計算機アーキテクチャ 132−12
システムソフトウェアと
オペレーティング・システム 80−12
ハイパフォーマンス
コンピューティング 75−12
(1999. 3. 4)

並列ボリュームレンダリングにおける静的負荷分散

鈴木 芳生　長澤 幹夫
email:yosuzuki@crl.hitachi.co.jp,　m-nagasa@crl.hitachi.co.jp
(株) 日立製作所 中央研究所

並列ボリュームレンダリング高速化のために、前処理過程でのデータ分割最適化による静的負荷分散を行った。データ分布に応じて適応的に画面およびデータの分割パタンを決定する手法を考案し、超並列計算機 Hitachi SR2201 上に並列レンダリングシステムとして実装した。台風シミュレーションデータの可視化に適用し、良好な負荷分散が得られることを確認した。システムは AVS モジュールとして実装され、分散環境下での数値実験と可視化のリアルタイム連携が可能となっている。

Static Load Balancing for Parallel Volume Rendering

Yoshio Suzuki and Mikio Nagasawa
Central Research Laboratory, Hitachi Ltd.

For the parallel volume rendering system, we present a new data-decomposition method for the optimal static load balancing. In our method, volumetric and pixel data are divided adaptively according to the data distribution structure itself. We have implemented this adaptive parallel decomposition algorithm on Hitachi SR2201, MPP system with hypercrossbar interconnection, using the message-passing libraries. By testing the performance of the new method in volume visualization of typhoon simulation data, we confirmed the system can obtain a good load balance with relatively small cost of communication overhead among parallel processors.

1 Introduction

With the development of massive parallel computers, numerical simulations are being done in three spatial dimensions with the increase of resolution and complexity. Consequently, the output data size becomes larger. Because of being difficult to understand such a raw data intuitively, it is important for scientific visualization to render the whole volume data. The resulting image enables us to understand the global nature of the data and to analyze the simulated phenomena.

In the usual scientific simulations on supercomputers or MPP, the simulation outputs are reduced and sent to the post-processing graphic workstations(GWS) for visualization. The graphic accelerators are applicable for the surface visualization such as a 3-D isosurface rendering. When the volume data is too big for GWS, however, we had better visualize the data concurrently with the simulation itself on the same MPP. This is suitable especially for the time-varying data as in the tracking and steering of simulation. We could cut the time of volumetric I/O transmission from MPP to GWS, and speed up the rendering itself by using the parallel processors on MPP.

It is well known that the volume rendering is suitable for the direct visualization of volumetric data. At the same time, it is known as very computationally intensive because of the large number of ray integrations through the voxels. This indicates the inherent parallelism in volume rendering. We investigate several parallel algorithms in this paper. In the image space decomposition, we divide the pixel data into some rectangle subimages. These pixel blocks are ray-integrated in parallel. Moreover, we divide also the volumetric data space in addition to the image space decomposition. This is due to the following two reasons: First, the typical 3D simulation data is too large for a single processor to posses all the data in its local memory. Secondly, in the volume rendering, the processing time is roughly proportional to the data size. Therefore, also the parallel processing time will be balanced among processors if we could divide the data evenly. The latter is our main idea to obtain a good load balance in parallel volume rendering by dividing the data at pre-processing steps.

The simple data decomposition method such as a uniform voxel slicing, however, is not able to obtain a good load balance. The data distribution of interest is often concentrated in the simulation of isolated systems. The dependency on the view direction is not constant either. Therefore, we propose new data decomposition methods that divide data according to the data structure itself and change the decomposition pattern adaptively with the data distribution. By these adaptive methods, the rendering volume data are divided to realize the uniform load balance among processors for every data distribution.

2 Related Work

There have been numerous approaches to speed up the rendering using parallel algorithm [1, 2, 3, 4, 5, ?]. Many of them were based on the image space decomposition method, in which sub-images are rendered in parallel after the allocation of divided scanlines or pixel blocks. In large applications, the volumetric data is also divided because of the insufficient local memory of parallel processors. The simple data decomposition method were proposed to divide the data space uniformly. Without the information of the volume data, however, such a naive decomposition method needs a lot of communications between processors, or causes total load imbalances among processors.

In the effective parallel computing, a good load balance is so important that many balancing techniques are proposed. The load balancing methods are classified into two categories: static load balancing and dynamical load balancing. In the fine-grained static load balancing, the data is divided into very small pieces. The number of sub-volume is larger than that of parallel processors. It is proposed to allocate the sub-volumes to processors in a cyclic way.

As the dynamical load balancing, Whitman [5] proposed the task adaptive method in image space decomposition. The processor that has done its task fetches another task from the processor that has the most task still unfinished. The decomposition entity could be much smaller as scanlines in order to get a good load balance by increasing the number of task tradings between processors.

3 Parallel Volume Rendering

3.1 Ray integration scheme

The volume rendering [6] is the direct rendering of scalar field data sampled in three dimensions. There are two ways of sampling the volume elements or voxels. One is the forward mapping in voxel space, and the other is the backward mapping in image space. The forward mapping algorithm is a kind of projection that the individual voxels emit the light to the image screen. While, the backward mapping algorithm is a kind of ray-tracing. The rays casted from the image plane go through the volume data and the opacity sampling are integrated along the ray axis. Our volume rendering belongs to the forward mapping algorithm based on the voxel space sampling with the smooth Gaussian filters [7].

In the ray integration, the color emissivity and opacity are calculated at each voxel point. The color is assigned by the Phong's shading model [8]. Then the color is accumulated by alpha blending scheme,

$$C_{i+1} = \alpha_s c_s + (1 - \alpha_s)C_i \qquad (1)$$

where, C_i is the accumulated color of pixel, c_s and α_s is the color and the opacity at the sampling point, respectively.

3.2 Parallel algorithm

We parallelize the volume rendering by both image and data space decomposition. As for the image space decomposition, each sub-image is generated independently without communication among parallel processors during the ray integration.

Our parallel rendering system consists of one host process and a group of parallel node processes. Each process is executed at the processor elements of MPP. The single host processor receives some informations, such as rendering parameters or viewing angles, from a front-end computer. Then the host processor sends the information to the node processors and wait for all the parallel nodes to finish sub-rendering. Finally, the host processor receives sub-images from the node processors, and reconstructs the complete image to display with those sub-images.

In the parallel rendering step, the node processor actually renders the voxel subspace. Each node processor does the ray integration to generate the sub-image, and sends the output sub-image to the host processor. For the load balancing, each node processor divides and extracts the data enough for generating the responsible sub-image. We will explain in detail the data decomposition methods in the next section.

Since the data is divided along z-axis for every sub-image, we can generate the result image by ray integration from back to front order according to the alpha-blending principle.

4 Data Decomposition for Static Load Balancing

In our volume rendering, the image is generated by projecting the sampling points in the data space onto screen. The processing time is roughly proportional to the data size sustained by each processor. Therefore, to divide the data space into sub-regions of equal size would result in a good load balance. There are many other data decomposition methods proposed [1, 2, 3]. The data space is divided along each axis in such decomposition methods. Hsu [1] divided the data space uniformly in each axis, that is, data space is divided into same-sized region. However if the data doesn't distribute uniformly in volume space as often the case, such decomposition method cannot get a good load balance.

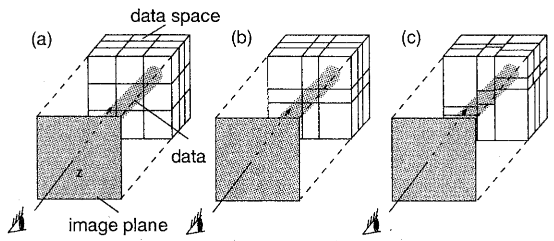

Figure 1: Decomposition method.(a)uniform decomposition, (b)rectilinear decomposition, and (c)structured decomposition.

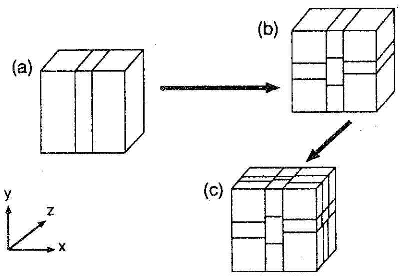

Figure 2: Structured data-array decomposition.(a)x-axis partition,(b)y-axis partition,and (c)z-axis partition.

In numerical simulations, there are many data types, FEM lattice data, particle data, FDM mesh, *etc.* Using the method which has the fixed decomposition pattern for every input data, we cannot expect a good load balance because the optimal decomposition is different for each simulation data set. The method that can change the decomposition pattern adaptively for input data distribution is really needed.

For the performance investigation, we have implemented four decomposition methods in our parallel rendering system. We define the projection space "cube" as its x-y plane parallel to the viewing screen of data-space as in Fig.1. The data is distributed in this "cube" space. The slicing pattern in the screen space is same with that of data space along x,y-axis.

4.1 Uniform decomposition

In uniform decomposition, the data space is divided with equal interval along each axis as in Fig.1(a). Even if the distribution of data is changed, the pattern of decomposition remains unchanged in uniform decomposition.

4.2 Rectilinear decomposition

It is often true that the data distributes around the center of screen. Thus, in rectilinear decomposition as in Fig.1(b), the data space is divided with smaller spacing around the center of screen. But using rectilinear decomposition, we cannot get good enough load balance for some kind of simulation with periodic boundary conditions. That has a complex data distribution as is often the case of homogeneous 3-D turbulence.

4.3 Structured decomposition

In uniform decomposition and rectilinear decomposition, the pattern of decomposition is fixed for every data distribution. Then we cannot obtain good load balance for the changing distribution because the divided data-size is not equal among processors.

Our goal is to divide data equally in number by changing the pattern of decomposition adaptively for every data. In uniform decomposition and rectilinear decomposition, the data space is divided with the projection space "cube", but the data is sliced with data-distribution structure itself in the structured decomposition as in Fig.1(c).

The sampling points have coordinates(x,y,z) and other simulation quantity. These are represented as an array of 3D-position coordinate. The data can be divided equally in number by dividing the data array evenly. However, spatial relation may be disrupted by such a decomposition because adjacent sampling points in data space may not always be stored adjacently in data array. The sampling points, located in neighborhood in data space, happen to be divided into different sub-arrays. In the ray integrations, the neighbor sampling points in data space will contribute the same or adjacent pixels. When such data is separated into different processors, each processor must communicate to get the other information afterward.

In the structured decomposition, we do sort the sampling points first. By this sorting, the order in array is made to be same with the order of data-value such as an order along x-axis in data space. Then, by dividing the data array evenly, we can divide the data into same sized sub-arrays without disrupting spatial relations in data space. We accelerate sorting by using "qsort" function.

In Fig.2, we depict structured decomposition. First, the data array is divided into n_x sub-arrays after being sorted along x-axis. Then each sub-array is divided into n_y sub-regions after being sorted along y-axis. Finally the data array is divided along z-axis in the same way.

The structured decomposition makes possible to divide the data into equally sized sub-set by dividing the sampling points according to the data distribution structure itself. As a result of sorting, the data is divided without disrupting spatial relations both in the data space and in the distributed memory space of MPP.

4.4 Hybrid decomposition

The sorting requires intensive computational cost. In fact, the computational cost of sorting reaches $O(N\log N)$ in the worst case where N is the total number of sampling points. The first sorting step in structured decomposition is as heavy as $O(N\log N)$, although the cost of second and third sorting after decomposition along x,y-axis reaches $O(\frac{N}{nx}\log\frac{N}{nx})$ and $O(\frac{N}{nxny}\log\frac{N}{nxny})$ respectively. Thus we could replace the first sorting step of structured decomposition with rectilinear decomposition as a hybrid decomposition approach. If the data is distributed locally around the center of data space, hybrid decomposition enables data slicing faster than structured decomposition, and therefore the better load balance than that of rectilinear decomposition is achieved. If the data has the characteristic distribution of homogeneous dispersion in space, the first step of rectilinear slicing of hybrid decomposition cannot achieve good load balance. Because of the inefficiency of the first step of rectilinear decomposition, the data may not be divided uniformly among processors in hybrid decomposition.

In each decomposition method, each processor possesses not only the data in its responsible pixel region but also the data in adjacent pixel region because the data in adjacent pixel region could contribute to render the sub-image as well. By this overlapping configuration, the additional communication could be greatly reduced between processors.

5 Results

We have implemented our parallel volume rendering algorithm to our MPP(Hitachi SR2201) system with the hypercrossbar interconnection of 64 processor elements. We used PARALLELWARE(EXPRESS) [9] for the communication between processors. We developed our system as AVS [10] modules for the cooperation with the simulation solver modules. On this software platform of AVS, concurrent execution of simulation and visualization are realized in the open system with MPP and GWS. And the system is developing to be capable of a real-time tracking and steering of large scale simulation.

We visualized typhoon simulation data as a test of our parallel volume rendering system. The data size of simulation is $135 \times 111 \times 14$ and the pixel resolution is 592×444.

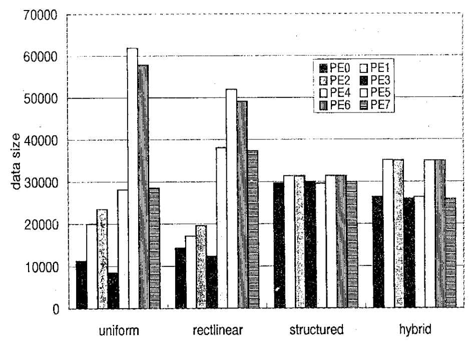

Figure 3: Memory load balance in number of sampling points among processor elements(PE).

5.1 Memory balance

In Fig.3, we compare the allocated data size of each node processor, for each decomposition method: "uniform" decomposition, "rectilinear" decomposition, "structured" decomposition, and "hybrid" decomposition with eight node processors. The rendering process consists of a host process and eight node processes. There is large imbalance in "uniform" decomposition and "rectilinear" decomposition. Some node processors have extremely large amount of data and others have little data in these decomposition methods. On the other hand, the data is divided uniformly in "structured" decomposition and "hybrid" decomposition. Since the data is distributed around the center of image in this test data, better load balance is achieved by using "rectilinear" decomposition than "uniform" decomposition, and we find that there is little imbalance in "hybrid" decomposition.

5.2 Power balance

In Fig.4, we compare the measured processing time of each node processor using "uniform" decomposition, "rectilinear" decomposition, "structured" decomposition, and "hybrid" decomposition with eight node processors. The rendering process consists of a host process and eight node processes. There is imbalance in "uniform" decomposition and "rectilinear" decomposition. Some node processors that have little data can generate sub-image faster, but other node processors that have much data need larger processing time. On the other hand, good load balance is achieved in "structured" decomposition and "hybrid" decomposition. The processing time of each node processors is almost equal. Considering both the data size after decomposition and the processing time of each nodes, we confirmed that "structured" and "hybrid" decomposition are suitable for good load balance.

5.3 Scalability

In Fig.5, we compare the speedup in parallel rendering of "uniform" decomposition, "rectilinear" decomposition, "structured" decomposition and "hybrid" decomposition. We tested the speedup for the case of 3 to 41 processors. As a result, we find little difference due to the variety of decomposition in a few processor cases. This is because the decomposition patterns are almost same with few processors. While in the case of large parallelism, "uniform" and "rectilinear" decomposition's performance is in decline, because the large difference between the optimal decomposition pattern and their fixed decomposition. A good performance is achieved in "hybrid" decomposition in large parallelism with many processors. Because we can shortcut the time-consuming sorting step calculation by "rectilinear" decomposition, the processing time of decomposition in "hybrid" decomposition is greatly improved. In the parallel rendering of hybrid decomposition with 41 processors, the processing time is 8.81 sec, that is, 21.5 times speedup is achieved.

5.4 Pattern of decomposition

We compare also the actual pattern of decomposition from different view direction using hybrid decomposition method in Fig.6,6. Because the data distribution is changed according to the view direction, the good load balance cannot be achieved by the method that has the fixed decomposition pattern as "uniform" and"rectilinear" decomposition.

In "hybrid" decomposition the area of sub-region is not equally sliced as in Figs.6,6 because the data is divided according to the data distribution structure itself. After changing the view direction, decomposition pattern and area of each sub-image is also changed adaptively so as that the data size becomes equal among parallel processors. In "hybrid" decomposition, the data is divide by rectilinear slicing at first step of decomposition. Thus the slicing pattern of "rectilinear" and "hybrid"decomposition along x-axis is equal. Similarly, because the decomposition algorithm after rectilinear slicing is same with "structured" decomposition, the slicing pattern along y-axis is similar to that of "structured" decomposition.

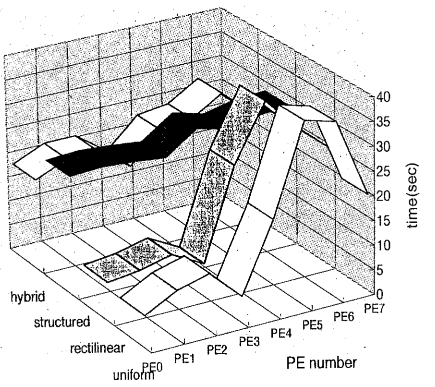

Figure 4: Load balance abong processor elements(PE) in processing time. The processing time of each process is plotted for each decomposition method.

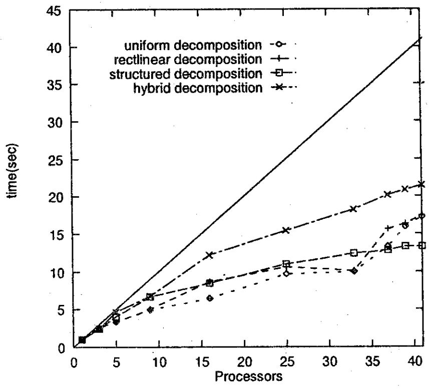

Figure 5: Speedup effect in parallel volume rendering. Saturation points is highest in "hybrid" decomposition.

6 Conclusion and Future Work

We developed a parallel volume rendering system for massively parallel processors. In our algorithm, the image-space is divided for parallelization and the volumetric data is also divided for load balancing. By slicing the data adaptively in pre-processing step, we realize optimal static load balancing for parallel volume rendering.

We investigated several decomposition methods for static load balancing. In "rectilinear" decomposition, the data space is divided with smaller spacing around the center of screen. In "structured" decomposition, the data array is divided according to the data structure itself not to the projection space. By changing the decomposition pattern adaptively, the data is always divided uniformly for every case from any view direction in "structured" decomposition. When the first step of sorting in "structured decomposition" happens

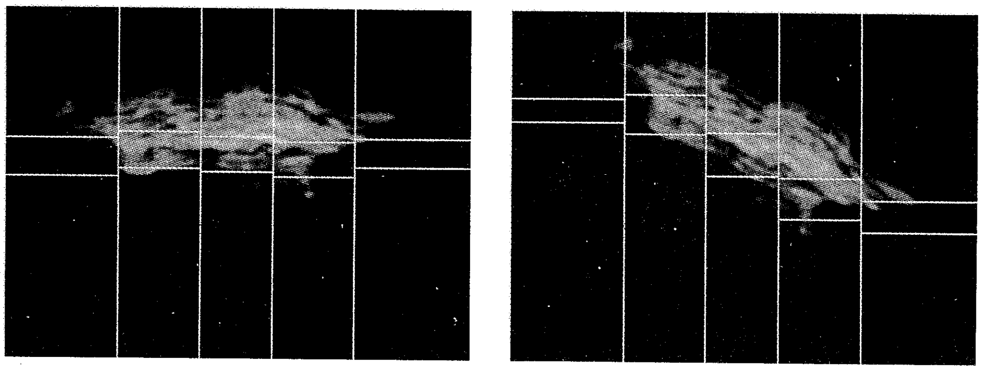
Figure 6: Hybrid slicing patterns with different view angles.

to be computationally intensive, we can improve its performance by replacing the sorting step with rectilinear slicing in "hybrid" decomposition.

We have implemented our system on Hitachi SR2201, MPP system with hypercrossbar interconnection, using the message passing libraries. By testing the performance in volume visualization of typhoon simulation data, we confirmed that our data decomposition method is effective for parallel volume rendering. We achieved about 21.5 times speedup using 41 processors. We developed our system as AVS-module in conjunction with CFD simulation solver for the concurrent execution of simulation and visualization.

Our goal is the real-time tracking and steering of scientific simulation. Thus we need further speed up of rendering. For this goal, we are going to refine our algorithm and to implement on the new more powerful Technical server(Hitachi SR8000) which has pseudo-vector processors connected with 3D hyper-crossbar network and has "co-operative micro-processors in single address space" feature.

Acknowledgment

The research results were achieved on the joint research of CRIEPI(Central Research Institute of Electric Power Industry) and Hitachi Ltd. The experiments were performed using Hitachi SR2201 at CRIEPI. The data set of typhoon simulation was provided courtesy of H. Hirakuchi and J. Tsutsui at CRIEPI and A. Kasahara at NCAR.

References

[1] W.M.Hsu."Segmented Ray Casting for Data Parallel Volume Rendering", Proc. of Parallel Rendering Symposium, pp7-14, 1993.

[2] K.-L.Ma, J.S.Painter, C.D.Hansen, M.F.Krogh, "Parallel Volume Rendering Using Binary-Swap Compositing", IEEE CG&A Vol.14, No.4, pp59-68, 1994.

[3] U.Neuman, "Parallel Volume-Rendering Algorithm Performance on Mesh-Connected Multicomputers", Proc. of Parallel Rendering Symposium, pp.97-104, 1993.

[4] B.Corrie, P.Mackerras,"Parallel Volume Rendering and Data Coherence", Proc. of Parallel Rendering Symposium, pp23-26, 1993.

[5] S.Whitman:"Dynamic Load Balancing for Parallel Polygon Rendering", IEEE CG&A Vol.14, No.4, pp41-48, 1994.

[6] Mark Levoy."Display of surface from Volume Data", IEEE CG & A, Vol.8 No.5, pp.29-37,1988.

[7] M.Nagasawa, K.Kuwahara,"Scientific Visualization of Physical Phenomena", (ed. N.M.Patrilkalkis,Springer-Verlag Tokyo) ,pp589-605, 1991.

[8] B.T.Phong,"Illumination for computer generated images",C.ACM,vol.13,No.6,pp.311-317,1975.

[9] Hitachi Ltd.,"PARALLELWARE-C-USER'S GUIDE", Hitachi Ltd., 1995.

[10] Advanced Visual System Inc.,"AVS USER'S GUIDE", Release 5, Advanced Visual System Inc., 1993.