Hindawi
Wireless Communications and Mobile Computing
Volume 2022, Article ID 3485505, 11 pages
https://doi.org/10.1155/2022/3485505

Research Article

GPU-Based In Situ Visualization for Large-Scale Discrete Element Simulations

Pengfei Qin,[1,2] Zhaojie Xia,[1,2] Guoxian Gao,[1,2] Xiaofang Tao,[1,2] and Li Guo[1,2,3]

[1]*State Key Laboratory of Multiphase Complex Systems, Institute of Process Engineering, Chinese Academy of Sciences, Beijing 100190, China*
[2]*School of Chemical Engineering, University of Chinese Academy of Sciences, Beijing 100049, China*
[3]*Innovation Academy for Green Manufacture, Chinese Academy of Sciences, Beijing 100190, China*

Correspondence should be addressed to Li Guo; lguo@ipe.ac.cn

Received 19 August 2022; Revised 16 September 2022; Accepted 28 September 2022; Published 16 November 2022

Academic Editor: Lei Liu

The result of discrete element simulations has tens of thousands of time frames and hundreds of billions of interacting particles in each frame, which may lead to very large data sets. The existing postprocessing visualization technology cannot deal with such large-scale data, which will bring a negative impact to the simulation, affect the simulation speed, and make it difficult to realize the runtime steering. In this work, an attempt has been made to optimize visualization methods by directly accessing DEM simulation results, data exchanging straightly between GPUs, and accelerating pixel composition by GPU. The key feature of this work is that the entire process from DEM simulation to graphical rendering and image composition is done inside the same GPU.

1. Introduction

The particle system is common in daily life, industrial process, ecological environment, and other aspects. With the rapid development of computer technology, discrete element method (DEM) simulation proposed in the 1970s has been widely used in particle systems [1, 2]. This method plays a significant role in the engineering and technology design of mining, architecture, pharmaceutical, and food processing [3–5].

However, the output of DEM simulation is complex and not intuitive, which is difficult to understand directly [6]. By showing the results of simulations graphically, scientific visualization can make many abstract, difficult-to-understand principles, or rules easy to understand, and the tedious and boring data becomes lively and interesting. Therefore, scientific computing visualization is a crucial part of the scientific discovery process, while DEM simulation is no exception.

Traditional visualization of DEM simulation generally adopts a postprocessing mode [6, 7], which is to save the results of simulation time steps to the storage device in a predefined frequency during the simulation. After that, the data can be visualized and analyzed when needed. There are three prominent problems in this manner: (1) the simulation result data must be sampled in both time and space dimensions due to the limited storage capacity. So, the valid information may be discarded that the scientific data integrity cannot be guaranteed; (2) due to the limited performance of disk I/O, operations such as transmission, management, and partition preprocessing of hyperscale data between compute nodes, disk arrays, and visualization nodes consume a large amount of system resources and time, which greatly reduces the efficiency of DEM simulation and fails to meet the requirements of real-time data analysis; (3) the simulation status cannot be monitored in real-time mode.

In recent years, computer hardware capability has been continuously improved, which enables scientists to solve more complex and larger scientific problems. At the same time, the generated data from simulation has also exploded. Now, the size of data from one simulation has reached the order of TB/PB. However, the speedup of supercomputer's I/O cannot catch up with that of computation. In order to deal with this situation, an idea has emerged that running simulation and visualization on the same parallel supercomputer in order to share data, which is so-called in situ visualization.

At present, the number of particles simulated by DEM can reach 100 billion [8], and the real-time storage of the data will seriously affect the simulation process. In situ visualization refers to a different approach with the traditional postprocessing: the data is processed at the same place and same time while it is being produced by the simulation, allowing visualization to be done without involving disk I/O. This effectively reduces data I/O overhead and meets the needs of large-scale numerical simulation real-time visualization. The in situ visualization deals with the result via direct access to the simulation's memory without occupying extra disk space.

There are many advantages to in situ visualization. Firstly, in situ visualization can deal with simulation results at smaller intervals, which may help to more detailed scientific discoveries. Since the disk space is limited and data I/O takes up a lot of time, simulation limits the number of steps they output for postprocessing visualization. But in situ visualization can work at a low cost and has a small influence on the simulation program. In addition, some simulations generate so much data that the data size must be reduced, for example, by subsampling, but this is not necessary for in situ visualization. Last but not least, in situ visualization is deeply coupled to the simulation, so it can be used to either monitor or steer simulations.

In practice, this approach is rarely used for two reasons. First, most scientists are reluctant to use their supercomputer for visualization, especially when computing resource is expensive. Second, combining parallel simulation programs with visualization programs would require a lot of effort. However, in situ visualization is a viable solution to the upcoming extreme-scale data problem in the field of scientific supercomputing, and it will be the inevitable direction for future scientific computing visualization.

The main contributions of this paper are summarized as follows:

(1) This paper proposes a novel parallel in situ visualization framework for DEM simulation program based on GPU. The entire process from data generation to graphics rendering to image compositing is all completed in GPU, minimizing the visualization time

(2) Based on C + +, CUDA, OpenGL, and other languages or libraries, this work solves specific problems such as parallel off screen rendering and data transfer between GPUs and realizes the abovementioned in situ visualization framework

(3) The paper has done extensive tests on the implemented program. The test results show that compared with traditional visualization programs, this method has an order of magnitude improvement in time. And this method has good scalability, the time required for visualization is proportional to the image resolution, and the higher the resolution, the shorter the visualization time per unit resolution

2. Related Work

In recent years, with the rapid increase in the amount of simulation data, in situ visualization method has been paid more and more attention and has gradually been widely used. In 2010, Yu et al. [9] proposed a customized coprocessing in situ processing method, which was applied to a turbulent combustion simulation and ran on the Cray XT5 supercomputer, using up to 15260 CPU cores. In 2013, Dorier et al. [10] designed and implemented an in situ visualization framework called Damaris and tested it in the CM1 atmospheric simulation, achieving good results. In 2014, Ahrens et al. [11] proposed a highly interactive, image-based in situ visualization method, which promoted the exploration of simulation results and significantly reduced data movement and storage. In 2017, Camata et al. [12] used Paraview Catalyst to visualize turbine current simulation in situ and found that both in situ visualization and in-transit data analysis are negligible and enable monitoring the sediment appearance at runtime.

The general in situ visualization framework meets the needs of in situ visualization of many simulation programs to some extent. In 2011, Fabian et al. [13] proposed a general in situ visualization framework, Catalyst. The framework is based on VTK graphics library and ParaView. Through the integration of Catalyst plug-ins, many simulations add the function of in situ visualization. In 2011, Kuhlen et al. [14] developed the in situ visualization library Libsim, which is based on VisIt visualization tools to enable simulation programs to achieve in situ visualization with minor modifications. However, the existing general in situ visualization framework has the problem of low efficiency.

For the existing DEM simulation program, Xiaojiang Fang has made a real-time visualization program before. Fang et al. [15] implemented a parallel online visualization program for DEM particle simulation in 2011. OpenGL library was used for image rendering, and image compositing was implemented with the IceT library. By introducing FBO (Frame Buffer Object (FBO)), the off-screen rendering of the simulation results is realized. Furthermore, image compositing was accelerated with the help of CUDA.

3. System Implementation

3.1. Implementation Strategy. The system design is shown in Figure 1. DEM simulation is carried on GPU, and the results are generated in its video memory. Then, the result is directly used by the rendering program. Each rendering process renders the result data in the video memory to generate local images, i.e., the rendering process in gpu1 generates local images corresponding to the local data in gpu1. Then, all the images generated by each rendering process are gathered by the image compositing process to produce the final image. Data transmission between GPUs is also completed directly without the help of CPU, and all the simulation

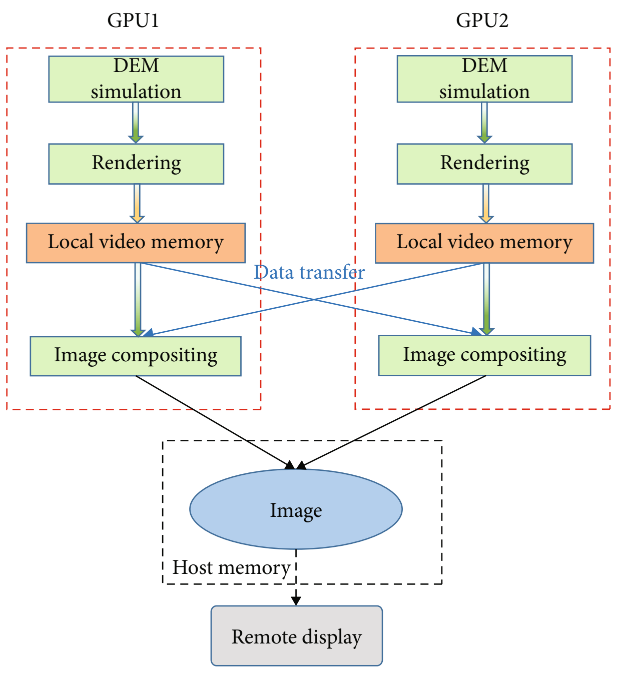

Figure 1: System design.

and graphic rendering tasks are executed inside GPU. These not only accelerate the rendering calculation but also avoid the data transferring consumption between GPU and CPU.

To do this, there are several problems that must be solved:

(1) In general, there is no display device installed on the simulation node. And some graphics cards dedicated to scientific computing, such as the Tesla series, have no image output capabilities. So, it is crucial to realize efficient in situ rendering in such a nonscreen visualization environment

(2) How to transfer data efficiently between CUDA-based simulation program and OpenGL-based visualization program

(3) Which parallel rendering architecture is chosen to optimize visualization

(4) How to realize pixel data transmission efficiently between GPU processes without CPU as a bridge

(5) How to use GPU to accelerate pixel composition in the stage of image composition

(6) Choose an appropriate image compositing strategy according to the computer hardware architecture

3.2. Parallel Off-Screen Rendering. In general, the output device of the visualizer is the screen. However, DEM simulation is a parallel program running on the cluster computer, with no screen as the output device. Therefore, solving the problem of parallel off-screen rendering is the first problem to realize in situ visualization [16].

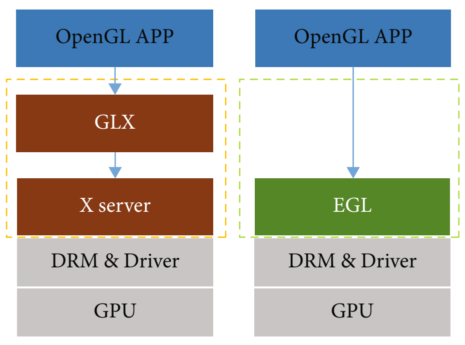

Figure 2: Compare GLX and EGL.

Our work uses the OpenGL library for image rendering. OpenGL is an industry standard open source library for 2D/3D image generation and manipulation. The most notable feature of the library is that it is independent of the operating system, and its application can be easily ported to another operating system with high compatibility and scalability. As a state-based API, OpenGL requires a context to maintain the state between multiple API calls. So, a tool library is needed to create the contexts. Prior to EGL, an application obtained an OpenGL context by calling the X server via the GLX library. Depending on the version of X server, X had to be run as a privileged process. While this approach enables the use of the graphics hardware in these GPU-accelerated supercomputers for graphics purposes, the launch of an X-Windows server on each compute node is far from elegant. The extra X Windows server software introduces additional complexities for queuing systems' prologs and epilogs and may introduce OS jitter or other system effects at a level unacceptable to the system operators. In addition, the complexity of configuring and supporting windowing systems for HPC can be daunting due to the degree of specialization (and often minimalism) of their compute node OS software.

As Figure 2 shows, EGL simplifies this architecture by allowing applications to obtain an OpenGL context without an X server. Embedded-System Graphics Library (EGL) [16] is a software interface that provides applications with access to platform native OpenGL rasterization. Different from the traditional library, such as GLX, EGL is specifically designed for supporting platforms that use custom windowing systems or no windowing system at all. So, EGL library is a good choice to create the OpenGL context.

There are many advantages to getting rid of the dependence on window system. By eliminating the need for a running windowing system, a significant application deployment obstacle is removed. The windowing system process and associated OS services consume compute node resources, occupying a small amount of system memory and increasing the number of active kernel threads and user mode processes. Launch-time loading of windowing system-associated shared libraries and configuration files creates additional I/O activity during parallel job launch, slowing job launch and potentially causing disruption to other running jobs. Moreover, another exciting benefit of EGL in the context of multi-GPU compute nodes is that it provides APIs that enable applications to take simple and direct

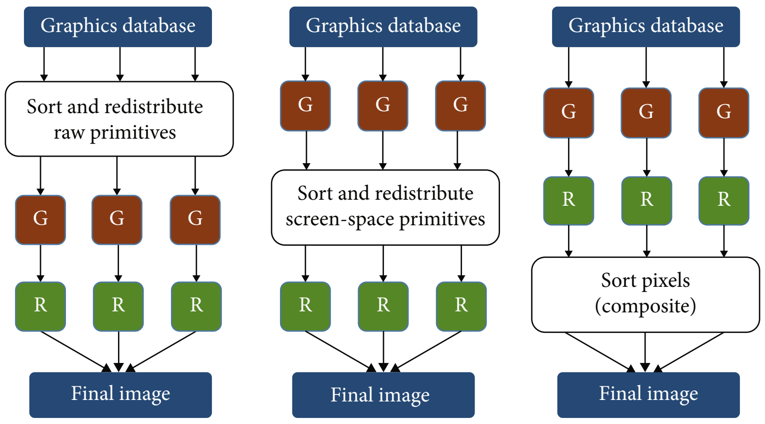
Graphics database
Sort and redistribute raw primitives
G
R
Final image
Sort and redistribute screen-space primitives
Sort pixels (composite)

Figure 3: Sort-first, sort-middle, and sort-last parallel rendering.

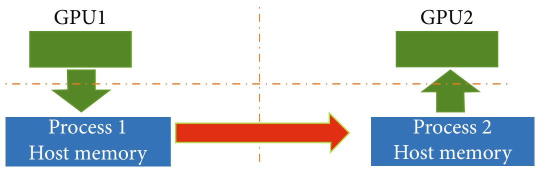

Figure 4: Traditional mode to transfer pixel data between GPUs.

control over the association of a host CPU thread with a particular physical GPU, thereby ensuring optimal use of the compute node NUMA topology. This will be discussed in detail in Section 3.7. Therefore, EGL is the best route for portable and efficient parallel OpenGL rendering on high-performance computing systems.

3.3. Coupling of Simulation Program and Visualization Program. The coupling between simulation and visualization is the key to the in situ program. In the DEM simulation, the calculation program is implemented in CUDA, and the result of each time step is generated in the video memory of GPU. Therefore, the best way to realize in situ visualization is to directly pass the pointer to the result data to the visualization program, rather than transport result data itself between two programs. This not only avoids the data duplication but also avoids the I/O bottlenecks of data transfer between CPU and GPU.

The in situ visualization software takes advantages of CUDA and OpenGL interoperability technology to map OpenGL's workspace to CUDA's memory. The simulation results of CUDA do not need to be sent back to CPU, but are directly used by the OpenGL process, which greatly speed up the entire procedure. The combination of CUDA and OpenGL can be realized by using two types of memory buffer, namely, Pixel Buffer Object (PBO) and Vertex Buffer Object (VBO). The VBO is responsible for the vertex data and passes the result of the CUDA to the OpenGL for rendering. Similarly, the PBO is in charge of the pixel data and passes the rendering result of the OpenGL to the CUDA to complete the image composition.

3.4. Parallel Rendering Architecture. The visualization of DEM simulation has the requirements of large-scale data rendering, high-resolution output, and real-time interaction. So, single processor is far from enough. It is necessary to use parallel rendering.

In order to realize parallel rendering, the parallel rendering architecture should be selected first, which determines the working mechanism of parallel graphics rendering system and is the foundation of it. The essence of the rendering task is to calculate the effect of each primitive on each pixel. Due to the arbitrary nature of the modeling and viewing transformations, a primitive can fall anywhere on (or off) the screen. Thus, we can view rendering as a problem of sorting primitives to the screen. A standard rendering pipeline consists of two principal parts: geometry processing (transformation, clipping, lighting, and so on) and rasterization (scan conversion, shading, and visibility determination). Molnar et al. describe a classification scheme of parallel rendering architecture, which is based on where the sort from object coordinates to screen coordinates occurs [17]. The sort can take place anywhere in the rendering pipeline: during geometry processing (sort-first), between geometry processing and rasterization (sort-middle), or during rasterization (sort-last), as shown in Figure 3.

Due to the large scale of the DEM simulation, if the sort-first or sort-middle parallel rendering architecture is adopted, the assignment of the primitives would take a large amount of computing power. In contract, there is no need to determine the attribution of particles in the sort last architecture, which reduces a lot of operations. In addition, our software is an interactive visualization system, and the number of particles in different regions of the screen is constantly changing, so the load balancing problem is difficult to solve. However, the load balance among processes has been considered in DEM simulation process, and the number of particles in each process is relatively balanced. Therefore, the rendering process of in situ visualization can adopt the same data division method as the simulation program.

3.5. Pixel Data Transmission for Image Composition. As mentioned above, image composition is required in the sort-last parallel rendering architecture after rasterization.

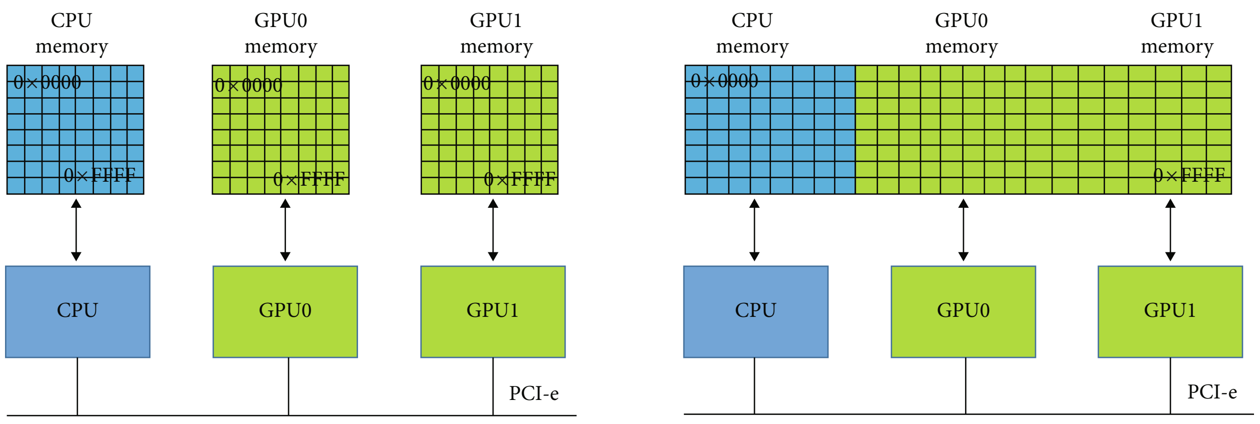

Figure 5: Unified virtual addressing.

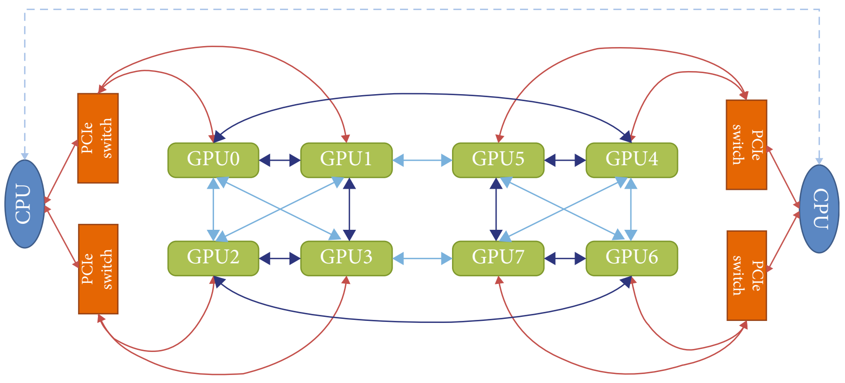

Figure 6: The topological structure of DGX-1 [20].

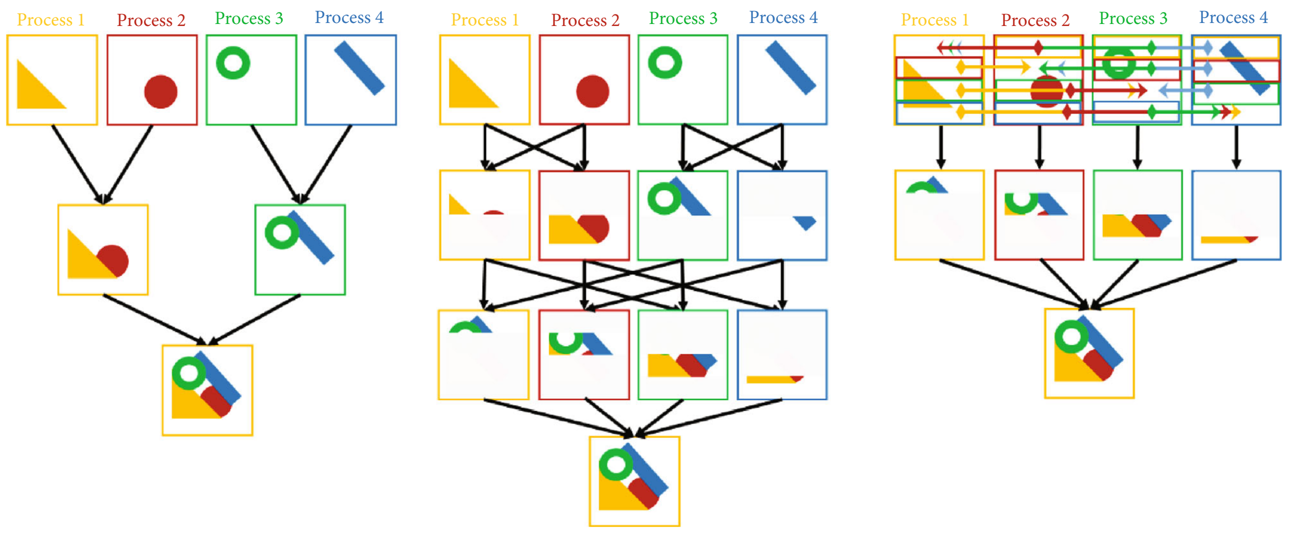

Figure 7: Binary tree, binary swap, and direct send.

Data transmission between rendering processors is the bottleneck in image composition which takes up more than half of the entire rendering time.

Figure 4 shows the process of transferring pixel data from one GPU to another in the image composition stage. The pixel data needs to be transferred from GPU to CPU first, then pixel data is transferred to the specified CPU process through data transfer between MPI processes and finally uploaded to the corresponding GPU. This process requires data transfer between GPU and CPU two times, in addition of data I/O between CPU processes. Such a data transmission mode consumes the bulk of the time of the image composition.

The emergence of unified virtual addressing (UVA) and GPU direct communication technologies developed by

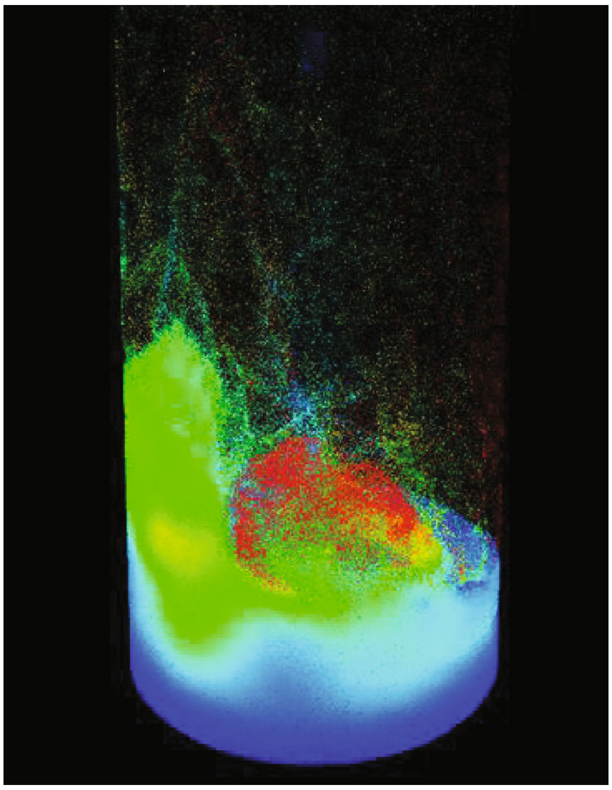

Figure 8: The final image of MTO.

NVIDIA Corporation enables unified addressing of GPU memory and CPU memory in multi-GPU systems, which further enables direct point-to-point access between GPUs [18].

UVA provides a unified virtual addressing between different devices, so that the program of each device can access the data of other devices through a unified pointer. As shown in Figure 5 left, CPU, gpu0, and gpu1 have their own independent memory spaces, which cannot be accessed by each other. However, with UVA, CPU and GPUs have a single memory space. Developers do not need to consider the separate memory spaces of the CPU and GPUs, but integrate them into a unified memory addressing space, which makes GPU direct communication possible.

The image composition algorithm based on GPU direct can not only avoid a large amount of data exchange between the GPU and the CPU but also use the high bandwidth between GPUs and powerful computing power of the GPU efficiently. GPU direct has two communication paths, one is through PCIe lanes, and the other is through NVLink links. Before the emergence of NVLink, multiple GPUs communicate with each other through the PCIe. However, with the increase in computing power, the transmission capabilities of PCIe can no longer meet the needs. To solve this problem, NVIDIA has developed a new interconnect architecture, namely, NVLink. NVLink is a bus and communication protocol which uses a point-to-point structure and serial transmission for the interconnection of multiple graphics processors, which greatly improves the data communication capabilities between GPUs. For example, the maximum communication speed is 16 GB/s if two GPUs are connected through the PCI lanes. In contract, if the GPUs are connected through NVLink links, the speed of direct communication can reach 80 GB/s, which is increased by 5 times. Moreover, GPU direct communication through NVLink frees the path between CPU and GPU, which avoids the communication competition between CPU-GPU and GPU-GPU.

We employ NVLink for GPU direct communication in order to improve the speed of image composition. The pixel data in process 1 does not need to pass through the CPUs, but is directly sent to the GPU of process 2. This mode solves the bottleneck problem of communication in the image composition.

3.6. Pixel Composition with CUDA. Image composition is the key final stage in sort-last parallel rendering which consists of two main stages: pixel transmission and pixel composition. We have optimized the pixel transmission process with NVLink GPU direct communication, but pixel composition is another bottleneck in image composition.

The Z-buffer algorithm is utilized for pixel composition, that is, comparing the depth value of the pixels at the corresponding positions of the two images and taking the pixel with the smaller depth value as the pixel value at that position in the composite image. The conventional method accomplishes the algorithm with CPUs, which deals with each pixel one by one. This option consumes more time and slows down the speed of image composition.

GPU is a highly parallel, multithreaded, many-core processor. Take Tesla V100 as an example, the number of stream processors has reached 5120. So, the GPU is especially well-suited to address problems that can be expressed as data-parallel computations, in which the same program is executed on many data. In Z-buffer algorithm, the calculation of each pixel in the image is independent of each other, so it is very suitable for accelerating with GPUs. As a result, the system takes advantage of CUDA for the pixel composition, making full use of the high concurrency of the GPU to replace the serial computing of the CPU.

3.7. NUMA-Aware Image Compositing Strategy. Image compositing stage is the key to the performance of sort-last rendering system, in which each processor generates images corresponding to its subset of data from the simulation locally, and these images should be combined into a final result after rendering separately. In Sections 3.5 and 3.6, we have optimized pixel data transmission and pixel data composition in the image compositing stage. In addition, it is equally important to choose an appropriate image composition algorithm. The choice of image composition algorithm is not only related to the algorithm itself but also related to the hardware structure. The best performing algorithms are different on different high-performance computing systems. This chapter will consider the characteristics of the algorithm and the hardware structure to choose the most appropriate image composition algorithm.

In general, nonuniform memory access (NUMA) is a computer memory design used in multiprocessing, where

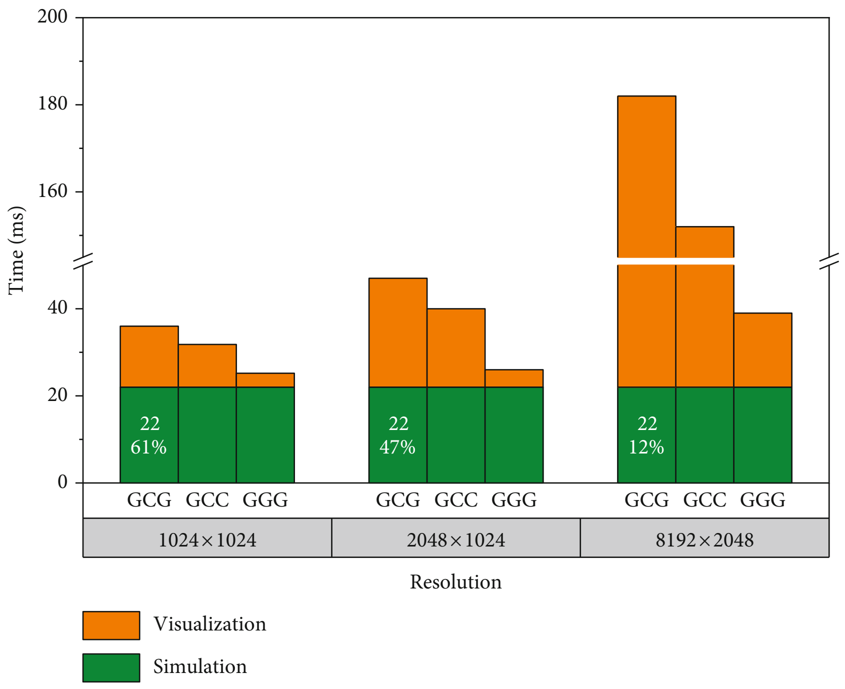

Figure 9: Time comparison of simulation and visualization at different resolution.

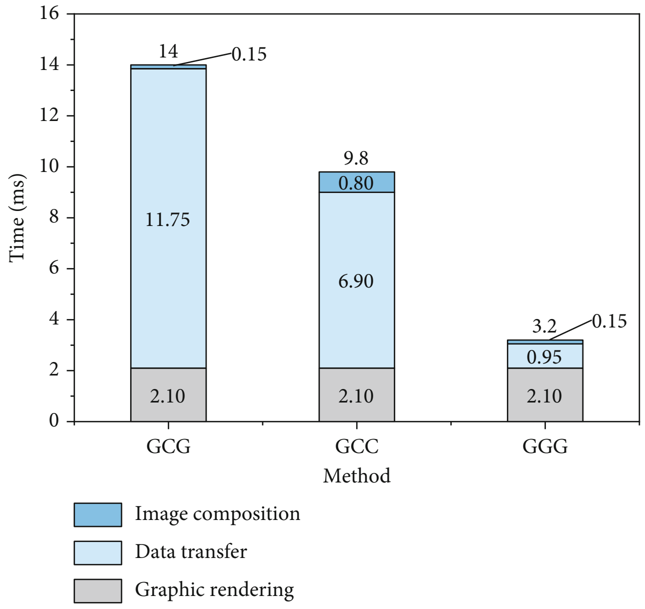

Figure 10: Time comparison in detail at 1024 × 1024.

the memory access time depends on the memory location relative to the processor [19]. Under NUMA, a processor can access its own local memory faster than nonlocal memory. However, the performance asymmetries arising from modern NUMA architectures affect not only CPU memory accesses but also data transfers to and among devices for disk and network I/O and accelerators such as GPUs. In modern computers with multi-GPUs, UVA provides a single virtual memory address space for all memory in the system, and GPU direct enables direct communication between GPUs, both of which makes CPU and GPU access host memory and video memory to form a NUMA memory access system.

We develop the in situ visualization system on dgx-1 platform. The access speed to different memory spaces, such as host memory and video memory, is distinct, so the DGX-1 platform is a NUMA system. The topology of the DGX-1 is shown in Figure 6. There are 8 GPUs in the system, where each GPU is connected to the CPU through a PCIe switch, and two GPUs are directly connected through NVLink. The connection between 8 GPUs uses a hybrid cubic grid topology. The 8 GPUs are divided into two groups. GPUs within groups are connected to each other, while each GPU is connected to only one of the GPUs of the other group. As shown in Figure 6, the GPUs in the left group are connected with each other, but the GPU0 in the left group, for example, is only connected with the GPU4 in the right group. Furthermore, the connections between GPUs are divided into low-speed channels and high-speed channels. In Figure 6, the dark blue arrows indicate Hamiltonian cycles defined on the mesh. Each connection in this ring enables 2× bandwidth compared to other links. In summary, there are three types of GPU communication channels in DGX-1: NVLink high-speed channels, NVLink low-speed channels, and PCIe channels. If we can make full use of the characteristics of NUMA architecture and choose a suitable image parallel compositing algorithm, the parallel image composition time will be reduced.

Currently, there are three parallel image synthesis algorithms: binary tree, binary swap [21], and direct send [22], which is shown in Figure 7.

The subimages rendered by each GPU are distributed in 8 GPUs, and our target is to compose them to the final

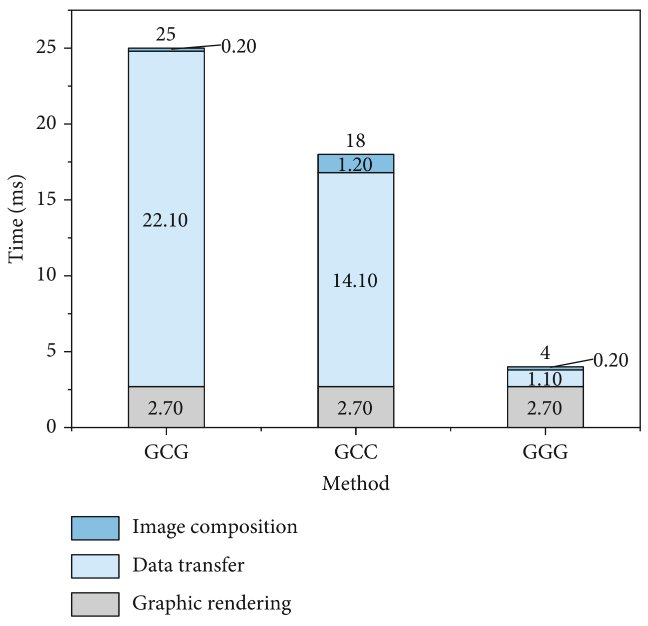

Figure 11: Time comparison in detail at 2048 × 1024.

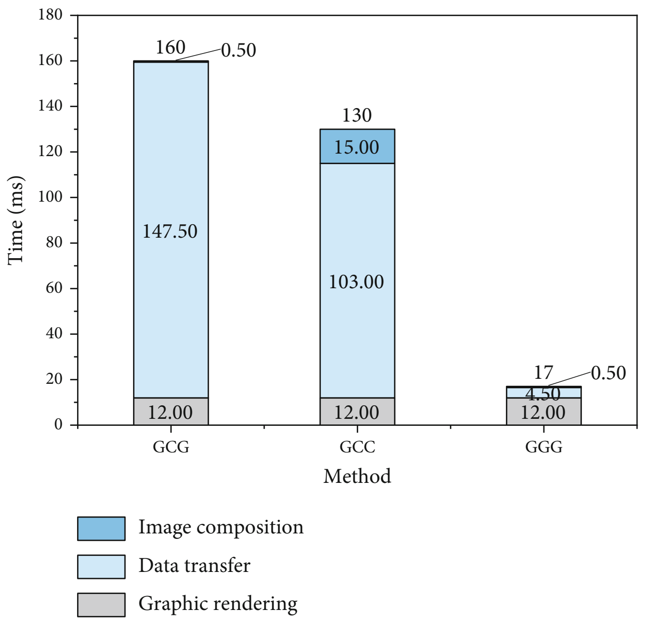

Figure 12: Time comparison in detail at 4096 × 2048.

image. When using binary tree and binary swap algorithm to realize image compositing, there are 8 processes, which are the power of 2 to 3, so there are three stages in the two algorithm. According to the topology of dgx-1 GPUs, there are NVLink links between the GPUs that need to transfer data in each stage. The direct send algorithm has only one stage. Each process divides its own subimage into 8 tiles and sends them to the process responsible for the image space, so there is data transfer between each of the 8 processes. However, it can be found from the topology that not both of the 8 GPUs have NVLink connections. For example, GPU0 in the left group is only connected to GPU4 in the right group and can only communicate with other GPUs through PCIe channels. In addition, multiple processes may send pixel data to the same process at the same time, the direct send compositing strategy has the risk of communication competition. From this perspective, the binary tree and binary swap algorithm are better than direct send. According to the characteristics of the algorithms, half of the binary tree algorithm exits the image synthesis process at each stage. All processes of binary swap and direct send algorithm participate in the whole process, so the utilization of computing resources is higher in the latter two algorithms. In conclusion, the efficiency of binary swap algorithm should be the best on dgx-1.

4. Result and Discussion

4.1. Test Environment. The visualization method described in this paper was tested with a DEM parallel simulation program. The original DEM simulation program remains basically unchanged. For the purpose of in situ visualization, three functions are added to the DEM program: initialize (), visualize (), and finalize (). The initialize () is responsible for the initialization of visualization-related resources, including the creation of OpenGL rendering context, the creation and resource allocation of FBO, VBO, and PBO, and initialization of shared memory. The visualize () is in charge of the main visualization work, which consists of geometry processing, rasterization, and image composition. And the finalize () commits to resource release and destruction.

In this paper, the DEM program simulates the evolution of methanol to olefins (MTO) with 6 million particles. The program is paralleled in 8 MPI processes. The result of visualization is shown in Figure 8. There is gas blowing up at the bottom of the reactor, and the solid particles are fully stirred by the gas.

The performance of our program was tested on DGX-1, which is a single-node multicore CPU multi-GPU platform. The platform is equipped with 2 Intel (R) Xeon (R) E5-2698 (2.20 GHz) CPU, 512 GB memory, and 8 Tesla V100 graphics cards with 32 GB video memory. The graphics cards are connected to each other by 24 NVLink links with the connection topology shown in Figure 5. The operating system on the platform is Ubuntu 16.04.

4.2. Comparison of Total Rendering Time. This test mainly compares the performance of the visualization method adopted in this paper with the other two common methods. Method1 is an optimization method adopted by Fang et al. [15], in which GPU is also used for rendering while CPU takes responsibility for data transmission, and pixel compositing is performed by GPU, called GCG for short. Method2 represents a typical real-time visualization method, in which GPU is used for rendering, but data transmission and pixel compositing are all carried by CPU, called GCC for short. Method3 is the approach proposed by this paper, in which image rendering, data transmission, and image compositing are all completed by GPU, called GGG for short.

We choose 1024 × 1024, 2048 × 1024, and 8192 × 2048 as the generated image resolutions. A represents the

Table 1: Visualization time at different particle sizes and image resolutions.

Number of particle	512×512	1024×1024	2048×2048	4096×4096	8192×8192
60,000	1.5	2.4	5	17.5	59.4
600,000	1.6	2.5	5.2	18	59.5
6,000,000	2.3	3.1	5.5	18.2	59.6

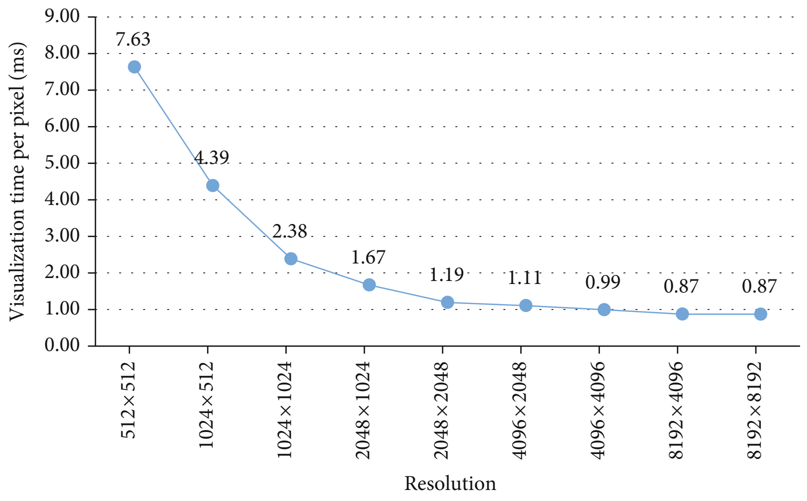

Figure 13: Visualization time per pixel at different resolution.

resolution of a small screen, *B* represents the screen resolution of an ordinary desktop computer, and *C* represents the resolution of a large screen for engineering applications. The image compositing algorithm adopted the binary swap strategy.

As Figure 9 shows, at all three image resolutions, the visualization time of GGG is significantly better than that of GCG and GCC, and it becomes more apparent at higher image resolutions. At 1024×1024 resolution, the visualization time of all three methods is less than the simulation time. If the visualization is performed every 10 time steps, the impact of the visualization on the simulation program can be ignored. But in the case of a resolution of 8192×2048, the visualization time of method1 and method2 has increased to 6-7 times the simulation time. If the interval is also every 10 simulation time steps, the visualization time will account for 37%~42% of the total time, which has seriously affected the simulation.

4.3. *Visualization Time Comparison in Detail.*

In order to analyze the performance of the method this paper presented, the time consumed in the three stages of graphics rendering, data transmission, and image composition at three image resolutions of 1024×1024, 2048×1024, and 8192×2048 were measured, respectively, which is shown in Figures 10–12.

In terms of the total time of visualization, GGG is the best, and the performance is better at higher resolution. From the view of each stage of the visualization, the rendering time of the three methods is the same, and the main difference is the data transmission time and the image composition time. GGG utilizes direct GPU communication to avoid data transfer between GPU and CPU, which makes full use of the bandwidth of GPU point-to-point communication, avoids I/O competition on the PCIe bus, and thus, reduces the time consumption on data transmission. Furthermore, GPU is applied to pixel compositing, which is more effective than CPU.

4.4. *Sensitivity to the Number of Particles.*

This test mainly measures the visualization time of the method proposed in this paper at different resolutions and different particle sizes for testing the sensitivity to the number of particles. We test three numbers of particles: 60,000, 600,000, and 6,000,000, and five image resolutions: 512×512, 1024×1024, 2048×2048, 4096×4096, and 8192×8192.

The main factors affecting the total visualization time are the image resolution and the number of particles. As Table 1 shows, the higher the resolution, the longer the visualization time. And the influence of the number of particles is more obvious when the resolution is small, but it is almost negligible when the resolution is high.

The influence of particle number is mainly reflected in two processes. First, the particle data needs to be transferred from simulation to visualization. And each process renders the particle data into an image. However, in the method adopted in this paper, CUDA and OpenGL interoperation is applied to directly pass the pointer of the simulation result to OpenGL for visualization, thereby avoiding the bottleneck problem of transmitting particle information from CPU to GPU. So, the first impact has been eliminated. Therefore, the effect of particle size on visualization is mainly reflected

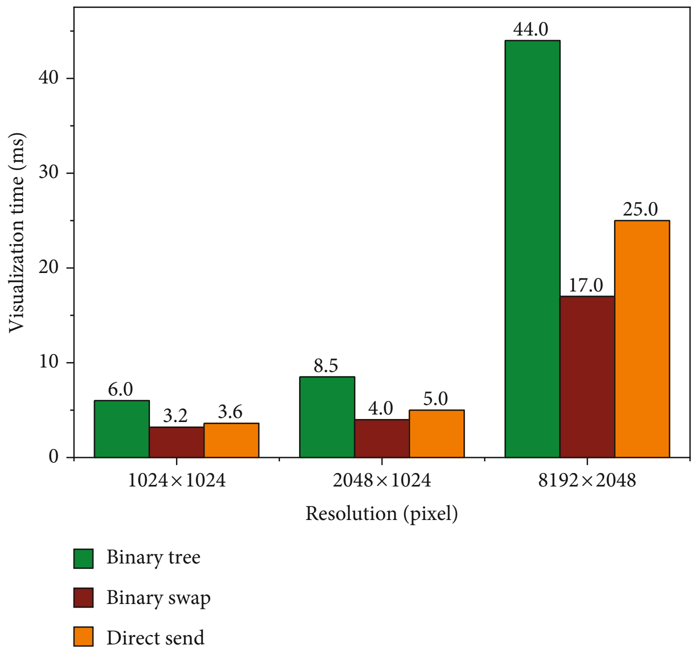

Figure 14: Comparison of different image compositing strategies.

in rendering. When the resolution is high, the time of image composition is much longer than that of rendering, so the effect of particle size on the total visualization time is not obvious.

4.5. Resolution Sensitivity Analysis. This test measures the visualization time of the proposed method at different resolutions. The simulation example employed in the test contains 6 million particles. We choose nine image resolutions: 512 × 512, 1024 × 512, 1024 × 1024, 2048 × 1024, 2048 × 2048, 4096 × 2048, 4096 × 4096, 8192 × 4096, and 8192 × 8192. Then, we calculate the average visualization time per unit resolution; the test results are shown in Figure 13. As the figure shows, the higher the resolution, the shorter the average visualization time per unit pixel. Finally, the curve tends to be stable.

4.6. Performance of Different Image Compositing Strategies. For the methods in this paper, three common image parallel compositing strategies are tested: binary tree, binary swap, and direct send. The test employed a DEM simulation example containing 6 million particles, the program has 8 simulation/visualization processes, and the test program runs on the DGX-1 platform. The total visualization time using three compositing strategies at three resolutions of 1024 × 1024, 2048 × 1024, and 8192 × 2048 was measured.

As Figure 14 shows, the performance of binary swap is the best in all the three resolutions, the binary tree strategy is the second, and direct send is the worst. The DGX-1 platform is a NUMA system, on which the speed of data transmission between GPUs is different. When using the direct send compositing strategy, a GPU needs data from the other 7 processes. Some of these 7 processes transmit data through the NVLink high-speed channel, some transmit data through the NVLink low-speed channel, and some transmit data through PCIe. This leads to waiting between processes, slowing down the image compositing. According to the topology of dgx-1, binary tree and binary swap composition strategy can avoid passing data through the PCIe channel. Taking GPU0 as an example, suppose we use the binary swap algorithm. In the first stage, GPU0 exchanges data with GPU1. In the second stage, it exchanges data with GPU2. In the third stage, it exchanges data with GPU4. It can be seen from the topology that GPU0 has NVLink connection with GPU1~GPU4, but no NVLink connection with GPU5~GPU7. Therefore, in the binary swap algorithm, GPU0 avoids direct communication with GPU without NVLink links. In addition, compared with the binary tree strategy, the processes in the binary swap participate in the image compositing during the whole process, so it is faster and more efficient.

5. Conclusion

This paper discusses a parallel in situ visualization method for large-scale DEM simulations, utilizing NVIDIA's latest communication hardware, NVLink, to realize direct high-speed communication between GPUs, avoiding data transfer between GPU and CPU. And the entire process from data generation to graphics rendering to image compositing is all completed in GPU, minimizing the visualization time.

The method described in this article takes advantage of sort-last parallel rendering architecture, and the data is divided in the same way as the simulation. The results generated by the simulation program are stored in the video memory. The graphics memory address is mapped to OpenGL through CUDA and OpenGL interoperation. OpenGL completes graphics rendering, and then, the rendering results images are mapped to CUDA to implement image composition.

The parallel in situ visualization method proposed in this paper was tested on a DEM parallel simulation program. The test results show that compared with traditional visualization programs, this method has an order of magnitude improvement in time. And this method has good scalability. The time required for visualization is proportional to the image resolution, and the higher the resolution, the shorter the visualization time per unit resolution.

Data Availability

The data that support the findings of this study are available from the corresponding author upon reasonable request.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This work was supported by the grants from the Strategic Priority Research Program of the Chinese Academy of Sciences (XDA21030700) and the Innovation Academy for Green Manufacture, Chinese Academy of Sciences (IAGM-2019-A03).

References

[1] Y. Shigeto and M. Sakai, "Parallel computing of discrete element method on multi-core processors," *Particuology*, vol. 9, no. 4, pp. 398–405, 2011.

[2] X. Huang, C. O'sullivan, K. Hanley, and C. Kwok, "Discrete-element method analysis of the state parameter," *Géotechnique*, vol. 64, no. 12, pp. 954–965, 2014.

[3] G. Liu, W. Xu, Q. Sun, and N. Govender, "Study on the particle breakage of ballast based on a GPU accelerated discrete element method," *Geoscience Frontiers*, vol. 11, no. 2, pp. 461–471, 2020.

[4] S. B. Yeom, E.-S. Ha, M.-S. Kim, S. H. Jeong, S.-J. Hwang, and D. H. Choi, "Application of the discrete element method for manufacturing process simulation in the pharmaceutical industry," *Pharmaceutics*, vol. 11, no. 8, p. 414, 2019.

[5] A. Vyazmensky, D. Stead, D. Elmo, and A. Moss, "Numerical analysis of block caving-induced instability in large open pit slopes: a finite element/discrete element approach," *Rock Mechanics and Rock Engineering*, vol. 43, no. 1, pp. 21–39, 2010.

[6] R. Kostek and A. Munjiza, "Visualization of results received with the discrete element method," *Computational Methods in Science and Technology*, vol. 15, no. 2, pp. 151–160, 2009.

[7] Q. Zhao, A. Lisjak, O. Mahabadi, Q. Liu, and G. Grasselli, "Numerical simulation of hydraulic fracturing and associated microseismicity using finite-discrete element method," *Journal of Rock Mechanics and Geotechnical Engineering*, vol. 6, no. 6, pp. 574–581, 2014.

[8] J. Xu, X. Liu, S. Hu, and W. Ge, "Virtual process engineering on a three-dimensional circulating fluidized bed with multiscale parallel computation," *Journal of Advanced Manufacturing and Processing*, vol. 1, no. 1-2, article e10014, 2019.

[9] H. Yu, C. Wang, R. W. Grout, J. H. Chen, and K.-L. Ma, "In situ visualization for large-scale combustion simulations," *IEEE Computer Graphics and Applications*, vol. 30, no. 3, pp. 45–57, 2010.

[10] M. Dorier, R. Sisneros, T. Peterka, G. Antoniu, and D. Semeraro, "Damaris/viz: a nonintrusive, adaptable and user-friendly in situ visualization framework," in *2013 IEEE Symposium on Large-Scale Data Analysis and Visualization*, pp. 67–75, Atlanta, GA, USA, 2013.

[11] J. Ahrens, S. Jourdain, P. O'Leary, J. Patchett, D. H. Rogers, and M. Petersen, "An image-based approach to extreme scale in situ visualization and analysis," in *SC'14: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 424–434, New Orleans, LA, USA, 2014.

[12] J. J. Camata, V. Silva, P. Valduriez, M. Mattoso, and A. L. Coutinho, "In situ visualization and data analysis for turbidity currents simulation," *Computers & Geosciences*, vol. 110, pp. 23–31, 2018.

[13] N. Fabian et al., "The paraview coprocessing library: a scalable, general purpose in situ visualization library," in *2011 IEEE Symposium on Large Data Analysis and Visualization*, pp. 89–96, Providence, RI, USA, 2011.

[14] T. Kuhlen, R. Pajarola, and K. Zhou, "Parallel in situ coupling of simulation with a fully featured visualization system," in *Proceedings of the 11th Eurographics Conference on Parallel Graphics and Visualization*, vol. 10, pp. 101–109, In Llandudno UK, 2011.

[15] X. Fang, J. Xu, H. Qi, X. He, and W. Ge, "In-situ visualization for GPU-accelerated parallel particle simulation," *Computers and Applied Chemistry*, vol. 28, no. 10, pp. 1234–1238, 2011.

[16] J. E. Stone, P. Messmer, R. Sisneros, and K. Schulten, "High performance molecular visualization: in-situ and parallel rendering with EGL," in *2016 IEEE International Parallel and Distributed Processing Symposium Workshops*, pp. 1014–1023, Chicago, IL, USA, 2016.

[17] S. Molnar, M. Cox, D. Ellsworth, and H. Fuchs, "A sorting classification of parallel rendering," *IEEE Computer Graphics and Applications*, vol. 14, no. 4, pp. 23–32, 1994.

[18] G. Kim, M. Lee, J. Jeong, and J. Kim, "Multi-GPU system design with memory networks," in *47th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 484–495, Cambridge, UK, 2014.

[19] C. Lameter, "NUMA (non-uniform memory access): an overview," *Queue*, vol. 11, no. 7, pp. 40–51, 2013.

[20] C. Christensen, T. Fogal, N. Luehr, and C. Woolley, "Topology-aware image compositing using NVLink," in *2016 IEEE 6th Symposium on Large Data Analysis and Visualization (LDAV)*, pp. 93-94, Baltimore, MD, USA, 2016.

[21] K.-L. Ma, J. S. Painter, C. D. Hansen, and M. F. Krogh, "Parallel volume rendering using binary-swap compositing," *IEEE Computer Graphics and Applications*, vol. 14, no. 4, pp. 59–68, 1994.

[22] K. Moreland, B. Wylie, and C. Pavlakos, "Sort-last parallel rendering for viewing extremely large data sets on tile displays," in *Proceedings IEEE 2001 Symposium on Parallel and Large-Data Visualization and Graphics*, pp. 85–154, San Diego, CA, USA, 2001.