendering times, respectively, of local subvolumes rendered by distinct processors. $\%LI_{\mathrm{e}}$ and $\%LI_{\mathrm{m}}$ denote the estimated and measured percent load imbalance values, respectively.

Table 5
Performance comparison on $K = 28$ processors

Datasets		Screen sizes								
		400 × 400			600 × 600			900 × 900		
		Decomposition schemes			Decomposition schemes			Decomposition schemes		
		Static	Adaptive		Static	Adaptive		Static	Adaptive	
			RP	RM		RP	RM		RP	RM
BF	S_K	3.26	10.37	13.13	3.56	14.00	16.25	3.32	15.26	17.20
	V_{CM}	0.00	10.32	1.86	0.00	10.36	2.29	0.00	10.32	2.94
	V_{RSM}	9.07	10.45	12.03	20.43	23.60	26.46	45.99	53.10	58.25
	$\%LI_{\mathrm{m}}$	636	21.45	12.81	643	22.08	10.99	649	21.68	11.88
CC	S_K	17.15	13.62	15.97	18.07	17.17	18.07	18.05	18.63	18.63
	V_{CM}	0.00	11.72	0.84	0.00	11.82	1.31	0.00	11.64	2.67
	V_{RSM}	10.82	10.90	11.48	24.36	24.52	25.50	54.85	55.19	56.97
	$\%LI_{\mathrm{m}}$	21.46	5.11	8.14	20.28	4.30	5.56	20.14	5.40	6.67
OP	S_K	6.85	12.38	16.95	6.81	15.51	19.07	6.74	18.99	20.19
	V_{CM}	0.00	26.87	4.17	0.00	26.51	4.49	0.00	27.12	5.30
	V_{RSM}	10.36	10.76	13.06	23.33	24.22	28.33	52.53	54.52	62.72
	$\%LI_{\mathrm{m}}$	300	11.80	12.12	303	9.39	9.73	305	9.19	8.03

In the static scheme, parallel rendering is conducted according to the OS decomposition inherited from the K-way partitioning of the unit node/edge weighted view-independent visualization graph, and hence this scheme is effectively equivalent to [31]. In the repartitioning (RP) scheme, the view-dependent coarse visualization graph $\mathcal{G}_{\mathcal{C}}^{v}$ is partitioned using MeTiS without considering the cluster migration cost. The RM scheme is the proposed remapping scheme. S_K represents the speedup at $K = 28$ processors, V_{CM} and V_{RSM} denote the communication volume (in Mbytes) due to cluster and ray-segment migration, respectively. $\%LI_{\mathrm{m}}$ denotes the measured percent load imbalance.

migration cost in adaptive decomposition. The comparison of the average performances of the adaptive schemes RP and RM shows that the decompositions produced by RM require 82% less cluster migration, whereas they incur only 10% more RSM, which corresponds to a 23% decrease in the total volume of communication. This experimental finding is expected since RM considers both cluster migration and ray-segment generation costs for minimization, whereas RP considers only the ray-segment generation cost for minimization. As seen in Table 5, the speedup difference between RM and RP on 28 processors is 2.2 on the overall average. Hence, the task migration cost should also be considered in adaptive OS decomposition for better parallel performance.

6.3. Overall performance

Fig. 10 shows the percent dissection of the view-dependent parallel rendering time. As seen in Fig. 10, the percent view-dependent preprocessing overhead ($\%T_{\mathrm{GU}} + \%T_{\mathrm{GP}}$) decreases with increasing screen size for a fixed (dataset, K) pair, and both $\%T_{\mathrm{GU}}$ and $\%T_{\mathrm{GP}}$ decrease almost below 1.5% in all visualizations using the largest screen size 900 × 900. This monotonic decrease is because both T_{GU} and T_{GP} are independent from the screen size. However, both $\%T_{\mathrm{GU}}$ and $\%T_{\mathrm{GP}}$ increase with increasing K for a fixed (dataset, screen size) pair, where the rate of increase in $\%T_{\mathrm{GP}}$ is larger than that of $\%T_{\mathrm{GU}}$. As node and edge weighting is performed in parallel, local graph-update

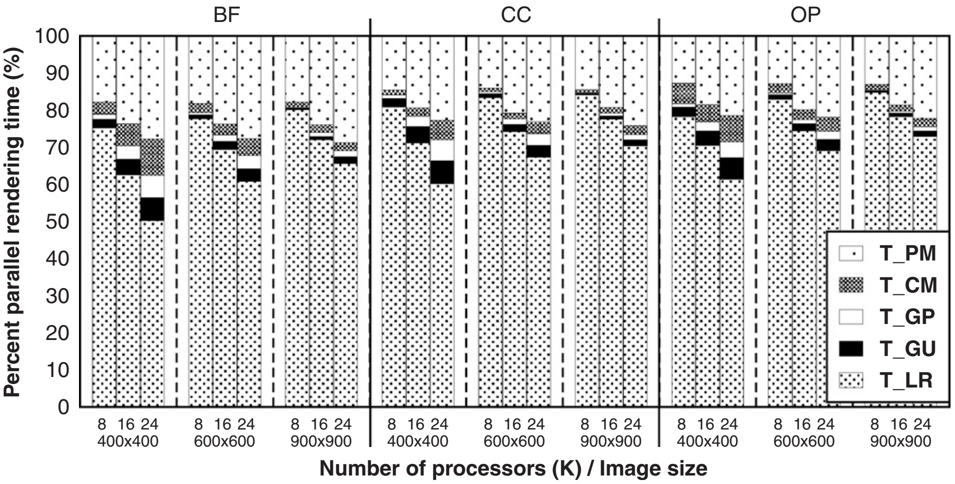

Fig. 10. The percent dissection of parallel rendering time into local rendering T_{LR}, graph update T_{GU}, graph partitioning T_{GP}, cluster migration T_{CM}, and pixel merging T_{PM} times.

computations do not contribute to the increase in $\%T_{GU}$. The communication overhead due to the all-to-all broadcast operation in the graph-update step is the source for this increase. The increase in $\%T_{GP}$ with increasing K is because RM-MeTiS is run serially on each processor.

Fig. 11 shows the communication volumes due to RSM (V_{RSM}) and cluster migration (V_{CM}). The following observations can be extracted from Fig. 11 for the variation of the pixel merging overhead. For a fixed (dataset, screen size) pair, V_{RSM} increases by almost a factor of 2 when K increases by a factor of 3 (from $K = 8$ to 24). For a fixed (dataset, K) pair, V_{RSM} increases with increasing screen size, where the rate of increase is similar to the rate of increase in the screen size. Note that increasing the screen size from 400×400 to 900×900 increases the screen size by a factor of approximately 5.

Table 6 shows the average rendering times, speedups, and percent efficiencies. As expected, the speedup and efficiency increase with increasing screen size for a fixed (dataset, K) pair. As seen in Table 6, the efficiency decreases slowly with increasing K for a fixed (dataset, screen size) pair, especially for the larger datasets CC and OP, and the larger screen sizes 600×600 and 900×900. For OP, the efficiency values remain above 60%, 68%, and 72% for screen sizes 400×400, 600×600, and 900×900, respectively, for all K.

Table 7 displays performance comparison of the PA schemes on the parallel pixel merging phase. In this table, T_{PM} denotes the total execution time of the pixel merging phase, and T_{PA}, T_{RSM} and T_{LPM} represent the dissection of T_{PM} into PA, RSM and LPM times, respectively. V_{RSM} denotes the total volume of communication during the RSM step. The mesh size is the size of the 2D coarse mesh imposed on the screen.

As seen in Table 7, the SA scheme achieves the best load balance in the LPM step for sufficiently large mesh sizes, whereas it incurs the worst communication volume. Note that T_{LPM} effectively shows the load balancing quality of the respective PA scheme since the local *RayBuffer* structures of the processors are identical along each row of the table. On the other hand, the MCA scheme achieves the lowest communication volume while incurring the worst load balance in the LPM step as expected. The BLMCA scheme shows in-between performance such that it almost achieves the load balancing quality of the SA scheme, and it approaches the communication volume quality of the MCA scheme. It is interesting to note that although the total communication volumes for both MCA and BLMCA schemes are considerably less than those of the SA scheme, the communication time values of the SA scheme are smaller than or equal to those of the MCA and BLMCA schemes. This is due to the fact that the scattering approach in the SA scheme achieves the balancing of communication requirements of individual processors, thus reducing the concurrent communication volume. Hence, the SA scheme, which does not involve any PA overhead, achieves the best overall performance in T_{PM} for all instances displayed in the table. The SA scheme with a coarse mesh size of 50×50 is used in all experiments.

6.4. Comparison with an IS-parallel DVR algorithm

We also conducted experiments to compare the performance of our OS-parallel DVR algorithm (RM) with our recently proposed IS-parallel DVR algorithm (IS) [3]. In this algorithm, the interaction between the image and OS is modeled as a hypergraph, where nets correspond to cell clusters and vertices correspond to pixel blocks assigned to processors for rendering. The remapping problem in IS parallelization is formulated as a hypergraph partitioning with fixed vertices problem. In this formulation, balancing the part weights corresponds to balancing the rendering loads of processors and minimizing the cutsize of the partition corresponds to minimizing the total volume of data communication due to the remapping of pixel blocks to processors. The details of this algorithm are provided in [3].

Table 8 provides dataset-specific speedups for varying numbers of processors and screen resolutions. According to

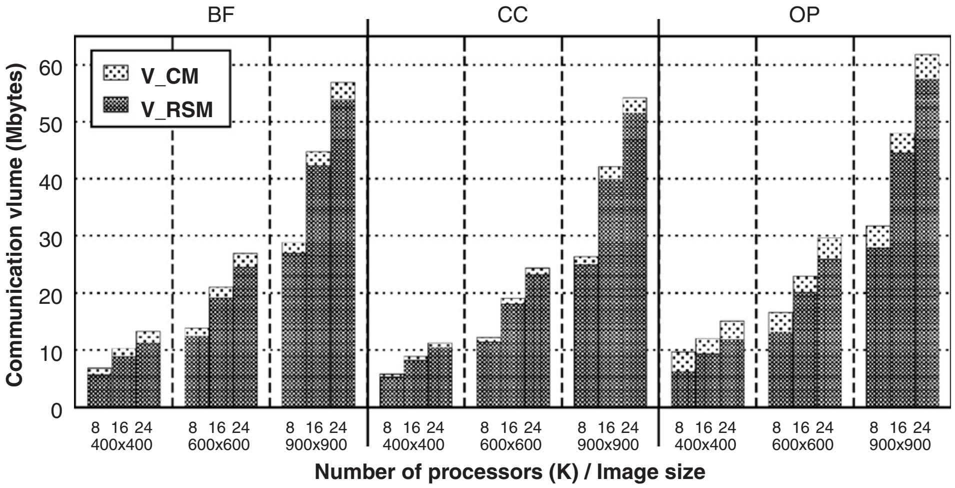

Fig. 11. The total communication volume (in Mbytes) and its dissection into ray-segment migration V_{RSM} and cluster migration V_{CM} components.

Table 6
Average rendering times (T_R) in seconds, speedups (S_K), and percent efficiency values ($\%E_K$)

Datasets	K	Screen sizes								
		400 × 400			600 × 600			900 × 900		
		T_R	S_K	$\%E_K$	T_R	S_K	$\%E_K$	T_R	S_K	$\%E_K$
BF	1	3.94	1.00	100	9.10	1.00	100	20.3	1.00	100
	4	1.19	3.31	82.8	2.63	3.46	86.5	5.82	3.49	87.3
	8	0.66	5.97	74.7	1.43	6.36	79.5	3.14	6.45	80.6
	12	0.48	8.21	68.4	1.01	9.01	75.1	2.21	9.16	76.3
	16	0.39	10.10	63.1	0.81	11.23	70.2	1.75	11.56	72.3
	20	0.35	11.26	56.3	0.68	13.38	66.9	1.47	13.81	69.0
	24	0.32	12.31	51.3	0.61	14.92	62.2	1.30	15.58	64.9
	28	0.30	13.13	46.9	0.56	16.25	58.0	1.18	17.20	61.4
CC	1	4.63	1.00	100	10.3	1.00	100	23.1	1.00	100
	4	1.35	3.42	85.5	2.98	3.46	86.5	6.56	3.52	88.0
	8	0.71	6.46	80.6	1.55	6.65	83.1	3.47	6.66	83.3
	12	0.50	9.13	76.1	1.08	9.55	79.6	2.39	9.67	80.6
	16	0.41	11.29	70.6	0.88	11.72	73.3	1.88	12.29	76.8
	20	0.35	13.01	65.0	0.72	14.32	71.6	1.57	14.71	73.5
	24	0.31	14.60	60.8	0.64	16.11	67.1	1.39	16.62	69.3
	28	0.29	15.97	57.0	0.57	18.07	64.5	1.24	18.63	66.5
OP	1	6.44	1.00	100	14.3	1.00	100	31.9	1.00	100
	4	1.84	3.50	87.5	3.97	3.59	89.6	8.78	3.64	91.0
	8	0.99	6.51	81.4	2.09	6.84	85.5	4.61	6.94	86.6
	12	0.69	9.33	77.8	1.44	9.89	82.4	3.17	10.09	84.1
	16	0.55	11.71	73.2	1.16	12.32	77.0	2.47	12.95	80.9
	20	0.47	13.70	68.5	0.96	14.88	74.4	2.05	15.60	78.0
	24	0.41	15.71	65.5	0.82	17.29	72.0	1.77	18.07	75.3
	28	0.38	16.95	60.5	0.75	19.07	68.1	1.58	20.19	72.1

Table 8, RM scales better than IS with increasing dataset size. On the other hand, for the datasets at hand, the performance of IS seems to be relatively independent of the dataset size. This can be explained with the fact that, in IS, the task decomposition and load balancing is on the IS and hence is not affected much by the dataset size. However, for the cases where data replication overhead is relatively important (e.g., small screen resolution or low network speed), increasing dataset size is supposed to be a bottleneck on scalability of IS.

Another important observation that can be made using Table 8 is about performance variation on dataset regularity, which is determined by the degree of variation among the cell sizes in a dataset. According to Table 8, IS seems to perform better on the regular dataset, which has low cell-size variation. As shown

Table 7
Performance comparison of pixel assignment schemes on the parallel pixel merging (PM) phase

K	Screen sizes		Pixel assignment schemes											
			SA				MCA				BLMCA			
			Mesh sizes				Mesh sizes				Mesh sizes			
			25^2	50^2	75^2	100^2	25^2	50^2	75^2	100^2	25^2	50^2	75^2	100^2
8	400^2	T_{PM}	0.133	0.131	0.134	0.128	0.142	0.142	0.152	0.167	0.140	0.141	0.153	0.165
		T_{PA}	0.000	0.000	0.000	0.000	0.067	0.067	0.076	0.091	0.066	0.067	0.079	0.088
		T_{RSM}	0.126	0.123	0.126	0.120	0.073	0.074	0.074	0.074	0.070	0.069	0.070	0.073
		T_{LPM}	0.008	0.008	0.009	0.009	0.013	0.013	0.013	0.013	0.010	0.010	0.011	0.011
		V_{RSM}	5.769	5.758	5.772	5.743	4.003	3.959	3.948	3.949	4.322	4.259	4.242	4.261
8	900^2	T_{PM}	0.587	0.593	0.591	0.588	0.598	0.606	0.601	0.626	0.594	0.601	0.606	0.619
		T_{PA}	0.000	0.000	0.000	0.000	0.263	0.274	0.269	0.299	0.260	0.262	0.271	0.285
		T_{RSM}	0.554	0.559	0.556	0.551	0.338	0.333	0.334	0.328	0.325	0.329	0.328	0.326
		T_{LPM}	0.040	0.039	0.040	0.040	0.054	0.055	0.055	0.055	0.047	0.048	0.049	0.050
		V_{RSM}	26.64	26.63	26.72	26.63	18.16	17.98	17.92	17.83	19.63	19.50	19.39	19.38
24	400^2	T_{PM}	0.097	0.094	0.104	0.100	0.130	0.133	0.148	0.161	0.121	0.126	0.132	0.151
		T_{PA}	0.000	0.000	0.000	0.000	0.050	0.057	0.069	0.082	0.051	0.059	0.062	0.081
		T_{RSM}	0.092	0.089	0.098	0.094	0.076	0.073	0.075	0.076	0.068	0.066	0.068	0.068
		T_{LPM}	0.007	0.006	0.007	0.006	0.017	0.016	0.017	0.017	0.009	0.008	0.009	0.009
		V_{RSM}	11.17	11.19	11.18	11.21	9.087	8.941	8.913	8.959	9.659	9.454	9.430	9.417
24	900^2	T_{PM}	0.409	0.394	0.408	0.413	0.524	0.511	0.511	0.549	0.482	0.484	0.493	0.524
		T_{PA}	0.000	0.000	0.000	0.000	0.159	0.163	0.165	0.195	0.155	0.159	0.168	0.198
		T_{RSM}	0.384	0.370	0.384	0.388	0.363	0.351	0.348	0.354	0.326	0.325	0.323	0.324
		T_{LPM}	0.034	0.030	0.029	0.029	0.052	0.048	0.049	0.048	0.038	0.034	0.035	0.036
		V_{RSM}	54.30	54.40	54.34	54.43	43.58	43.21	43.04	42.90	46.53	45.62	45.34	45.11

Table 8
Performance comparison of the RM and IS [3] algorithms

Algorithm	Datasets	Screen sizes								
		1200 × 1200			1800 × 1800			2400 × 2400		
		$K = 8$	$K = 16$	$K = 32$	$K = 8$	$K = 16$	$K = 32$	$K = 8$	$K = 16$	$K = 32$
RM	BF	6.49	11.74	19.21	6.54	11.97	19.39	6.57	12.11	20.61
	CC	6.64	12.14	20.18	6.68	12.40	20.93	6.76	12.58	21.37
	OP	7.11	13.12	22.58	7.18	13.44	22.86	7.20	13.48	23.02
IS	BF	6.97	12.73	19.95	7.27	13.75	22.82	7.48	13.75	23.36
	CC	6.99	13.03	22.21	7.42	13.98	25.49	7.55	14.51	26.70
	OP	6.92	12.37	20.83	7.15	13.54	22.98	7.20	13.97	24.17

by the coefficients of variations provided in Table 2, the BF and OP datasets are rather irregular, whereas the CC dataset is a regular one. Performance of RM seems to be independent of the dataset regularity. This is basically because, in RM, the task decomposition and load balancing is on the data space. Further experimental results, which can be found in [3], indicate that high screen resolutions favor the IS algorithm, whereas RM performs better at low screen resolutions.

7. Conclusion

An efficient object space (OS)-parallel direct volume rendering (DVR) algorithm was developed for visualization of unstructured grids on distributed-memory architectures. A fast ray-casting-based DVR algorithm was selected as the underlying sequential algorithm. The adaptive OS decomposition problem was modeled as a graph partitioning (GP) problem to minimize both the amount of redundant computation/storage and volume of interprocessor communication while maintaining the computational load balance. An effective view-independent cell clustering scheme was introduced to induce more tractable, contracted view-dependent computational graphs for successive visualizations. An efficient and highly accurate estimation scheme was proposed for view-dependent node and edge weighting of the coarse computational graphs. A GP-based model was proposed for the solution of the general remapping problem to enhance the adaptive OS decomposition model and make it also consider minimization of the task migration overhead for better performance in successive visualizations. The remapping tool RM-MeTiS was developed by modifying and enhancing the original MeTiS package for partitioning the remapping graphs. Performance of the proposed parallel DVR algorithm was tested on a 28-node PC cluster over three benchmark volumetric datasets.

Acknowledgment

The computational resources used in this work are provided by the TUBITAK ULAKBIM High Performance Computing Center.

Appendix A. Koyamada's DVR algorithm

In the tetrahedral cell model, each cell contains four nodes and four triangular faces. The *face normal* is defined to be oriented outward from the parent cell. If a face of a cell is shared by two cells, that face is called *internal*. Otherwise, it is called *external*. The visualization of a volumetric dataset $\mathcal{V}$ for a given viewing parameter set v is called a *visualization instance* and is denoted by the 2-tuple $(\mathcal{V}, v)$. v consists of the view-direction vector, view-up vector, view-reference point, view-plane window, and screen size. The nodes of $\mathcal{V}$ are initially in the *WCS*, and they are transformed into the *NPCS* by using v. In the NPCS, $\mathcal{V}$ is viewed in the positive z direction, and the (x, y) coordinate values of the nodes of $\mathcal{V}$ are, in fact, their (x, y) coordinate values on the image plane. In a visualization instance $(\mathcal{V}, v)$, a face of a cell of $\mathcal{V}$ is an ff face or a bf face if the z component of the face normal (in the NPCS) is negative or positive, respectively. A ray enters a cell through an ff face and exits the cell through a bf face.

In Koyamada's [26] algorithm, the tetrahedral cell data is stored in two arrays: *VtxArray* and *CellArray*. *VtxArray* keeps the scalar values and the original x, y, z coordinate values (in the WCS) of the nodes of $\mathcal{V}$. It also maintains a view-dependent NPCS component for each node to keep the transformed and projected x, y, z values. *CellArray* stores the identifiers of the four nodes of each cell of $\mathcal{V}$. It also stores the indices of the four neighbors of each cell through its four faces and also identifies each of its internal faces by its index in the respective neighbor cell to avoid the search for finding the entry face of the ray to the next cell during the ray traversal. Moreover, the algorithm uses a *RayBuffer* structure that holds a linked list of composited ray segments for each pixel of the screen. Each item of a linked list stores the composited RGB color values, the composited opacity value, and the exit-point z value of the respective ray segment. The lists are maintained in sorted increasing order according to the exit-point z values.

The first step of Koyamada's algorithm is to generate the ray segments to be traced. In the original algorithm, ff external faces are sorted in increasing order with respect to the z coordinates of their centroids. This, in fact, is an approximate order, which may be wrong in some cases. As the objective in this work is producing high-quality and correct images in a fast way, this step of the original algorithm is slightly modified. Instead of sorting the external faces at the beginning, we scan convert them one by one in any order. For each pixel covered by the projection area of an ff external face, we generate a ray segment, and traverse it through the volume for composition starting from the ff external face until it exits from a bf external face. Finally, we insert the composited ray segment into the respective list in the *RayBuffer* structure, in sorted order, according to its exit z value.

Each ray is followed in the volume utilizing the cell-to-cell connectivity information. To trace a ray inside the volume, a few things have to be known for each cell that is hit by the ray: the entry face, the (z, s) values at the entry point to the cell, the exit face and the (z, s) values at the exit point from the cell. Since the exit-point values from a cell can be used as the entry-point values to the next cell, and the first entry-point values are determined for each ray segment during the scan conversion of ff external faces, the problem of tracing a ray segment reduces to the problem of determining its exit point Q from a cell where its entry point P to the cell is given.

Koyamada proposes a ray–face intersection test that directly determines if the face is intersected by the ray. After identifying the exit face, the (z, s) values (z_Q, s_Q) at the exit point Q are computed through 2D inverse-distance interpolation of the (z, s) values of the nodes of the exit face with respect to point Q. As the exit-point (z_Q, s_Q) values are computed and the entry-point (z_P, s_P) values are already known, the next step is to take samples and composite them along the ray between the entry and exit points P and Q. Our sequential and parallel DVR implementations support both *mid-point* and *equidistant* sampling schemes. In mid-point sampling, a new sample is generated in the middle of the line segment formed by the entry and exit points of the ray intersecting the cell. In equidistant sampling, samples are generated at fixed intervals of length Δz, and hence, for some cells, more than one sample is generated.

Koyamada's algorithm exploits the fact that the change of the scalar in any direction is linear in a tetrahedral cell to speed up interpolation operations through the utilization of *linear sampling* method. That is, scalar value s_X at each sampling point X along line segment PQ is efficiently computed through 1D inverse-distance interpolation of (z_P, s_P) and (z_Q, s_Q) values with respect to point X. Then, scalar value s_X is mapped to a color value C_X and an opacity value O_X by applying a *transfer function* f_{T}, which converts numerical value s_X to color and opacity values to represent the characteristics of the simulation results. Finally, these color and opacity values are composited in front-to-back order as

$$(C_X, O_X) = f_{\mathrm{T}}(s_X), \quad O_{i+1} = O_i + O_X(1 - O_i),$$
$$C_{i+1} = (C_i O_i + C_X O_X(1 - O_i))/O_{i+1},$$

where (C_i, O_i) and (C_{i+1}, O_{i+1}) values are the composited (color, opacity) values before and after processing the sampling point X, respectively, and C_0 and O_0 are initially set to zero [14].

At the end of this process, the *RayBuffer* structure contains all composited ray segments. Lengths of the individual ray lists in the *RayBuffer* structure depend on the visualization instance. In convex datasets, the length of each ray list structure is either 0 or 1. Hence, the color field of the single ray segment of each active pixel contains the final color of the respective pixel of the image. However, in non-convex datasets, a ray may enter and exit the volume multiple times, thus generating multiple ray segments. For example, the ray shown in Fig. 1 generates two ray segments. Hence, non-convex datasets necessitate a final composition process over the ray segment lists of multiple length.

The good features of Koyamada's DVR algorithm are as follows. It exploits the OS coherency during ray-segment traversal through connectivity information. It performs two ray–face intersection tests per cell on the average, whereas the conventional approach [14] always checks three faces. It performs 1D inverse-distance interpolation for each sampling and 2D inverse-distance interpolation for each ray–cell intersection, whereas the conventional approach performs expensive 3D inverse-distance interpolation for each sampling. Moreover, it uses the results of the ray–face intersections to reduce the cost of 2D interpolation operations.

Appendix B. Extensions for close-up visualization instances

The validity of the node and edge weighting scheme proposed in Section 2.3 depends on the assumption that the whole volume is visible on the screen. However, after a number of visualizations, the simulation scientist may want to see some details about a particular area of the volume from different viewing directions. He may select a window on the view plane such that only a particular area of the volume is visible through the screen. In such close-up visualization instances, some cells and possibly some cell clusters will become invisible and hence will not incur any rendering computations. Here, we propose an extension of our node and edge weighting scheme to handle close-up visualization instances.

For a given v, each cluster of $\mathcal{V}$ is classified as *off-screen*, *on-screen*, or *partially on-screen* through a preculling operation if its bounding box is totally outside, totally inside, or partially inside the viewing frustum, respectively. The nodes corresponding to the off-screen clusters are deleted from $\mathcal{G}_C^v$ together with all edges incident to those nodes. The weights of the nodes corresponding to the on-screen clusters are computed as described in Section 2.3.1. The node weight estimation heuristic is modified for the partially on-screen clusters as follows. Consider a node n_i of $\mathcal{G}_C^v$ corresponding to a partially on-screen cluster $\mathcal{C}_i$ of $\mathcal{V}$. After the node transformation phase, each face (and each cell in equidistant sampling) of $\mathcal{C}_i$ is classified as off-screen, on-screen, or partially visible if the rectangular box bounding of its projection area on the image plane is totally outside, totally inside, or partially inside the screen boundaries, respectively. The off-screen bf faces of $\mathcal{C}_i$ do not contribute to the ray–face intersection count (I_i^v) of $\mathcal{C}_i$ (Eq. (1)). Each on-screen bf face f of $\mathcal{C}_i$ contributes to I_i^v by its projection area estimate a_f, computed according to Eq. (2) as described in Section 2.3.1. Each partially on-screen bf face f of $\mathcal{C}_i$ contributes to I_i^v by $\beta_f^v \times a_f$, where β_f^v denotes the visibility ratio of the screen-space bounding box of face f. Recall that the sampling count (S_i^v) of $\mathcal{C}_i$ is equal to I_i^v in mid-point sampling. In equidistant sampling, the constant-time scheme proposed for estimating S_i^v cannot be used for a partially on-screen cluster $\mathcal{C}_i$. Instead, the volumes of all cells in the WCS are pre-computed once during the view-independent preprocessing phase. The volume of each partially on-screen cell multiplied by the visibility ratio β^v of its screen-space bounding box and the volume of each on-screen cell are added to obtain an approximation to the total visible volume of $\mathcal{C}_i$ in the WCS. Then, multiplying the result by $\gamma^v/\Delta z$ gives an estimate for S_i^v as mentioned in Section 2.3.1.

In edge weighting, the weight of each edge between a pair of on-screen clusters is computed as described in Section 2.3.2. Each edge incident to an at least one off-screen cluster is already deleted as mentioned above. The weight $w(n_i, n_j)$ of each edge e_{ij} between a pair of partially on-screen clusters (a pair of on-screen and partially on-screen clusters) is computed as follows. Off-screen faces shared between $\mathcal{C}_i$ and $\mathcal{C}_j$ do not contribute to $w(n_i, n_j)$. Each on-screen and partially on-screen face f shared between $\mathcal{C}_i$ and $\mathcal{C}_j$ contribute to $w(n_i, n_j)$ by a_f and $\beta_f^v \times a_f$, respectively.

References

[1] H. Berk, C. Aykanat, U. Güdükbay, Direct volume rendering of unstructured grids, Comput. Graphics 27 (3) (2003) 387–406.

[2] G. Burns, R. Daoud, J. Vaigl, LAM: an open cluster environment for MPI, in: Proceedings of the Supercomputing Symposium '94, 1994, pp. 379–386.

[3] B.B. Cambazoglu, C. Aykanat, Hypergraph-partitioning-based remapping models for image-space-parallel direct volume rendering of unstructured grids, IEEE Trans. Parallel Distributed Systems, in press.

[4] W. Camp, S. Plimpton, B. Hendrickson, R. Leland, Massively parallel methods for engineering and science problems, Comm. ACM 37 (1994) 31–41.

[5] J. Challinger, Parallel volume rendering for curvilinear volumes, in: Proceedings of the IEEE Scalable High Performance Computing Conference, April 1992, pp. 14–21.

[6] J. Challinger, Scalable parallel volume raycasting for nonrectilinear computational grids, in: Proceedings of the IEEE/ACM Parallel Rendering Symposium '93, October 1993, pp. 81–88.

[7] J. Challinger, Scalable parallel direct volume rendering for nonrectilinear computational grids, Ph.D. Thesis, University of California, 1993.

[8] J. Comba, J.T. Klosowski, N. Max, J.S.B. Mitchell, C.T. Silva, P.L. Williams, Fast polyhedral cell sorting for interactive rendering of unstructured grids, Comput. Graphics Forum 18 (3) (1999) 369–376.

[9] J. Eyles, S. Molnar, J. Poulton, T. Greer, A. Lastra, N. England, L. Westover, PixelFlow: the realization, in: Proceedings of the Siggraph/Eurographics Workshop on Graphics Hardware, August 1997, pp. 57–68.

[10] R. Farias, J. Mitchell, C.T. Silva, ZSWEEP: an efficient and exact projection algorithm for unstructured volume rendering, in: ACM/IEEE Volume Visualization and Graphics Symposium, October 2000, pp. 91–99.

[11] R. Farias, C.T. Silva, Out-of-core rendering of large unstructured grids, IEEE Comput. Graphics Appl. 21 (4) (2001) 42–50.

[12] R. Farias, C.T. Silva, Parallelizing the ZSWEEP algorithm for distributed-shared memory architectures, in: International Volume Graphics Workshop, 2001, pp. 181–192.

[13] C.M. Fiduccia, R.M. Mattheyses, A linear-time heuristic for improving network partitions, in: Proceedings of the 19th ACM/IEEE Design Automation Conference, 1982, pp. 175–181.

[14] M.P. Garrity, Raytracing irregular volume data, IEEE Comput. Graphics Appl. 24 (5) (1990) 35–40.

[15] C. Giertsen, Volume visualization of sparse irregular meshes, IEEE Comput. Graphics Appl. 12 (2) (1992) 40–48.

[16] A. Grama, A. Gupta, G. Karypis, V. Kumar, Introduction to Parallel Computing, Design and Analysis of Algorithms, second ed., Addison-Wesley Publishing Company, Harlow, 2003.

[17] S. Guthe, S. Roettger, A. Schieber, W. Strasser, T. Ertl, High-quality unstructured volume rendering on the PC platform, in: Proceedings of the Siggraph/Eurographics Workshop on Graphics Hardware, 2002, pp. 119–125.

[18] B. Hendrickson, R. Leland, The Chaco user's guide: version 2.0, Technical Report, SAND94-2692, Sandia National Laboratories, 1994.

[19] C. Hofsetz, K.-L. Ma, Multi-threaded rendering unstructured-grid volume data on the SGI Origin 2000, in: Proceedings of the Third Eurographics Workshop on Parallel Graphics and Visualization, September 2000.

[20] E. Horowitz, S. Sahni, Fundamentals of Computer Algorithms, Computer Science Press, Rockville, MD, 1978.

[21] G. Karypis, V. Kumar, Multilevel *k*-way partitioning scheme for irregular graphs, J. Parallel Distributed Comput. 48 (1) (1998) 96–129.

[22] G. Karypis, V. Kumar, A fast and high quality multilevel scheme for partitioning irregular graphs, SIAM J. Sci. Comput. 20 (1) (1998) 359–392.

[23] G. Karypis, V. Kumar, MeTiS: a software package for partitioning unstructured graphs, partitioning meshes and computing fill-reducing orderings of sparse matrices, Technical Report, Department of Computer Science and Engineering, University of Minnesota, 1998.

[24] B.W. Kernighan, S. Lin, An efficient heuristic procedure for partitioning graphs, Bell System Tech. J. 49 (1970) 291–307.

[25] Y. Kopidakis, M. Lamari, V. Zissimopoulos, On the task assignment problem: two new efficient heuristic algorithms, J. Parallel Distributed Comput. 42 (1) (1997) 21–29.

[26] K. Koyamada, Fast traversal of irregular volumes, in: T.L. Kunii (Ed.), Visual Computing, Integrating Computer Graphics with Computer Vision, Springer, New York, 1992, pp. 295–312.

[27] K. Koyamada, T. Nishio, Volume visualization of 3d FEM results, IBM J. Res. Develop. 35 (1/2) (1991) 12–25.

[28] H. Kutluca, T. Kurc, C. Aykanat, Image-space decomposition algorithms for sort-first parallel volume rendering of unstructured grids, J. Supercomput. 15 (1) (2000) 51–93.

[29] W.-S. Lin, R.W.H. Lau, K. Hwang, X. Lin, P.Y.S. Cheung, Adaptive parallel rendering on multiprocessors and workstation clusters, IEEE Trans. Parallel Distributed Systems 12 (3) (2001) 241–258.

[30] V.M. Lo, Heuristic algorithms for task assignment in distributed systems, IEEE Trans. Parallel Distributed Systems 37 (11) (1988) 1384–1397.

[31] K.-L. Ma, Parallel volume ray-casting for unstructured-grid data on distributed-memory architectures, in: Proceedings of the IEEE/ACM Parallel Rendering Symposium '95, 1995, pp. 23–30.

[32] K.-L. Ma, T.W. Crockett, A scalable cell-projection volume rendering algorithm for unstructured data, in: Proceedings of the IEEE/ACM Parallel Rendering Symposium '97, 1997, pp. 95–104.

[33] K.-L. Ma, S. Parker, Massively parallel software rendering for visualizing large-scale data sets, IEEE Comput. Graphics Appl. 21 (4) (2001) 72–83.

[34] K. Moreland, B. Wylie, C. Pavlakos, Sort-last parallel rendering for viewing extremely large data sets on tile displays, in: Proceedings of the IEEE 2001 Symposium on Parallel and Large-Data Visualization and Graphics, October 2001, pp. 85–92.

[35] NASA dataset archive. ⟨http://www.nas.nasa.gov/Research/Datasets/datasets.html⟩.

[36] L. Oliker, R. Biswas, PLUM: parallel load balancing for adaptive unstructured meshes, J. Parallel Distributed Comput. 52 (2) (1998) 150–177.

[37] C.-W. Ou, S. Ranka, Parallel incremental graph partitioning, IEEE Trans. Parallel Distributed Systems 8 (8) (1997) 884–896.

[38] M.E. Palmer, S. Taylor, Rotation invariant partitioning for concurrent scientific visualization, in: Parallel Computational Fluid Dynamics '94, 1994.

[39] R. Samanta, T. Funkhouser, K. Lai, Parallel rendering with *k*-way replication, in: Proceedings of the IEEE Symposium on Parallel Graphics, October 2001, pp. 75–84.

[40] R. Samanta, T. Funkhouser, K. Lai, J.P. Singh, Hybrid sort-first and sort-last parallel rendering with a cluster of PCs, in: Proceedings of the Siggraph/Eurographics Workshop on Graphics Hardware, August 2000, pp. 97–108.

[41] K. Schloegel, G. Karypis, V. Kumar, Multilevel diffusion schemes for repartitioning of adaptive meshes, J. Parallel Distributed Comput. 47 (2) (1997) 109–124.

[42] K. Schloegel, G. Karypis, V. Kumar, Wavefront diffusion and LMSR: algorithms for dynamic repartitioning of adaptive meshes, IEEE Trans. Parallel Distributed Systems 12 (5) (2001) 451–466.

[43] P. Shirley, A. Tuchman, A polygonal approximation to direct scalar volume rendering, IEEE Comput. Graphics Appl. 24 (5) (1990) 63–70.

[44] C.T. Silva, J.S.B. Mitchell, The lazy sweep ray casting algorithm for rendering irregular grids, IEEE Trans. Visualization Comput. Graphics 3 (2) (1997) 142–157.

[45] H.S. Stone, Multiprocessor scheduling with the aid of network flow algorithms, IEEE Trans. Software Eng. 3 (1) (1977) 85–93.

[46] C. Walshaw, M. Cross, M.G. Everett, Parallel dynamic graph partitioning for adaptive unstructured meshes, J. Parallel Distributed Comput. 47 (2) (1997) 102–108.

[47] J. Wilhelms, A. Van Gelder, A coherent projection approach for direct volume rendering, IEEE Comput. Graphics Appl. 25 (4) (1991) 275–284.

[48] J. Wilhelms, A. Van Gelder, P. Tarantino, J. Gibbs, Hierarchical and parallelizable direct volume rendering for irregular and multiple grids, in: Proceedings of the Visualization '96, 1996, pp. 57–64.

[49] P.L. Williams, Interactive direct volume rendering of curvilinear and unstructured data, Ph.D. Thesis, University of Illinois at Urbana-Champaign, 1992.

[50] P.L. Williams, N.L. Max, C.M. Stein, A high accuracy volume renderer for unstructured data, IEEE Trans. Visualization Comput. Graphics 4 (1) (1998) 37–54.

[51] C.M. Wittenbrink, Irregular grid volume rendering with composition networks, in: Proceedings of SPIE Visual Data Exploration and Analysis V, February 1998, pp. 284–294.

[52] C.M. Wittenbrink, Survey of parallel volume rendering algorithms, in: Proceedings of the PDPTA'98 Parallel and Distributed Processing Techniques and Applications, July 1998, pp. 1329–1336.

[53] R. Yagel, D. Reed, A. Law, P.-W. Shih, N. Shareef, Hardware assisted volume rendering of unstructured grids by incremental slicing, in: Proceedings of the 1996 Symposium on Volume Visualization, November 1996, pp. 55–62.

Cevdet Aykanat received the BS and MS degrees from Middle East Technical University, Ankara, Turkey, both in electrical engineering, and the PhD degree from 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 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.

Berkant Barla Cambazoglu graduated from Bursa Erkek Lisesi. He received his 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.

Ferit Findik received his BS (1995) and MS (1997) degrees in Computer Engineering from Bilkent University, Ankara, Turkey. He currently works at Microsoft Corp. at Redmond as a software design engineer. His area of expertise is commercial software design and development in systems and operations management.

Tahsin Kurc is an Assistant Professor in the Department of Biomedical Informatics at the Ohio State University. His research interests include runtime systems for data-intensive computing in parallel and distributed environments, and scientific visualization on parallel computers. He received his PhD in computer science from Bilkent University, Turkey, in 1997 and his BS in electrical and electronics engineering from Middle East Technical University, Turkey, in 1989.