# Enhancing the Performance of a BSP Model-Based Parallel Volume Renderer with a Pofile Visualiser

Hong Xie

Department of Information Technology
Murdoch University,
Murdoch, WA 6150,
Australia
H.Xie@murdoch.edu.au

Chun Che Fung

School of Electrical and Computer Engineering
Curtin University of Technology
Bentley, W.A.
Australia
TFUNGCC@cc.curtin.edu.au

**Abstract**: Volume rendering is computation intensive and can benefit from the combined processing power of a workstation cluster. Carefully balancing the workload among all workstations in the cluster is critical in achieving high efficiency in parallel volume rendering. In this paper we describe how the load balance of a BSP model-based parallel volume render program can be improved substantially using a profile visualiser. The profile visualiser distinguishes between the two types of load imbalance: those caused by the poor design of the parallel volume renderer and those caused by the sharing of the workstations by the other users. This information allows us to concentrate on improving the performance of the parallel program by designing a better load balance strategy for the parallel volume render.

**Keywords**

Volume rendering, BSP, visualisation, load balance.

## I. PARALLEL VOLUM RENDERING

In medical imaging such as Computerised Tomography (CT), Magnetic Resonance Imaging (MRI), Positron Emission Tomography (PET) and Single Photon Emission Computed Tomography (SPECT), *volume rendering* plays a crucial role in the analysis and visualisation of the 3D images [1,2,3]. In contrast to surface rendering, volume rendering directly transforms the density information of a volume into grey or colour intensity in the 2D viewing plane without the need of detecting, formatting or modelling the object surfaces. Volume rendering avoids the difficult segmentation process and the possible artifacts that will be introduced. It also provides a better mechanism for displaying weak or fuzzy surfaces and internal structures.

Volume rendering, however, involves huge amount of data and is extremely computation intensive. For example, a typical volume of 128×256×256 voxels contains 16 M bytes of raw data. If the surface gradients are kept, it can take up to 128 M bytes of memory. To generate a single image from such a volume requires billions of floating point operations. This can take up a few minutes to hours of computing time, depending on the performance of the machine, the rendering algorithm used, and the image quality required. Obviously, this is unacceptable for real-time and interactive applications. This has provided the motivation for many research efforts into the development of parallel rendering algorithms.

The Bulk Synchronous Parallel (BSP) model has been proposed as a parallel programming model independent from the details of the parallel hardware [4]. Under the BSP model, a parallel machine consists of a set of processor-memory pairs connected by an efficient communication network that supports the remote memory access and the barrier synchronisation of all processors. A BSP computation consists of a collection of processes, proceeding in phases. Each phase is called a *superstep*. All processes are synchronised by a barrier synchronisation at the end of each superstep. Within each superstep, a process performs computation on data held locally. It also initiates remote data accesses. However these remote data accesses are asynchronous (i.e. non-blocking), and none is guaranteed to complete until the end of the superstep, where the barrier synchronisation of all processes take place. Therefore these remote data are not guaranteed to be available until the beginning of the next superstep [5]. A research group at Oxford University has implemented a C programming library for writing parallel programs based on the BSP model [6]. A BSP model based parallel volume renderer was shown to have achieved high parallel efficiency [7]. Figure 1 shows two 3D images generated by this parallel volume render.

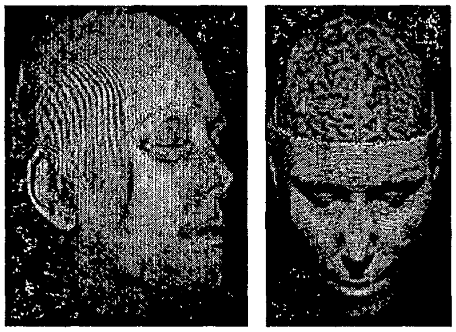

**Figure 1:** Two 3D images rendered from two volume data sets

In this paper, we will show how the BSP model-based parallel volume renderer [7] is enhanced using a superstep profiling system. The parallel program runs on many parallel machines including workstation clusters. One of the most important factors that affect the overall performance of a BSP model based parallel volume renderer is whether the computation maintains a balanced workload on all processors. This requires a careful design of the load balance strategy so that each process gets an equal share of the work during the computation, particularly during those computation intensive supersteps.

In the workstation cluster situation, however, the problem is complicated by the fact that most workstation clusters are used as shared resources. Therefore the ultimate load balance of the cluster is affected not just by the load balance of a specific parallel program, but also by the usage pattern of the workstations by other users. The later is often unpredictable and is generally outside the control of the designers of the parallel program. Unfortunately it causes the fluctuation of the execution time of the parallel program, and worse, it hides the true picture of the load balance situation of the parallel program under development. With the profile visualiser we can see whether the load balance is caused by the poor load balance strategy of the parallel program or by the other user programs running at the same time. We discuss the two types of load imbalance in details in the next section.

## II. TWO TYPES OF LOAD IMBALANCE

The performance of a parallel program running on a multi-user, shared, workstation cluster is affected by two different types of load balancing issues. The first type of load imbalance is caused by the poor load balance strategy used in the parallel program. Such load imbalance is intrinsic to the program, as it will occur every time the program is running, even when the machine is dedicated to that single program. This type of load balance problem can be solved by devising a better load balancing strategy in the parallel program.

The second type of load imbalance is caused by the factors outside of the control of the parallel program. Usually it is due to uneven workload among different workstations, because one or a few machines in the cluster are more heavily used than other workstations by programs other than the parallel program under development. Since it is not easy to predict how and when other users will use the shared the workstations, the behaviour of the parallel program can not always be repeated. In general, this type of load balance problem cannot be easily solved by redesigning the parallel program. It is possible that future operating systems for workstation clusters may incorporate some kind of dynamic load balance strategy in process and thread scheduling to alleviate the problem. However that is outside of the scope of this paper.

Much of the current efforts in parallel programming are to achieve as much as we can a balanced workload among all processors. In order to improve the load balance of a parallel program, we need to identify the first type of the load imbalance. Unfortunately when the program runs on a shared workstation cluster, it becomes difficult to separate the two types of load imbalance from the overall timing data.

For example, Figure 2 shows the timings of running an earlier version of our parallel volume renderer twice on each of the clusters with 1 to 12 workstations, compared to the theoretical times (ie one-processor time divided by the number of processors). The diagram reveals two facts: 1) there is a significant gap between the observed times and the theoretical times. This suggests that there may be load imbalances that increase the execution times. 2) the figure consistently shows that the same program (with the same input data) takes different times to complete on the same machine. This suggests that there exist the second types of load imbalances. It is difficult, however, to conclude whether the parallel program itself is load balanced (in terms of the first type load balance problem), and if it is not well balanced, how severely the imbalance is and where the imbalance occurred in the program.

In the next section we will describe a BSP profiler and visualiser that can be used to separate the two types of load imbalance, thus providing important clue as where the load imbalances are located and how to improve the load balance of the BSP program.

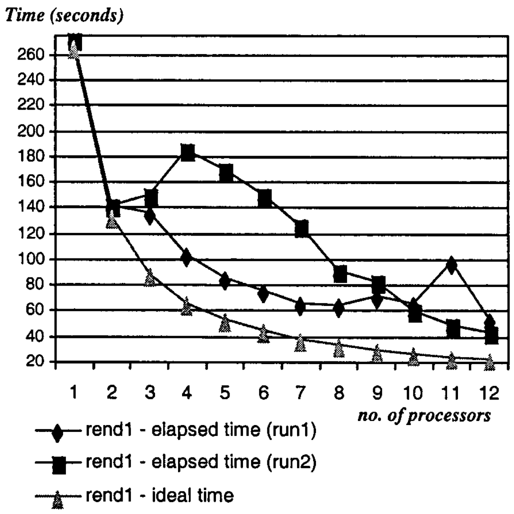

**Figure 2**. Times taken to run the same program on the clusters with 1 to 12 workstations.

## III. PROFILE VISUALISER

The load imbalance of the parallel volume renderer is analysed through the disclosure of the process load balance for each superstep. Two components are included in the superstep profiling system: a performance profiling tool, and a profile visualisation tool. The former component is designed to obtain comprehensive profiling information including time costs for both computation and inter-process communication between the processes. The profiling information is then displayed and shown as performance profiling graphs using the visualisation tool. The performance profiling tool is written in C and is linked to the BSP program. When the BSP program runs, the superstep profile is generated automatically for each run. The profile visualiser is written in Java. It displays the superstep profile in an easy-to-understand graphical format. The superstep profile graphs show for each superstep, the user and system time taken by the parallel program as well as the total elapsed time during the superstep. Any load imbalance will be exposed and highlighted.

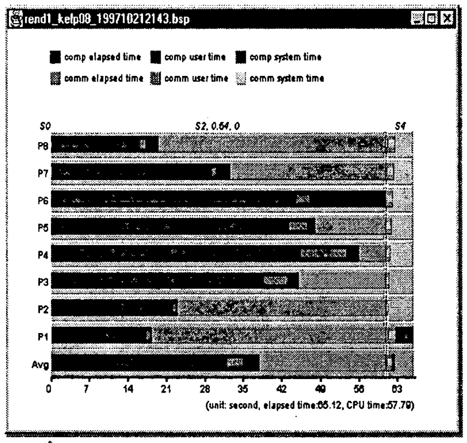

**Figure 3**. A superstep profile graph

For example Figure 3 shows a superstep profile graph. The profile was obtained from the profile data generated by running the parallel volume renderer on a cluster with 8 workstations. The graph shows both the first type of imbalance and the second type of imbalance (on workstation P6 in superstep 2). This graph reveals that the load balance strategy used by the parallel program did not perform very well, hence there is a significant potential to lift to the performance of the program by devising a better load balance strategy. It also shows that the observed performance of the parallel program was distorted because workstation P6 was overloaded with other tasks, hence slows down the speed of the program by roughly 7 seconds. We can realistically expect that the same program could take 7 less seconds to complete on the same machine if no other users are using the machine at the same time. The graph also shows in which supersteps the imbalances occurred. This information provided us important clue on how to improve the load balance of the above mentioned parallel program. The revised parallel program achieves much better load balance [7]. Figure 4 shows the improved performance for the revised program (with a better load balance algorithm). The data shown in Figure 4 were obtained by running the revised program (with the same input data) on the same set of clusters used in Figure 2, when no other users were using the machines. Compared to Figure 2, Figure 4 shows a much improved overall performance of the volume renderer.

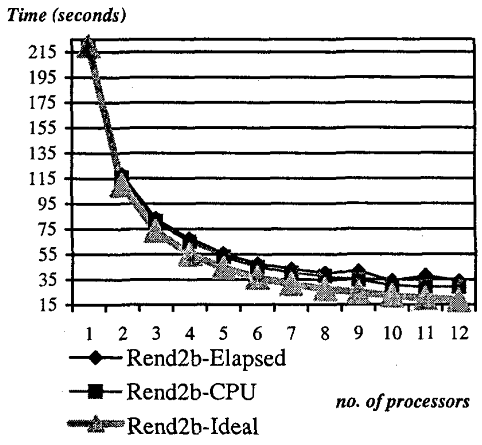

**Figure 4**. Comparision: results obtained from the new load balance strategy

## IV. CONCLUSION

In developing parallel program, especially parallel programs based on the BSP model, load balance is critical in achieving high parallel efficiency. While workstation clusters provide a cheap alternative for parallel computing, the nature of the multi-user, shared user environment often complicates the parallel program development. This is because the irregular and unpredictable usage pattern of the workstations often distorts the true picture of the performance of the parallel program under development. To use the workstation clusters to develop efficient parallel programs, especially BSP programs, it is desirable to be able to distinguish the load imbalance caused by the parallel program itself and that caused by the other user programs. In this paper we described just such a software tool. The superstep profiling system can reveal the two different types of load imbalance and their locations in an easy-to-understand graph. We have also demonstrated the usefulness of this tool by showing how the load imbalance in a BSP model-based parallel volume renderer was discovered. The discovery and isolation of the load imbalance provided us with important insight and clue on how to improve the load balance of the parallel volume renderer.

## References

[1]. R.A. Drebin, L. Carpenter, and P. Hanrahan. Volume rendering. *Computer Graphics*, 22(4):65-74, August 1988.

[2]. M. Levoy. Display of surfaces from volume data. *IEEE Computer Graphics and Applications*, May 1988, pp.29-37.

[3]. H.-P. Meinzer, K. Meetz, and D. Scheppelmann. The Heidelberg ray tracing model. *IEEE Computer Graphics and Applications*, pages 34-43, November 1991.

[4]. Valiant, L.G.: "A bridging model for parallel computation", *Communications of the ACM*, Vol. 33, No. 8, (1990) pp. 103-111.

[5]. McColl, W. F.: Scalable computing. In J. van Leeuwen (Ed.), *Computer Science Today: Recent* Trends and *Developments*, LNCS Vol. 1000, (1995) pp. 46-61, Springer-Verlag.

[6]. Hill, J.M.D., McColl, W. J M D Hill, B McColl, D.C. Stefanescu, M.W. Goudreau, J.M.D. Hill, K. Lang, S.B. Rao, T. Suel, T. Tsantilas and R H Bisseling: BSPlib: The BSP Programming Library. *Parallel Computing*, Vol. 24, No.14, 1998, pages 1947-1980

[7].. Xie, H.: "Slit-light Ray Tracing of Medical Slices on Multiple Processors: the BSP Approach", Proceedings of the 21st Australiasian Computer Science Conference (ACSC'98), Perth, 4-6 February (1998). In Australian Computer Science Communications, Vol.20, No.1, pp. 145-155., Springer.