Multi-GPU Load Balancing for In-situ Visualization

R. Hagan and Y. Cao
Department of Computer Science, Virginia Tech, Blacksburg, VA, USA

Abstract— *Real-time visualization is an important tool for immediately inspecting results for scientific simulations. Graphics Processing Units (GPUs) as commodity computing devices offer massive parallelism that can greatly improve performance for data-parallel applications. However, a single GPU provides limited support which is only suitable for smaller scale simulations. Multi-GPU computing, on the other hand, allows concurrent computation of simulation and rendering carried out on separate GPUs. However, use of multiple GPUs can introduce workload imbalance that decreases utilization and performance. This work proposes load balancing for in-situ visualization for multiple GPUs on a single system. We demonstrate the effectiveness of the load balancing method with an N-body simulation and a ray tracing visualization by varying input size, supersampling, and simulation parameters. Our results show that the load balancing method can accurately predict the optimal workload balance between simulation and ray tracing to significantly improve performance.*

Keywords: Multi-GPU Computing, Load Balancing, In-situ Visualization, N-body Simulation, Ray Tracing

1. Introduction

GPU computing offers massively parallel processing that can greatly accelerate a variety of data parallel applications. Use of multiple GPUs can lead to even greater performance gains by overlapping computations by executing multiple tasks on different GPUs. This provides an opportunity for handling larger scale problems that a single GPU cannot process in real-time. The resulting increase in runtime speeds can allow for real-time navigation and interaction, which can lead to a much more effective visualization experience. By designing effective algorithms to run on multiple GPUs, a considerable improvement in computational power can be realized. Effective load balancing can greatly increase utilization and performance in a multi-GPU environment by distributing workloads equally.

These properties make multiple GPUs suitable for in-situ visualization applications that use the GPU for concurrent simulation and rendering for interactive visualization. N-body simulation is one such application that involves computation of the interaction among a group of bodies. The N-body problem can be solved by computing the force of all bodies on each other. This problem is used in many domains, including biomolecular and physics applications. As in the work of [1], the gravitational N-body problem can be expressed as

$$F_i = Gm_i \sum_{1 \le j \le N, j \ne i} \frac{m_j r_{ij}}{||r_{ij}||^3} \quad (1)$$

where F_i is the computed force for body i, m_i is the mass of body i, r_{ij} is the vector from body i to j, and G is the gravitational constant.

In a molecular simulation, an N-body simulation algorithm can be used to compute the interaction of each atom in the molecule. This application can benefit from use of multiple GPUs to both compute new frames of simulation and render these new frames in parallel. Computation of simulation with rendering in real-time allows for interactive update in visualization applications. This can result in a smooth interaction experience with use of increased processing power to accelerate computing.

While multiple GPUs can offer a large performance gain in visualization applications, many challenges exist in scheduling multiple tasks. Load imbalance can lead to underutilization of available resources and reduced performance in the visualization. Multi-GPU computing can therefore benefit from a method for load balancing to improve workload distribution. Load balancing needs to account for performance in order to maximize use of available resources. Use of the load balancing method in an N-body simulation accounts for workload differences between simulation and rendering to maintain more equal workload distribution. Visualization algorithms such as ray tracing can be computationally expensive, so specific techniques to distribute and load balance rendering workloads can offer considerable performance gains for visualization applications. Taking advantage of concurrent computations of visualizations with simulation can lead to a significant performance improvement to maintain interactivity in these applications.

While load balancing can improve performance when using multiple GPUs, several factors need to be accounted for in visualization. The cost of simulation and rendering can depend on the chosen algorithms for each. The input data size may differ for applications, which can have a varying effect on simulation and rendering time. Accuracy of simulation can be improved by using more accurate techniques or by decreasing the timestep in simulation. In visualization, image quality is important to produce a better result for interactive viewing. Ray tracing is a rendering method that can produce realistic results on the GPU for visualization.

Supersampling can further improve image quality by using multiple samples per pixel that can decrease aliasing. Use of ray tracing with supersampling can greatly improve the results in interactive visualization but significantly increases computations that can be accelerated through use of multiple GPUs. Supersampling and improving the accuracy of simulation can vary the cost of computations, which will require adjusting load balancing for multi-GPU processing. Our method addresses this issue to improve performance with multi-GPU visualization.

Due to the significant gains possible with use of multiple GPUs, we implement load balancing for multi-GPU visualization applications. Our work provides several contributions, including:

- Acceleration of an N-body simulation and ray tracing application using multiple GPUs
- Performance analysis of workload variation based on multiple input parameters
- A load balancing method to predict optimal workload distribution and significantly improve performance

2. Related Work

There have been several related areas of previous research, including multi-GPU computing and visualization using the GPU.

Previous work in multi-GPU visualization has included several applications that use multiple GPUs for rendering. Fogal et al. present a system for visualizing volume datasets on a GPU cluster [2]. However, their work could benefit from additional load balancing between GPU tasks that could provide more flexible and effective workload distribution for simulation and rendering. Monfort et al. present an analysis of split frame and alternate frame with multiple GPUs for a game engine [3]. They present an analysis of load balancing for a combined rendering mode to improve utilization of multiple GPUs. Binotto et al. present work in load balancing in a CFD simulation application [4]. They use both an initial static analysis followed by adaptive load balancing based on various factors including performance results. While these previous works present load balancing techniques, they do not focus on load balancing for in-situ visualization based on rendering and simulation tasks. We present a performance model and load balancing technique for simulation and rendering that allows improved load balancing and accounts for the pipelining process necessary in a multi-GPU environment.

Other work has focused on streaming for out-of-core rendering. Gobbetti et al. present Far Voxels, a visualization framework for out-of-core rendering of large datasets using level-of-detail and visibility culling [5]. Crassin et al. present GigaVoxels, an out-of-core rendering framework for volume rendering of massive datasets using a view-dependent data representation [6]. While our load balancing method also uses multiple GPUs and similar pipelining to visualize datasets, we focus specifically on load balancing for in-situ visualization using ray tracing.

Several other frameworks have been proposed that use multiple GPUs for general-purpose computations. Harmony presents a framework that dynamically schedules kernels [7]. Merge provides a framework heterogeneous scheduling that exposes a map-reduce interface [8]. DCGN is another framework that allows for dynamic communication with a message passing API [9]. However, these works focus on providing a general framework not specific to visualization and could benefit from additional tools for load balancing. We employ similar techniques to improve multi-GPU performance, but we provide improved load balancing in a visualization and simulation application.

Other previous work has focused on GPU computing in molecular dynamics applications, relating to the N-body simulation and visualization used in our work. Past work has included Amber, a molecular dynamics software package that offers tools for molecular simulation [10]. This simulation can be used to compute the change in atoms over time due to an N-body simulation. Other research in molecular dynamics has included work by Anandakrishnan et al. to use an N-body simulation to compute the interaction of atoms in a molecule [11]. Humphrey et al. present Visual Molecular Dynamics (VMD), a software package for visualization of molecular datasets [12]. However, VMD provides primarily off-line rendering that does not simulate and render each frame interactively to allow for real-time user interaction. Stone et al. present work that computes molecular dynamics simulations on multi-core CPUs and GPUs [13]. Furthermore, they visualize molecular orbitals in an interactive rendering in VMD. We also apply our work to an N-body simulation, but we focus on load balancing using multiple GPUs for both simulation and rendering while their work focuses on data parallel algorithms on single GPUs. Chen et al. present work in multi-GPU load balancing applied to molecular dynamics that uses dynamic load balancing with a task queue [14]. Their framework focuses on fine-grained load balancing usable within a single GPU, while our work focuses on coarse-grained task scheduling among GPUs for both simulation and visualization. While there has been considerable work in visualization and multi-GPU computing, we focus on a method for load balancing between simulation and rendering to improve utilization in the pipelined memory model useful for in-situ visualization.

3. Methods

Our approach addresses the issue of workload imbalance for in-situ visualization applications. We will first describe the problems in this application area, and then we present our load balancing method for solving these issues.

3.1 Multi-GPU Architecture

In comparison with a single GPU, a multi-GPU implementation has several advantages. Most notably, multiple GPUs can overlap concurrent computations on several GPUs at once. However, unlike a single GPU, memory transfers are required to ensure that a GPU has the required data. Figure 1 shows the multi-GPU configuration used with our application. It identifies how multiple GPUs can be used for overlapping computation between simulation and rendering, while host memory is used to transfer results among GPUs. For in-situ visualization applications, the simulation data must be transferred to rendering tasks to render the resulting image. This data is first transferred to the host and then transferred to the recipient GPU. This creates a pipelined model of execution where multiple GPUs can process data concurrently but must transfer data through host memory.

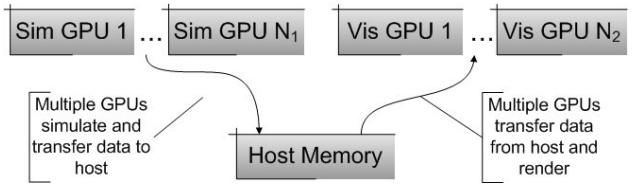

Fig. 1: Diagram of multi-GPU configuration: "Sim" refers to simulation and "Vis" refers to visualization

3.2 Multi-GPU Workload Imbalance

While use of multiple GPUs can greatly increase the available processing power and improve performance by overlapping computation, use of multiple GPUs introduces issues of workload distribution among processors. This workload imbalance results from the synchronization necessary through host memory. Each task either sends or receives data through a buffer. However, use of a single buffer would require simulation tasks to wait for rendering tasks to read this data before writing the next frame. This can result in significant idle times that can decrease utilization and performance. Multiple buffers can allow one task to read or write data to multiple buffers before having to wait for other tasks to process the data as shown in Figure 2. Thus, having multiple buffers can improve load balance at the cost of additional memory.

The amount of host memory is finite, however, which requires the tasks to eventually wait if workload imbalance is significant. If simulation of a single frame requires less time than rendering, host memory eventually becomes full, which requires simulation to wait for rendering. When rendering of a single frame takes less time than simulation, the buffers in host memory become increasingly empty, which requires rendering tasks to wait for simulation. Figure 3 shows the case where performance can decrease due to improper workload distribution. Since rendering GPUs need to read simulation data before simulation can overwrite it, idle

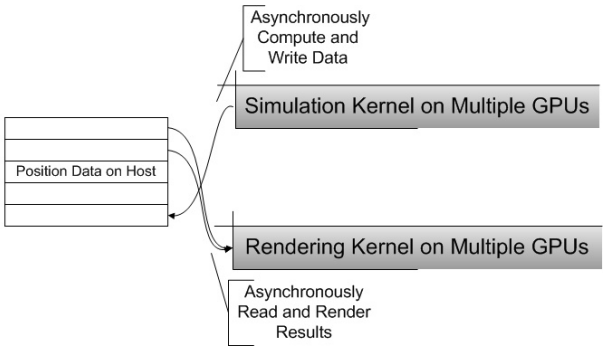

Fig. 2: Memory transfers between host and GPU memory

time can be introduced if simulation time is shorter than rendering.

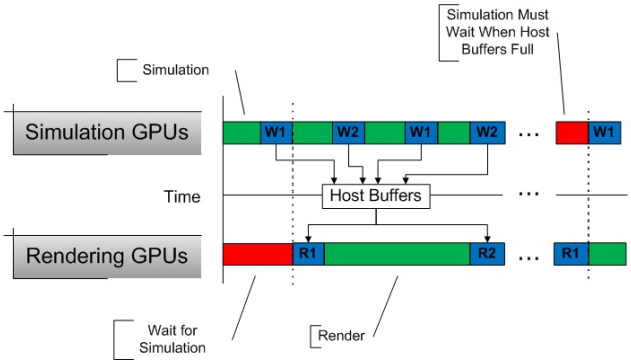

Fig. 3: Simulation tasks must wait for ray tracing to read results. "W1" refers to writing data to the first buffer in host memory from a GPU, while "R2" refers to reading data from the second buffer in host memory

Accounting for workload imbalance can eliminate this idle time by distributing work equally among available processors as shown in Figure 4. Load balance among tasks allows tasks to send and receive data at an equal rate and thus improve utilization. However, this requires formulating a method for load balancing. Varying workloads for tasks creates issues for the problem of load balancing. For example, the type of rendering technique or number of samples in supersampling can change the workload and introduce additional idle time. Factors of simulation such as accuracy or type of simulation could also affect runtime, resulting in a different optimal workload balance as well. These various issues demonstrate the important need for load balancing for in-situ visualization. Given an initial set of characteristics for rendering and simulation for a specific visualization, finding the optimal load balance can reduce idle time and improve performance.

3.3 Load Balancing

The use of our multi-GPU implementation allows for load balancing techniques for in-situ visualization. Our test

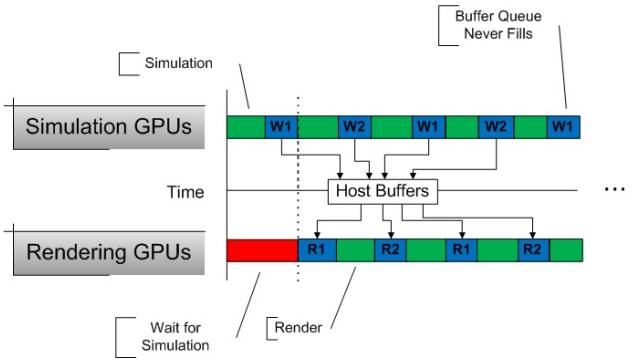

Fig. 4: Use of load balancing reduces idle time

application uses a gravitational N-body simulation based on the method of [1] to compute interactions of particles, while ray tracing renders the results. Particle position data is transferred through the host using multiple buffers in order to pipeline position data between simulation and rendering. Since simulation and rendering may have different amounts of workload, it is important to address the possibility of workload imbalance between the tasks.

In this application load balancing requires partitioning the dataset in order to distribute work to GPUs. Two types of work partitioning are possible with the application: *inter-frame* and *intra-frame*. Inter-frame partitioning involves distributing complete frames of data in order to achieve load balancing. Ray tracing in this implementation uses inter-frame partitioning to render entire frames in order to avoid communication in combining results and improve performance. Intra-frame partitioning distributes parts of a single frame to GPUs for processing. The N-body simulation utilizes intra-frame partitioning by having each GPU update only a subset of the particles for a single frame of data. Since each frame of simulation requires previous data, simulation cannot be computed out of order. Thus, multiple GPUs can only accelerate simulation by having each GPU update a subset of the dataset. Intra-frame partitioning for simulation is therefore necessary to apply load balancing. While this requires communication to combine the results for each frame, the computation can be distributed among multiple GPUs. Thus, this load balancing method uses groups of GPUs to compute frames of data for simulation while rendering has single GPUs separately compute rendering results for consecutive frames. The partitioning of both rendering and simulation tasks allow for load balancing in the visualization application. As more processors are dedicated to simulation, fewer are dedicated to rendering consecutive frames. The total visualization time for a frame is used to determine the optimal load balance between rendering and simulation among the available processors.

To address the issue of workload imbalance, we present a load balancing method to achieve the optimal distribution of work to improve performance. We present a three-dimensional parameter matrix M that can be used to determine the appropriate balance for the visualization application. The input dimensions of M include the number of samples for supersampling, the number of iterations for simulation, and the input size, while the associated output values are performance times for these configurations. Our method first collects performance results for this matrix and then computes the desired workload balance for a new set of input parameters. Thus, we find the solution to the function:

$$f(i, s, p) = g \tag{2}$$

where f is the function to compute optimal workload distribution, i is the number of iterations for simulation, s is the number of samples for supersampling, p is the number of particles in the simulation, and g is the number of GPUs allocated for rendering versus simulation. The predicted optimal load balance is computed based on previous results through trilinear interpolation. Given known optimal workload distributions for sets of input parameters, our model predicts the optimal load balancing result g for a new set of input parameters:

$$\begin{aligned} L_{isp} = {} & L_{000}(1-i)(1-s)(1-p) && + L_{100}i(1-s)(1-p) \\ & + L_{010}(1-i)s(1-p) && + L_{001}(1-i)(1-s)p \\ & + L_{101}i(1-s)p && + L_{011}(1-i)sp \\ & + L_{110}is(1-p) && + L_{111}isp \end{aligned} \tag{3}$$

where L is the optimal load balance, i is the number of simulation iterations, s is the number of samples for ray tracing, and p is the number of particles. Here, i, s, and p are normalized to the range [0, 1] for interpolation, and the result g is rounded to the nearest integer. This result gives a prediction for the optimal load balance for a given set of input parameter values.

4. Results

The multi-GPU load balancing method was tested with an N-body simulation and ray tracing of thousands of spheres. All tests were done on a single computer with eight GTX 295 graphics cards.

The final result of the visualization and the differences in supersampling can be seen in Figure 6. Aliasing artifacts due to inadequate sampling can be seen in the image on the left with one sample per pixel. Using sixteen samples per pixel in a random fashion, however, significantly improves the results.

While supersampling improves the quality of the final image, it comes at a performance cost as shown in Table 1. The increase in execution time for a greater number of samples for supersampling is linear. Thus, the tradeoff between performance and image quality must be considered when

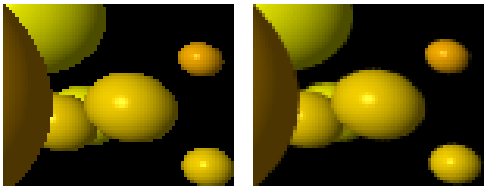

Fig. 5: Comparison of single sample (left) and 16 sample randomized supersampling (right)

Table 1: GPU execution time (ms) for ray tracing based on number of samples for supersampling for 1000 particles

1 sample	4 samples	8 samples	12 samples	16 samples
46.082 ms	163.82 ms	316.56 ms	469.39 ms	621.60 ms

choosing an appropriate number of samples for supersampling.

Table 2 shows the performance time for simulation when performing multiple iterations with a smaller timestep. The performance of simulation shows a linear increase in time with an increase in number of iterations. While a smaller timestep provides more accurate simulations, it introduces additional computations for each frame. Thus, using a smaller timestep but increasing the number of iterations leads to an increase in performance. Table 3 shows the percent difference in positions of simulation from 12000 iteration simulation, which uses the smallest timestep. Each simulation is carried out over the same total time, with a smaller timestep for simulations run for more iterations. With a smaller timestep, the accuracy of the simulation is improved due to the finer granularity used for integration in the N-body simulation.

Table 4 shows a linear decrease in the execution time for simulation when partitioning the dataset to simulate on multiple GPUs. Due to slight constant overhead of launching the kernel, etc., six GPUs gain a slightly less than six times speedup over use of one GPU.

Ray tracing has a longer execution time than simulation for smaller dataset sizes. Simulation takes less time for smaller datasets, but with an increased number of simulation iterations this cost can exceed that of ray tracing with fewer samples. These differences in workload affect the final optimal load balance.

Table 2: GPU execution time for simulation based on number of iterations

20 iterations	40 iterations	60 iterations	80 iterations
31.87 ms	62.56 ms	92.70 ms	122.86 ms

Table 3: Percent difference in positions of simulation from 12000 iteration simulation

Iterations	2000	4000	6000	8000	10000	12000
Percent	58.07	35.37	26.87	21.81	14.94	0.00

Table 4: Execution time for simulation based on number of GPUs used

1 GPU	2 GPUs	3 GPUs	4 GPUs	5 GPUs	6 GPUs
31.87 ms	16.69 ms	11.52 ms	9.32 ms	7.58 ms	6.44 ms

4.1 Workload Characteristics

The multiple input parameters for this application result in many possibilities for workloads. These varying workloads can introduce a performance penalty if not accounted for in distribution of work. Figure 7 shows the trends for performance times for different workloads (number of ray tracing tasks) with varying dataset sizes with 16 sample ray tracing and 80 iteration simulation. The cost of simulation increases more as dataset size increases due to the nature of the N-body simulation, while ray tracing scales linearly with dataset size. This causes the overall performance to be increasingly limited by simulation time for larger datasets.

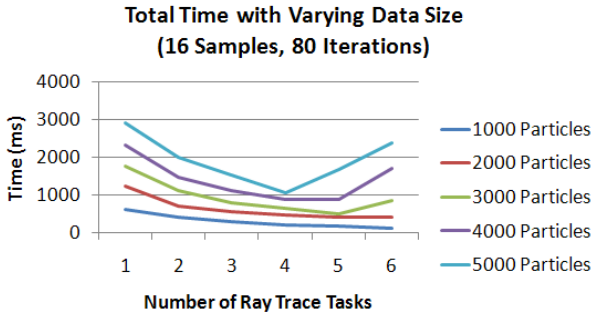

Fig. 6: Performance time for various input sizes with 16 samples, 80 iterations

Figure 8 shows performance for different input sizes with a varying workload distribution for four sample ray tracing and 80 iterations for simulation. This graph shows that allocating more GPUs for simulation when the number of samples is low can result in performance gain. The difference in the trend from Figure 7 also demonstrates that different input parameters can lead to significantly different optimal workload distributions that requires load balancing.

4.2 Load Balancing

Figure 9 shows a trend of optimal load balance based on the number of iterations for simulation. As shown, increasing the number of iterations requires a greater number of GPUs dedicated to simulation to achieve optimal load balance. With the fewest iterations for simulation, the majority of GPUs should be allocated for ray tracing due to the greater cost of ray tracing.

Increasing the number of samples for supersampling increases the cost of ray tracing and also impacts the load balancing scheme. Figure 10 shows that increasing the number of samples for supersampling results in need of

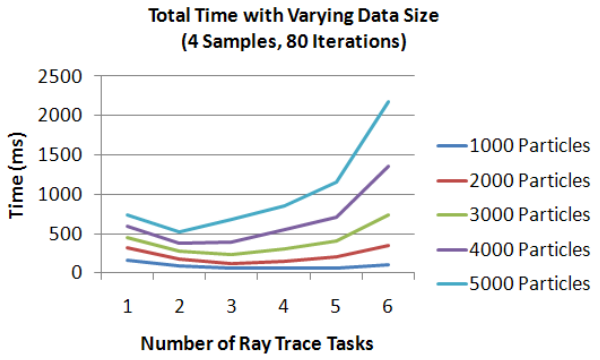

Fig. 7: Performance time for various input sizes with four samples, 80 iterations

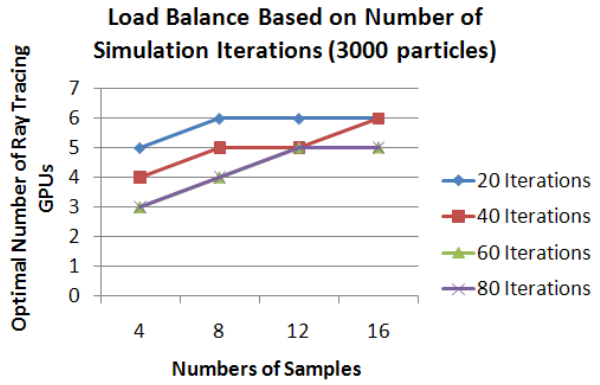

Fig. 8: Load balancing for various simulation iterations with four samples, 3000 particles

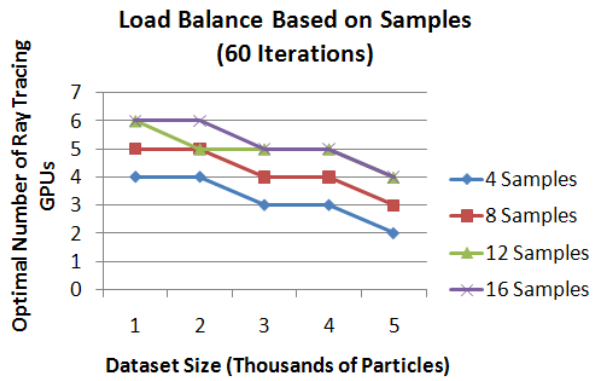

Fig. 9: Load balancing for various numbers of samples for ray tracing with changing dataset size

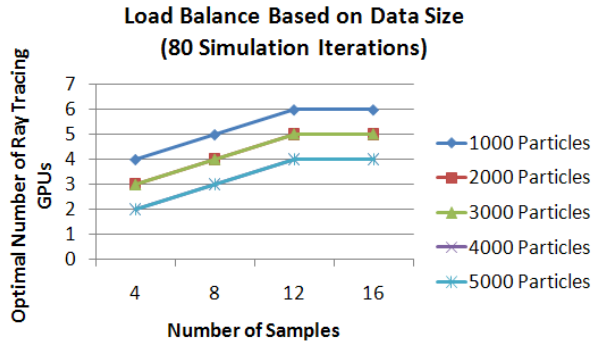

Fig. 10: Load balancing for various dataset sizes with 80 simulation iterations

additional ray tracing tasks to improve workload balance. A larger dataset size requires fewer GPUs for ray tracing due to the smaller increase in cost of ray tracing with larger datasets.

Figure 11 shows the trend for varying dataset size and number of samples with a constant simulation. With a larger dataset size, simulation becomes increasingly expensive while ray tracing cost increases at a linear rate. Therefore, it becomes necessary to compute simulation on an increasing number of GPUs with larger datasets to maintain workload balance.

These results demonstrate that significant workload imbalance can be introduced based on differing workloads of rendering and simulation. Each configuration leads to a different optimal load balancing configuration. A load balancing method must be able to account for these varying trends in order to achieve effective performance.

4.3 Performance Model

We now present a summary of the results of applying our load balancing method. Table 5 shows the percent error in our proposed prediction model when compared to the actual optimal load balanced configuration. The performance model was tested by computing the average percent error for 40 values for varying one dimension (simulation iterations), 20 for varying two dimensions (iterations and samples), and 12 for varying three dimensions (iterations, samples, and data size). These results show that the average percent difference is below 5 percent for two and three dimensions, and below 10 percent for all categories. Interpolation with fewer dimensions yields a larger error due to discretization error with selection of number of GPUs for load balancing. Using a larger number of data points with more dimensions in interpolation decreases this discretization error.

The performance of our load balancing method was also compared against the worst and average case workload distribution. Figure 12 shows the speedup of using the load

Table 5: Percent error in load balancing model for a varying number of dimensions

1 dimension	2 dimensions	3 dimensions
9.88%	2.67%	1.67%

balancing method. These results show that by using the model, a significant speedup can consistently be achieved with different parameter configurations.

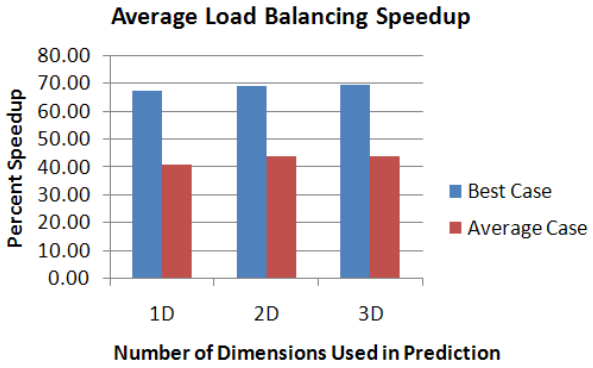

Fig. 11: Load balancing speedup over the average and worst cases

5. Conclusions and Future Work

We have proposed a multi-GPU load balancing solution for in-situ visualization. We have presented an analysis of the workload properties and load balancing results in an N-body simulation and ray tracing visualization. Our results show that workloads can vary greatly for different sets of input parameters, which demonstrates the need for load balancing in multi-GPU computing. Our multi-GPU implementation demonstrates the use of intra- and inter-frame task partitioning for scheduling of GPU tasks to allow the use of load balancing. The results of our tests show that the load balancing method can accurately predict optimal workload balance to significantly improve performance by increasing utilization of available resources.

This work could be extended in multiple ways in future work. Our load balancing approaches could be extended to additional visualization applications, where other rendering and simulation methods with varying workloads could also be addressed. Different performance models may be useful for other applications as well. The pipelining model used in this application would also be useful for out-of-core rendering of massive models.

6. Acknowledgements

This work is funded by Air Force Research Laboratory Munitions Directorate, FA8651-11-1-0001, titled "Unified High-Performance Computing and Visualization Framework on GPU to Support MAV Airframe Research." I also thank Dr. Eli Tilevich for support and discussion on the project.

References

[1] J. P. L. Nyland, M. Harris, "Fast n-body simulation with cuda," *GPU Gems 3*, pp. 677–695, 2007.

[2] T. Fogal, H. Childs, S. Shankar, J. Krüger, R. D. Bergeron, and P. Hatcher, "Large data visualization on distributed memory multi-gpu clusters," in *Proceedings of the Conference on High Performance Graphics*, ser. HPG '10. Aire-la-Ville, Switzerland, Switzerland: Eurographics Association, 2010, pp. 57–66. [Online]. Available: http://portal.acm.org/citation.cfm?id=1921479.1921489

[3] J. R. Monfort and M. Grossman, "Scaling of 3d game engine workloads on modern multi-gpu systems," in *Proceedings of the Conference on High Performance Graphics 2009*, ser. HPG '09. New York, NY, USA: ACM, 2009, pp. 37–46. [Online]. Available: http://doi.acm.org/10.1145/1572769.1572776

[4] A. Binotto, C. Pereira, and D. Fellner, "Towards dynamic reconfigurable load-balancing for hybrid desktop platforms," in *Parallel Distributed Processing, Workshops and Phd Forum (IPDPSW), 2010 IEEE International Symposium on*, 04 2010, pp. 1 –4.

[5] E. Gobbetti and F. Marton, "Far voxels: a multiresolution framework for interactive rendering of huge complex 3d models on commodity graphics platforms," in *ACM SIGGRAPH 2005 Papers*, ser. SIGGRAPH '05. New York, NY, USA: ACM, 2005, pp. 878–885. [Online]. Available: http://doi.acm.org/10.1145/1186822.1073277

[6] C. Crassin, F. Neyret, S. Lefebvre, and E. Eisemann, "Gigavoxels: ray-guided streaming for efficient and detailed voxel rendering," in *Proceedings of the 2009 symposium on Interactive 3D graphics and games*, ser. I3D '09. New York, NY, USA: ACM, 2009, pp. 15–22. [Online]. Available: http://doi.acm.org/10.1145/1507149.1507152

[7] G. F. Diamos and S. Yalamanchili, "Harmony: an execution model and runtime for heterogeneous many core systems," in *Proceedings of the 17th international symposium on High performance distributed computing*, ser. HPDC '08. New York, NY, USA: ACM, 2008, pp. 197–200. [Online]. Available: http://doi.acm.org/10.1145/1383422.1383447

[8] M. D. Linderman, J. D. Collins, H. Wang, and T. H. Meng, "Merge: a programming model for heterogeneous multi-core systems," *SIGPLAN Not.*, vol. 43, pp. 287–296, March 2008. [Online]. Available: http://doi.acm.org/10.1145/1353536.1346318

[9] J. A. Stuart and J. D. Owens, "Message passing on data-parallel architectures," in *Proceedings of the 23rd IEEE International Parallel and Distributed Processing Symposium*, May 2009.

[10] D. A. Pearlman, D. A. Case, J. W. Caldwell, W. S. Ross, T. E. Cheatham, S. DeBolt, D. Ferguson, G. Seibel, and P. Kollman, "Amber, a package of computer programs for applying molecular mechanics, normal mode analysis, molecular dynamics and free energy calculations to simulate the structural and energetic properties of molecules," *Computer Physics Communications*, vol. 91, no. 1-3, pp. 1 – 41, 1995. [Online]. Available: http://www.sciencedirect.com/science/article/B6TJ5-4037S49-D/2/0df1c6e2cbc422472f7498040a749b20

[11] R. Anandakrishnan and A. V. Onufriev, "An n log n approximation based on the natural organization of biomolecules for speeding up the computation of long range interactions," *Journal of Computational Chemistry*, vol. 31, no. 4, pp. 691–706, 2010. [Online]. Available: http://dx.doi.org/10.1002/jcc.21357

[12] W. Humphrey, A. Dalke, and K. Schulten, "VMD – Visual Molecular Dynamics," *Journal of Molecular Graphics*, vol. 14, pp. 33–38, 1996.

[13] J. E. Stone, J. Saam, D. J. Hardy, K. L. Vandivort, W.-m. W. Hwu, and K. Schulten, "High performance computation and interactive display of molecular orbitals on gpus and multi-core cpus," in *Proceedings of 2nd Workshop on General Purpose Processing on Graphics Processing Units*, ser. GPGPU-2. New York, NY, USA: ACM, 2009, pp. 9–18. [Online]. Available: http://doi.acm.org/10.1145/1513895.1513897

[14] L. Chen, O. Villa, S. Krishnamoorthy, and G. Gao, "Dynamic load balancing on single- and multi-gpu systems," in *Parallel Distributed Processing (IPDPS), 2010 IEEE International Symposium on*, 04 2010, pp. 1 –12.