Membrane Layer Method to Separate Simulation and Visualization for Large-scale In-situ Visualizations

Akira Kageyama, Naohisa Sakamoto and Kohei Yamamoto
Department of Computational Science, Kobe University, Kobe 657-8501, Japan

Keywords: In-situ Visualization, High Performance Computing, CFD, Magnetohydrodynamics, MPMD.

Abstract: With the progress of simulation technology, the demand for the in-situ visualization in high-performance computing is increasing. We propose a multiple-program, multiple-data (MPMD) approach to the in-situ visualization to realize interactive in-situ visualizations. We separate processor nodes in a massively parallel computer system into two parts; one devoted only to the simulation and the other devoted only to the visualization. Both the simulation and visualization programs are parallelized by MPI (Message Passing Interface). The number of MPI processes for the visualization is the same or larger than that for the simulation. The multiple camera method that we proposed in our previous study enables interactive analysis of the visualization movies thus generated. The simulation and visualization programs are separated by a layer mimicking a semipermeable membrane. It is semipermeable because information runs in one-way from the simulation to the visualization through the layer, and the layer prevents the simulation program from being affected by visualization program's possible delay. The membrane is implemented as two independent MPI programs which correspond to the front and back faces of it. The four MPI programs (the simulation, visualization, and membrane faces) are executed at once based on an MPMD framework.

1 INTRODUCTION

In contrast to the post-process visualization, the presence of the in-situ visualization is gradually rising (Bethel et al., 2013). A straightforward way to realize the in-situ visualization is to perform the visualization on the same computer system as the simulation [see Fig. 1(a)]. The visualization in this method, however, erodes the computational resources for the simulation such as the memory and processing time. One can avoid the resource erosion by performing the visualization on another computer system [see Fig. 1(b)]. In this approach, one has to send simulation data through the network between the two computer systems whose bandwidth is usually narrow.

We take in this paper another approach to the in-situ visualization, which is a kind of combination of the two styles shown in Figs. 1(a) and (b). In this in-situ visualization, both the simulation and visualization run in the same computer system but on different processor nodes; see Fig. 2.

The motivation of this study stems from the recent trends of the growing number of processor nodes in supercomputer systems. The K computer, for example, has more than 82,000 SPARC64 VIIIfx processor

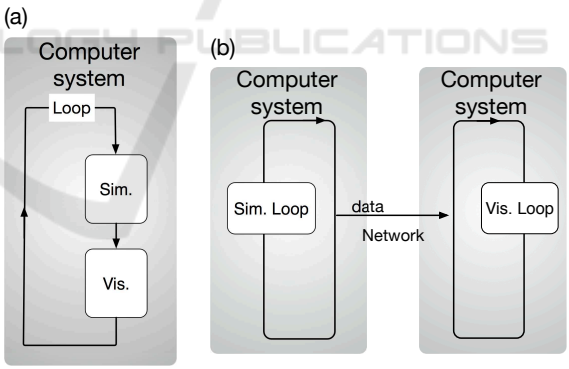

Figure 1: Two styles of in-situ visualization. (a) Visualization is performed on the same computer system as the simulation. (b) Visualization is performed on a different computer system from the simulation. The simulation data to be visualized are transferred through the network connecting the two systems.

nodes. It is difficult for most simulation programs to occupy such a system with full nodes by one job. As the Amdahl's law says (Amdahl, 2007), the speedup of the execution time by the parallelization saturates at the node number around 5000, when just 5% of the code is not parallelized. Since the number of processors in today's supercomputers is excessive for most

Kageyama, A., Sakamoto, N. and Yamamoto, K.
Membrane Layer Method to Separate Simulation and Visualization for Large-scale In-situ Visualizations.
DOI: 10.5220/0006854901060111
In *Proceedings of 8th International Conference on Simulation and Modeling Methodologies, Technologies and Applications (SIMULTECH 2018)*, pages 106-111
ISBN: 978-989-758-323-0

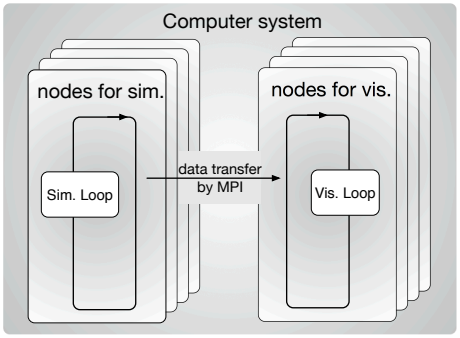

Figure 2: In-situ visualization proposed in this paper. Both the simulation and visualization are performed on the same (parallel) computer system. The same or even larger number of processor nodes is allocated to the visualization than the processor nodes for the simulation.

simulation programs, we have to conceive a new way to make the best use of supercomputer systems, and we believe that the in-situ visualization is a promising candidate.

Recently, a similar in-situ visualization to our method is reported (Buffat et al., 2017). In addition to the existence of the membrane layer described below, a difference between their study and ours is the number of processor nodes devoted to the visualization. The number of visualization nodes in theirs is smaller than the number of nodes used for the simulation. On the other hand, we use the same or even larger number of processor nodes for the visualization compared with the simulation nodes.

2 DESIGN

In the dual-system approach to the in-situ visualization shown in Fig. 1(b), simulation data are sent to the visualization system through the network between the two computer systems. On the other hand, in our method, we do not have to invoke such a (usually slow) data transfer because both the simulation and the visualization programs run on the same computer system: We can take advantage of the broad bandwidth between the processing nodes in a supercomputer system and a customized library of MPI (Message Passing Interface) optimized to the system.

In our in-situ visualization method, the simulation and visualization are almost independent applications. Since there is no tight connection or sync mechanism between them, the execution speeds of the two applications can be very different. Before each execution, one is supposed to allocate adequate numbers of processor nodes for the simulation and visualization in such a way that the two applications run with a balanced pair of loads. In some cases, however, the balance might collapse. Preparing for such a situation, we set a principle that priority is given to the simulation rather than the visualization: The simulation should run without being affected by the visualization, even if the visualization takes tremendously long time to render an image. In other words, under this principle, we want the simulation job to run without waiting for the end of the rendering even if we might lose certain range of image sequence of the output visualization movie.

In practice, a simulation researcher usually has an estimated time for a pure simulation job (without an in-situ visualization) under a given number of parallel processors, say N_s. In our approach to the in-situ visualization, we allocate additional nodes N_v in the same computer system devoted to the visualization. Since the visualization job on these N_v nodes does not affect the simulation job on N_s nodes, the estimated time for the simulation job is basically the same if we can ignore the data transfer time from the simulation to the visualization by making use of an intermediate layer, which will be described in the next section.

3 MEMBRANE LAYER

We place a virtual "semipermeable membrane" between the simulation and the visualization, to fully separate the two programs. See Fig. 3.

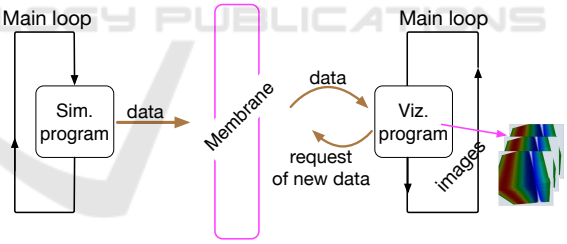

Figure 3: The membrane model proposed in this study. It separates the simulation program and the visualization program. The visualization program is invisible to the simulation program, and vice versa, by the membrane. The simulation program "throw" numerical data to be visualized to the membrane and the membrane is always ready to receive them. The membrane pass the data to the visualization program on demand from the visualization program.

One of the purposes of the membrane is to make the two programs being "invisible" each other. The simulation program, which is an MPI-based parallel program, can communicate only with the membrane: The visualization program is invisible for the simulation program. On the other hand, the visualization program, which is also an MPI-based parallel program, can communicate only with the membrane and the simulation program is invisible. For simulation

researchers, the invisibility is helpful because it releases them from the burden of visualization-related tasks. They just throw simulation data to the (front side of) membrane and the all tasks are automatically done.

The membrane is semipermeable because the information goes in one way. While the simulation data is sent from the simulation program to the visualization program, no information is sent from the visualization to the simulation. (Only one exception is a vital sign of the visualization program: A flag signal is sent to the simulation when the visualization happens to die due to some error.)

The membrane is implemented as two independent MPI programs. They conceptually correspond to the front face (simulation side) and the back face (visualization side) of the membrane. The front and back face programs have N_f and N_b MPI processes, respectively. See Fig. 4.

The purpose of the front face program is to receive the simulation data from the simulation program. Since the front face program is devoted to this task of data receiving, the simulation program can assume that the data transfer is completed without delay, even if other programs (the back face of the membrane and the visualization) are doing their work. By this mechanism, the membrane allows the simulation program to run without being influenced by the visualization, along the "simulation first" principle mentioned above.

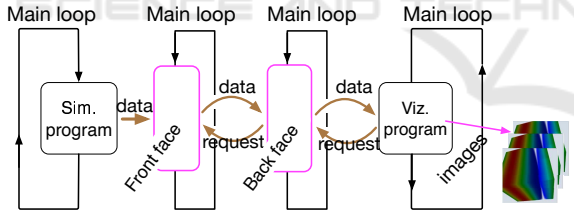

Figure 4: The implementation of the membrane. It consists of two independent MPI programs, which correspond to the front and the back faces of the membrane.

In total, we have four MPI programs to realize the proposed in-situ visualization method: One simulation program, two membrane programs (front & back faces), and one visualization program. We execute the four as an MPMD (Multiple-Program, Multiple-Data) type application. Each program is a standalone, independent MPI program. (They have their own `MPI_Init` and `MPI_Finalize` calls.) By submitting the four MPI executables to a supercomputer at once as an MPMD application, a comprehensive MPI communicator (`MPI_Comm_world`) is automatically constructed. The four programs communicate with each other by the standard MPI functions within the communicator `MPI_Comm_world`.

As the simulation goes on, it sporadically sends data D_n to the front face of the membrane, with D_n being a set of simulation to be visualized. The index $n(=1,2,3,\ldots)$ is a time counter. It is usually much smaller than the simulation's main loop counter ℓ; for example, $\ell = 100\,n$ when one visualizes the data every 100 steps of the simulation. In most of its execution time, the front face program waits for the new data D_n from the simulation program and receives it immediately when it is sent. After completing the transfer of D_n, the simulation program continues its job and the front face program sends D_n to the back face program of the membrane.

On the contrary to the front face, the back face is rather busy. Its mission is to store the latest simulation data, and pass them to the visualization program when they are requested.

When the visualization program has finished its job for the latest visualizing data D_{n-1}, it sends a request signal to the back face of the membrane for the next data. There are three possible cases here.

Case 1: The visualization job (for D_{n-1}) is finished so swiftly that the membrane programs still hold D_{n-1}. In this case, the back face of the membrane does not return any data to the visualization until it receives the new data D_n from the simulation (via the front face of the membrane).

Case 2: The membranes already have the new data D_n when the request signal comes from the visualization. This is the ideal case for the load balance between the simulation and visualization. The back face sends D_n to the visualization in response to the request.

Case 3: The visualization job (for D_{n-1}) is so slow that the simulation job produces the new data D_{n+1} while the visualization (for D_{n-1}) is still running. In this case, the data D_n on the membranes are overwritten by D_{n+1}. Information of D_n will be missing in the visualization movie produced by this in-situ visualization method. This is the defect we intentionally accept, following the "simulation first" principle.

4 INTERACTIVE VIEWING OF IN-SITU VISUALIZATION MOVIES

A problem in the in-situ visualization as a practical tool for the simulation's analytics is that the user cannot interactively change the visualization-related variables, such as the view point, viewing direction, visualization methods and their parameters. To overcome this problem, we proposed a multi-camera ap-

proach (Kageyama and Yamada, 2014). Since the details are reported in detail in our paper, here we briefly summarize it.

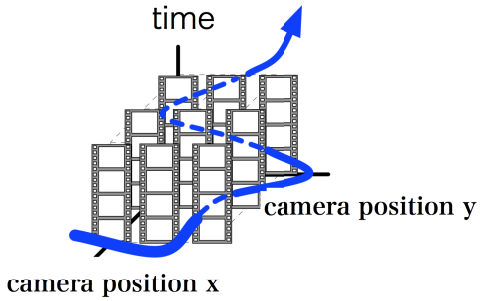

Figure 5: The movie database in our multiple camera method (Kageyama and Yamada, 2014) for interactive in-situ visualization. Applying the in-situ visualization on many cameras scattered in and around the simulation region, we get a movie database, or a "field" of images defined in four-dimensional (4-D) space; 3-D for the camera position and 1-D for time or frame in the movies. The user specifies a "path" in the 4-D space. A specially designed browser extract a sequence of still images on the path and show them as an animation on the PC's screen.

We scatter a lot of virtual cameras, or view points, inside and around the simulation region, applying in-situ visualizations at once with them. As a result of this multiple in-situ visualizations, the same number of movie files as the number of cameras are produced at the end of the simulation. We construct a database by associating the camera position to each movie. As shown in Fig. 5, the database is regarded as a "field" of still images defined in a four dimensional space: The still images are located in a three-dimensional spatial (x, y, z) dimension and one-dimensional temporal (t) dimension. We retrieve the movie database from the supercomputer system to a PC and explore the database with a specially designed database browser. The browser shows image sequences (animations) on a window of a PC's screen. An image sequence is extracted from the database according to a user-specified "path" in the four dimensional space. By this method, we can attain the interactive viewing of in-situ generated visualizations. For example, the user can walk through the simulation space through the PC's interface (the mouse and keyboard) to find a hot spot of interest in the simulation region, and observe the structure or dynamics there in detail from the closest camera to the spot with adequate visualization methods and parameters.

A key point of our interactive in-situ visualization is to have sufficient number of view points or cameras. The larger the number of cameras, the smoother we get in the walk-through. In the present study, we place N_v cameras inside and around the simulation region, and perform N_v in-situ visualizations in parallel, i.e., one visualization per one node.

We use functionalities of the KVS (Kyoto Visualization System) (Sakamoto and Koyamada, 2015) in the visualization program. The KVS is an efficient framework for scientific visualization and provides a rich set of modules for the visualization pipeline, i.e., data importing, filtering, mapping and rendering (Telea, 2014). The KVS uses OpenGL graphics library for hardware accelerated rendering with GPU (Graphics Processing Unit). Therefore, the visualization program requires a software emulation to run on a supercomputer system without GPUs. In this paper, we use the off-screen rendering framework for the KVS (Nonaka et al., 2018) based on OSMesa (Off-Screen Mesa), which has become the de facto standard for the CPU extension of OpenGL.

5 TEST

We test our in-situ visualization method on a supercomputer system Oakforest-PACS (Fujitsu PRIMERGY CX600 M1) which consists of 8208 nodes with one Intel Xeon Phi in each node. The target simulation is our original MHD (magnetohydrodynamics) simulation code, named *cg-mhd*.

We solve the compressible MHD equations in a rectangular region with *cg-mhd*. The 2nd-order finite difference method on the cartesian coordinate system is used for the spatial derivatives in the equations and the 4th-order Runge-Kutta method is used for the temporal integration. The simulation region is divided into multiple sub-rectangular regions and one MPI process is allocated to each sub-region. For tests, we run *cg-mhd* with 32 nodes, or $N_s = 32$.

The original *cg-mhd* code is a parallelized, standalone, simulation code for various studies on MHD phenomena, such as MHD convection, MHD dynamo, MHD self-organization, etc. To incorporate the code into the proposed in-situ visualization, all we need to do is the following two minor modifications.

One is to change the name of the MPI's top-level communicator. The original *cg-mhd* code uses the default MPI communicator `MPI_Comm_world` for the management of all MPI processes in the program. Since the communicator variable `MPI_Comm_world` is used for the inter-program communications in our MPMD application, we rename the top-level communicator in the *cg-mhd* from `MPI_Comm_world` to other name (`comm_sim`).

The second change we made to *cg-mhd* is an obvious one: We add a function call to send simulation data to the front face of the membrane. It is just a call of `MPI_Send`.

While the simulation program (*cg-mhd*) and the two membrane programs (front and back faces) are written in Fortran 2003, the main part of the visualization program, KVS, is written in C++. We implement a wrapper layer in Fortran 2003 over KVS. By making use of the standard `ISO_C_BINDING` function included in Fortran 2003 language specification, we can easily pass the simulation data to KVS. Our software is portable to any supercomputer system as long as a Fortran 2003 compiler is supported.

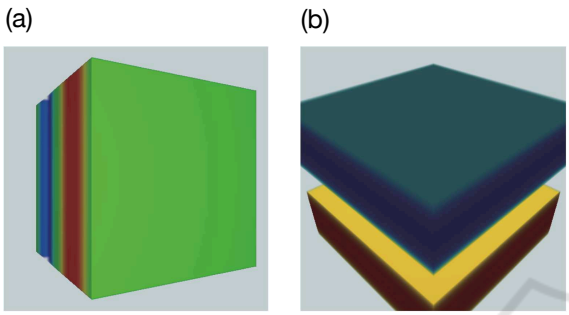

Figure 6: Snapshots of a test in-situ visualization of an MHD (magnetohydrodynamics) simulation in a rectangular region. An Alfvén wave is propagated in a direction and temperature (a) and density (b) fields are visualized with KVS. The two panels (a) and (b) are taken from different camera positions and on different time steps.

As a test, we visualize three scalar fields (pressure, density, and temperature) simulated by *cg-mhd*. There are various possible ways to implement the membrane. Here in this paper, we parallelize them by field-base decomposition. The front face program has three MPI processes, each of which is in charge of receiving one of the scalar fields from the simulation program. The back face of the membrane has the same number (three) of MPI process. Each process in the back face receives the field data from its counterpart in the front face.

The visualization program has 32 MPI processes, or 32 cameras. Fig. 6 shows sample snapshots by two cameras among the 32 cameras. A test MHD wave (Alfvén wave) propagation was successfully simulated by *cg-mhd* and visualized by KVS.

To estimate the effect of the membrane, we compared two in-situ visualization methods, i.e., with and without the membrane in the MHD simulation. The computations were performed on a different computer system (Fujitsu FX-10). The elapsed real time in seconds (averages of five runs) for 10,000 steps of the simulations are shown in Fig. 7. The in-situ visualizations are called every 50 steps in both cases. Thanks to the membrane, the time for the simulation is drastically reduced.

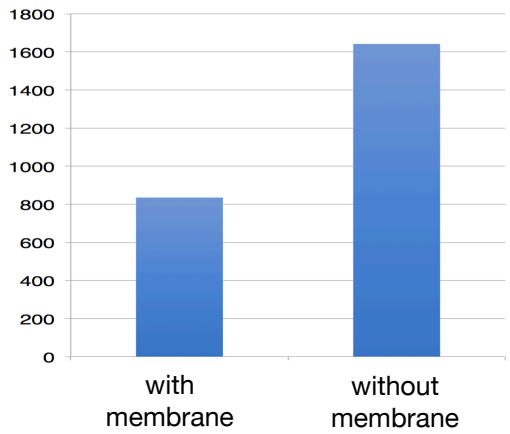

Figure 7: Elapsed time (in seconds) for the MHD simulation with and without the membrane. In-situ visualizations are called every 50 steps. The simulation time is almost halved, owing to the membrane.

6 CONCLUSIONS

When one performs a massively parallel simulation on a supercomputer system, he/she faces two technical problems. One is the difficulty in the data visualization. As the scale of a simulation grows, the post-process visualization, i.e., visualization after simulation, becomes impractical due to the enormous size of the numerical data produced by every simulation. Although the in-situ visualization, i.e., visualization at the same time as the simulation, is free from the enormous numerical data, it slows down the simulation (in the mono-system application) or is baffled by the limited bandwidth of the inter-computer network (in the dual-system application).

The other technical problem in HPC is that it is difficult to increase the number of processor nodes used in a massively parallel computer system for a simulation due to the saturation of parallel efficiently. After a certain amount of (usually painful) time spent for the optimization of a parallel simulation code, it is almost impossible to double the number of using processor nodes under a given set of simulation parameters.

In this paper, we propose a framework for the in-situ visualization for large-scale simulations to resolve the above two technical problems at once. In this framework, we allocate—lavishly—a large number of processor nodes only for the visualization in the same supercomputer system as the simulation. Since the simulation nodes and the visualization nodes are closed in the same computer system, the data can be swiftly transferred from the simulation nodes to the visualization nodes, owing to the optimized network

system and the MPI implementation for the supercomputer system. (This resolves the first difficulty of the visualization in HPC.) In our proposal, the size of the allocated nodes for visualization is very large; it can be the same or even larger than the size of the simulation nodes. (This resolves the second difficulty of the parallelization.)

A simple in-situ visualization loses the user's interactivity such as the viewpoint control, which is very important for effective analysis. We can overcome this problem by making use of the multi-camera method proposed in (Kageyama and Yamada, 2014), in which we apply multiple in-situ visualizations at once from many view points scattered as dense as possible in and around the simulation region. The abundant nodes for the visualization enables us to perform such a lavish visualization style.

Using more nodes for the visualization than for the simulation, our proposal can be regarded as a new way of supercomputer usage: The machine is more for the visualization rather than for the simulation. It would not be unreasonable because, in extremely large-scale HPC, researchers tend to spend more time for the visualization than for the simulation.

We have developed an MPMD application in which four MPI programs are executed at once. The first program is for the simulation. Although we used in this paper an MHD simulation code as a test, this simulation part is interchangeable in our framework.

To enable visualization on CPU-based supercomputer systems without GPU, we have developed a parallel visualization program based on KVS, which is a general-purpose visualization framework written in C++, in its off-screen rendering mode. To receive data sent from simulation program, we implement a wrapper layer for this visualization program in Fortran 2003 over KVS.

We place a membrane-like layer between the simulation and the visualization programs. One of the purposes of the membrane is to fully separate the simulation and visualization. From the simulation code level, we do not "see" the existence of the visualization program: We do not call any visualization-related function from the simulation. All we have to do in the simulation code is to throw numerical data to the membrane. The invisibility reduces the burden of the visualization-related works to simulation researchers in realizing the in-situ visualization on supercomputers. The membrane also prevents the simulation program from being influenced by the visualization program.

The membrane is implemented by two independent MPI programs. The two programs correspond to the front and back faces of the membrane. The front face is to receive numerical data from the simulation and the back face is to pass the data to the visualization.

As a proof-of-concept experiment, we have performed a relatively a small-scale experiment in which 70 MPI processes are invoked in total: 32 for the simulation + 3 for the front face membrane + 3 for the back face membrane + 32 for the visualization, and confirmed that this in-situ visualization framework works.

ACKNOWLEDGEMENTS

This work was supported by JSPS KAKENHI Grant Numbers JP17H02998 and JP17K00169. This work was also supported by SCAT (Support Center for Advanced Telecommunications Technology Research, Foundation), Japan.

REFERENCES

Amdahl, G. M. (2007). Computer Architecture and Amdahl's Law. *IEEE Solid-State Circuits Newsletter*, 12(3).

Bethel, E. W., Childs, H., and Hansen, C. (2013). *High performance visualization : enabling extreme-scale scientific insight*. CRC Press.

Buffat, M., Cadiou, A., Le Penven, L., and Pera, C. (2017). In situ analysis and visualization of massively parallel computations. *International Journal of High Performance Computing Applications*, 31(1):83–90.

Kageyama, A. and Yamada, T. (2014). An approach to exascale visualization: Interactive viewing of in-situ visualization. *Computer Physics Communications*, 185:79–85.

Nonaka, J., Matsuda, M., Shimizu, T., Sakamoto, N., Fujita, M., Onishi, K., Inacio, E. C., Ito, S., Shoji, F., and Ono, K. (2018). A Study on Open Source Software for Large-Scale Data Visualization on SPARC64fx based HPC Systems. In *International Conference on High Performance Computing in Asia-Pacific Region*, pages 278–288.

Sakamoto, N. and Koyamada, K. (2015). KVS: A simple and effective framework for scientific visualization. *J. Adv. Simulation. Sci. Eng.*, 2(1):76–95.

Telea, A. (2014). *Data visualization : principles and practice*. CRC Press, 2nd edition.