DOI: 10.14529/jsfi170302

Performance Evaluation of Runtime Data Exploration Framework based on In-Situ Particle Based Volume Rendering

Takuma Kawamura[1], *Tomoyuki Noda*[1], *Yasuhiro Idomura*[1]

We examine the performance of the in-situ data exploration framework based on the in-situ Particle Based Volume Rendering (In-Situ PBVR) on the latest many-core platform. In-Situ PBVR converts extreme scale volume data into small rendering primitive particle data via parallel Monte-Carlo sampling without costly visibility ordering. This feature avoids severe bottlenecks such as limited memory size per node and significant performance gap between computation and inter-node communication. In addition, remote in-situ data exploration is enabled by asynchronous file-based control sequences, which transfer the small particle data to client PCs, generate view-independent volume rendering images on client PCs, and change visualization parameters at runtime. In-Situ PBVR shows excellent strong scaling with low memory usage up to $\sim$ 100k cores on the Oakforest-PACS, which consists of 8,208 Intel Xeon Phi7250 (Knights Landing) processors. This performance is compatible with the remote in-situ data exploration capability.

Keywords: in-situ visualization, volume rendering, runtime steering, strong scaling, performance evaluation.

Introduction

Recent advances in HPC technology enabled extreme scale simulations at several tens of Peta-FLOPS, while the performance gap between computation and data I/O is enhanced. Because of this severe I/O bottleneck, data handling procedures such as "data output from simulations to storage" and "data input from storage to visualization applications" are becoming very costly, and conventional post processing visualization strategies often fail. In-situ visualization is one of promising solutions to this issue. In this approach, the I/O bottleneck is avoided by combining simulations and visualization applications to visualize simulation data at runtime on the same computing nodes. This requires extreme scale parallel visualization with high scalability at the same level as the simulations.

Computational fluid dynamics (CFD) applications are widely used in various science and engineering fields, and are expected to be one of major applications on future exa-scale systems. Although variety of scientific visualization methods have been developed for CFD applications, visualization methods applicable to massively parallel processing of extreme scale volume data are still limited. Therefore, volume rendering methods for in-situ approaches should be carefully selected from the viewpoint of massively parallel processing and flexible visual analytics. Volume rendering is one of scalable visualization methods, which are suitable especially for CFD applications. In addition, in Ref. [4], it was shown that the volume rendering makes visual analytics flexible by extending the definition of transfer functions (TFs) into multi-dimension. Multi-dimensional transfer functions (MDTFs) generate not only three dimensional (3D) volume rendering, but also iso-surfaces, slice planes, image cropping, and image composition.

Another important requirement is interactivity of in-situ visualization. In the conventional in-situ visualization, visualization parameters such as viewpoint and TFs are prescribed before batch processing. But, such prescribed parameters often generate undesirable images. This problem obviously leads to the loss of computational resources, and is fatal at extreme scale. In order to extract important features from extreme scale data, one needs to change visualization

[1]Japan Atomic Energy Agency, Kashiwa, Chiba, Japan

parameters repeatedly in an interactive manner. To this end, the so-called runtime visualization approaches [18] [19] are becoming more important in modern visual analytics. In the runtime visualization approaches, runtime steering of visualization parameters enables interactive in-situ data exploration, which minimizes such a visualization failure.

However, it is not so clear whether one can design such interactive in-situ visualization frameworks based on the conventional volume rendering algorithms, which may suffer from the following issues. Firstly, the conventional algorithms of volume rendering require large rendering primitive data (e.g., splatting kernels, cells, and polygons), which can be comparable or even larger than the original volume data. Since exa-scale simulations have to overcome severe constraint on the memory size per node, this feature is a critical issue. Secondly, even if one can store such large rendering primitive data, calculation of semi-transparency attribute requires costly collective communication for sorting or searching of sub-images, which could lead to performance degradation and prevent strong scaling. Thirdly, the conventional volume rendering generates view-dependent images, and thus, interactive in-situ visualization becomes extremely costly. Existing parallel visualization libraries such as VTK-m [12], VisIt [1], and ParaView [2] support in-situ visualization based on the conventional volume rendering algorithms. However, the above bottlenecks were not resolved yet. Therefore, it is important to construct in-situ visualization frameworks based on a new volume rendering algorithm, which can resolve the above issues, and demonstrate its performance on the latest many-core platforms.

In this work, we address these issues by using the In-Situ PBVR [6], which was developed using visualization algorithm PBVR [17] [15], whose rendering primitive can be processed view-independent and data size is controllable by image quality. This framework converts extreme scale volume data into small view-independent rendering primitive (particle) data via Monte-Carlo sampling, and transfers it to client PCs, where it is rendered at interactive frame rate. Since the particle generation process does not require visibility ordering, and thus, inter-node communication, this framework works only with the embarrassingly parallel Monte-Carlo sampling, which can be computed on the same massively parallel nodes as the simulations with much less memory usage. Therefore, it is expected that this framework does not suffer from the aforementioned bottlenecks. In addition, the In-Situ PBVR supports flexible MDTF design, which is essential for visualizing complicated multivariate volume data generated from extreme scale simulations with high fidelity and reality. We examine the performance of the In-Situ PBVR, which is coupled to a multi-phase multi-component thermal-hydraulic CFD code, on the Oakforest-PACS, which consists of 8,208 Intel Xeon Phi7250 (Knights Landing) processors.

1. Related Works

Volume rendering requires visibility ordering for alpha blending over the entire volume data, which leads to high calculation and memory costs. In realizing interactive visual analytics, the development of parallel volume rendering with high frame rate and low memory cost is of critical importance. So far, such parallel volume rendering methods have been developed on various parallel platforms, and they are categorized into two types: methods optimized for multi-threaded accelerators such as Xeon Phi and GPU, and methods optimized on massively parallel CPU platforms. Since massively parallel accelerator platforms are one of promising approaches towards exa-scale machines, the former is attracting much attention in recent works.

On massively parallel platforms, most parallel volume rendering methods adopts the so-called sort-last approach, in order to efficiently use distributed memory. While this approach is

suitable for processing subdivided volume data on distributed memory, visibility ordering for image composition requires costly inter-node communication. To resolve this issue, significant efforts were made in former works. By optimizing time series processing pipeline and reducing the cost of image composition using improved direct send method, Peterka et al. [13] improved the performance of parallel volume rendering using 4,096 cores of IBM BlueGene/P and processed about 644 million grids (864^3) of volume data at the frame rate of $\sim$ 0.3 frame per second (fps). However, in this work, the cost of image composition rose exponentially, and exceeded the cost of rendering at 4,096 cores. Perterka et al. further optimized this approach using a faster compositing algorithm Radix-k and extended the numerical experiment up to 32k cores [14]. In the case of about 1.4 billion grids (1120^3), the performance acceleration was saturated at 16k cores, while in the case of about 90 billion grids (4480^3), the scalability was extended up to 32k cores. Howison et al. [3] optimized the volume rendering using Levoy's method [11] with a hybrid MPI/OpenMP parallelization model on Cray XT5. They conducted scaling tests using 98 billion grids (4608^3), and the performance was scaled up to 216k cores.

There are several on-going efforts to develop parallel volume rendering methods for accelerators. Embree [21] is a photorealistic ray-tracer, which consists of a set of low-level kernels for multiple platforms, and has a simple API to port its kernels. OSPRay [20] is a multi-platform ray-tracing framework for scientific visualization on GPUs and multiple CPU architectures with varying SIMD widths, and is integrated into VisIT and ParaView. BnsView [8] is a molecular visualization framework which delivers fast volume rendering and ball-and-stick ray casting on Xeon Phi and is implemented in a SPMD language. Larsen et. al. [9] presented a method for ray-tracing consisting entirely of data parallel operators such as map, gather, scatter, reduce, and scan, which are optimized for CPU and GPU. VTK-m library [12] employs Larsen's algorithm for the ray-tracer and supports in-situ visualization mode on various multithreaded devices such as CPU, GPU, and MIC. In our previous work [5], PBVR was implemented on GPU, and an order of magnitude speedup was achieved compared with CPU rendering.

In terms of interactive data exploration, transfer of the original data requires a dedicated visualization cluster and a sustained network bandwidth equal to the solution output rate in order to support the so-called runtime visualization [10]. Tu et al. [19] proposed an online approach with image delivery and demonstrated efficient monitoring of tera-scale earthquake simulations running on supercomputers with thousands of processors. Over a wide-area network, they were able to interactively change visualization parameters to visually monitor simulation runs [18]. This kind of online approach was also extended to lightweight in-situ application called Strawman [10], which supports multiple programming languages and data models with refined system interface. In Ref. [6], we proposed a runtime visualization framework, In-Situ PBVR, which enables an online approach by transferring lightweight volume rendering primitive data.

2. In-situ PBVR

PBVR comprises two processes: particle generation and particle projection [7] (see Fig. 1). The first process constructs a particle density function based on physical variables, and generates particles for representing volume data. In Fig. 1, the left shows the cell-by-cell particle generation using Monte-Carlo sampling. The particles are randomly located in each cell and its color is given by interpolated physical values.

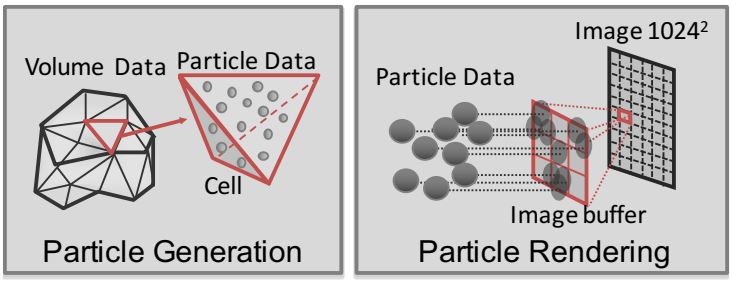

Figure 1. Image generation process of PBVR

The second process projects particles onto an image plane, and the particles are stored in the corresponding particle buffer, and final color and brightness values are synthesized from the illuminant particles in the buffer. In Fig. 1, the right shows the particle projection onto screen. The screen has the particle buffer which is consist of color and depth buffers, and each pixel is subdivided to sub-pixels. After the projection, the color and depth information in sub-pixels are synthesized to calculate the final pixel color.

PBVR generates particles by referring to the particle density function, which represents a number of particles in a unit volume. The particle density function is derived from a user-specified MDTF. The particles are generated in a cell-by-cell manner. In each cell, the locations of the particles are calculated via Monte Carlo sampling to avoid lattice patterns. The number of particles in each cell is calculated by volume integration of the particle density function. The

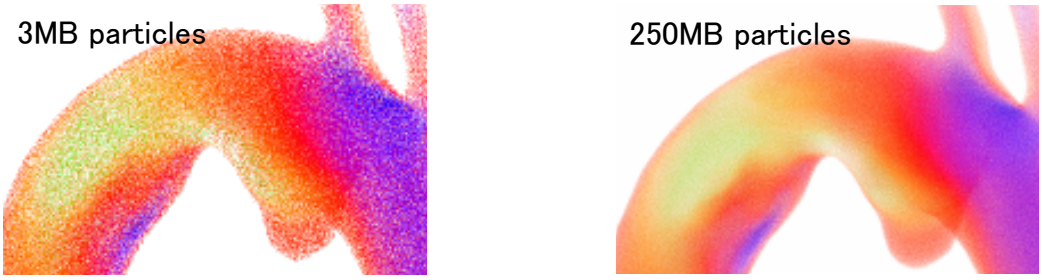

Figure 2. The PBVR image quality and the number of particles

size of particle data often becomes several orders of magnitude smaller than that of the original simulation data, and is determined by a flexible level of details (LOD) control. The left and right of Fig. 2 shows coarse and fine images with $1,024 \times 1,024$ pixels generated using $\sim$ 3MB particle data ($\sim$ 0.1M particles) and $\sim$ 250MB particle data ($\sim$ 9.3M particles), respectively. Here, a single particle has 27MB data consisting of particle position (4Byte$\times$3), RGB color (1Byte$\times$3), and normal vector (4Byte$\times$3). One can control the image resolution by varying the number of particles depending on the purpose of visualization. For example, light weight particle data may be useful for interactive data exploration via narrow bandwidth network. On the other hand, heavy particle data may be desirable for generating high fidelity images, once MDTFs are properly designed through in-situ data exploration.

In-situ PBVR consists of three main components: "Sampler" connected to the simulation solver on computing nodes, "Daemon" launched on an interactive node, and "Viewer" providing the multivariate volume renderer and GUI to design MDTFs on client PC. Particle generation and projection processes are computed by Sampler and Viewer respectively, and Daemon controls data transfer between them. Fig. 3 shows the whole design of the developed framework.

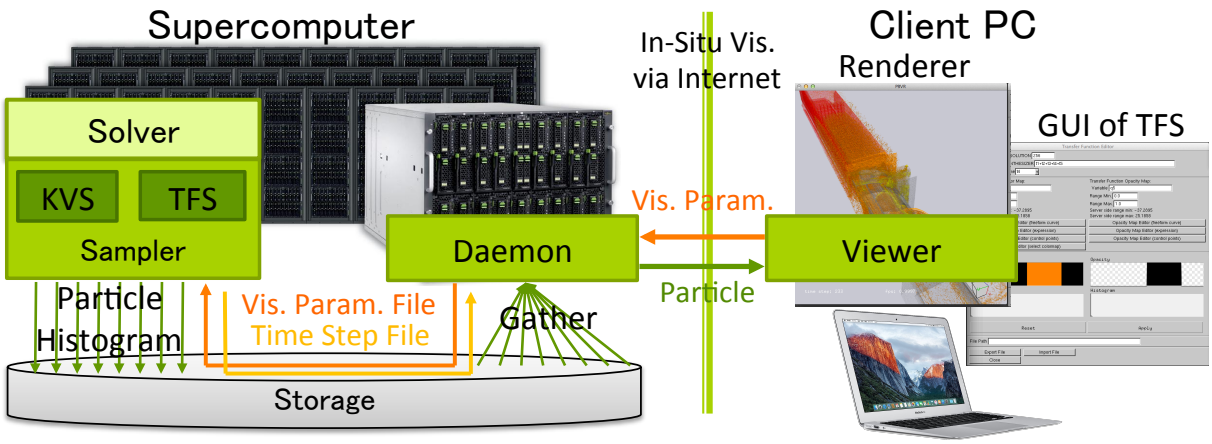

Figure 3. Three components of In-situ PBVR. From left to right, Sampler, Daemon, and Viewer

2.1. Viewer

Viewer on client PCs receives particle files, histogram files, and time step files from Daemon. Here, the particle file contains particle data, the histogram file has the information on distribution of (synthesized) variables, and the time step file contains other control parameters. Viewer renders particle data by projecting particles onto an image plane. Since the particle data is view-independent, one can easily change viewpoints, shading types, and light directions to explore the volume data. Since the maximum particle data size is several hundreds MB, Viewer works on PCs with small memory, and its thread parallel implementation with OpenMP enables interactive data exploration at interactive frame rate [5].

Another important interactive feature is flexible design environment of MDTFs supported by Transfer Function Synthesizer (TFS) [4]. Based on algebraic expressions, TFS generates new variables at runtime following definitions given as functions of existing multiple variables, coordinates, and time. The definitions of new variables are stored as character data of algebraic expressions. 1D TFs, which defines the color and the opacity as a function of a single variable, are designed using GUIs based on algebraic expressions, control points, or freeform curves. 1D TFs are stored either as character data or as data tables. Finally, algebraic synthesis of multiple 1D TFs gives a MDTF, which defines the color and the opacity independently as functions of multiple variables. The definition of MDTF is stored as character data of algebraic expressions. Together with other visualization parameters such as the number of particles, the above MDTF information is transferred to Daemon, and stored in a visualization parameter file.

2.2. Sampler

Sampler is written in C++ based on a visualization library KVS [16], which supports various data structures for visual programing. The routines of Sampler are wrapped in C interface, which can be called also from Fortran programs. Sampler is parallelized using a hybrid MPI and OpenMP parallelizaiton model, where MPI parallelization follows domain decomposition of the simulations, while OpenMP is applied to cell-by-cell parallelization in each subdomain. This parallelization strategy makes the API of Sampler quite simple. Sampler is easily coupled to the simulation by calling the following function from each MPI process at each time step.

```
void generate_particles( Type** subvolume, Param* parameters )
```

Here, `subvolume` is data array of resulting multivariate volume data in each subdomain, and `parameters` involve the information of volume data size and MPI parameters such as the rank ID, the number of MPI processes, and MPI communicators. A pseudo code of `void generate_particles` is shown in Code 1, where `parameters` are visualization parameters including MDTF read from the visualization parameter file, `multi_dim_tf` gives the color and the opacity by processing MDTF, `interpolator` calculates trilinear scalar interpolation at given point, `gradient` defines the normal vector from the gradient of the opacity around given point, `conv_opa2dens` converts the opacity to the normalized particle density [7], `particle_object` is a dynamic array for generated particles, which have point (particle position), color (8bit RGB), and gradient (normal vector), following the KVS format. Since the number of particles in each cell varies depending on multivariate volume data and MDTFs, the particle generation is parallelized in a cell-by-cell manner with dynamic scheduling. In each cell, the number of particles is computed, particles are generated following the opacity or the particle density via Monte-Carlo sampling, and the color is computed from MDTFs. The most expensive part of the particle generation is `multi_dim_tf`, in which algebraic expressions of MDTFs, which is given as character data, are interpreted and expanded into a tree structure by using a function purser, and then, volume data synthesis, 1D TF computation, and MDTF synthesis are computed sequentially. In order to reduce the cost of `multi_dim_tf`, we synthesize volume data before starting the particle generation, and keep a copy of the volume data of opacity, which is used for calculation of the normal vector given by the gradient of the opacity.

Although Sampler can be processed using the original volume data on the memory of computing nodes, we produce a copy of the volume data before the particle generation, so as not to affect the simulation. As a result, the current Sampler consumes memory space for a copy of the original multivariate volume data, the volume data of opacity, and generated particles.

When the visualization parameter file is updated, visualization parameters are distributed from the master process, and the corresponding histogram data computed by using the new visualization parameters are reduced to the master node. Apart from this initialization process, the particle generation is processed without any MPI communication, and the generated particle data is written from each process using POSIX I/O in an asynchronous manner.

2.3. Daemon

Daemon is operated on an interactive processing node or on a login node, which can access the storage and are connected to the internet, and controls the following data transfer between Viewer and Sampler.

- Sampler ← Viewer: visualization parameter file
- Sampler → Viewer: particle file, histogram file, time step file

Daemon interacts with Viewer via socket communications (through ssh tunnel), while it interacts with Sampler via asynchronous file-based control sequences. When new visualization parameters are transferred from Viewer, Daemon writes them to a new visualization parameter file. As shown in Code 1, when Sampler is launched from the simulation, it detects the new parameter file, and the master process reads new visualization parameters and distributes them via MPI_Bcast. In principle, this process may be constructed without MPI_Bcast, provided that all processes can read the same visualization parameter file. However, depending on the timing of file update, all computing nodes may not access the same file, and the coherence of visualization parameters may break down. To avoid such a failure, the above file control sequence is designed. Even

Algorithm 1 `generate_particles` on each MPI rank

```
1: MPI_Barrier
2: if Open( visualization_parameter_file ) == true then
3:     if mpi_rank == 0 then
4:         Read( visualization_parameter_file ) → parameters
5:         Rename( visualization_parameter_file → old_parameter_file )
6:     end if
7:     MPI_Bcast( parameters )
8:     MPI_Reduce( histograms )
9:     Write( histogram_file ) ← histograms
10: end if
11: #pragma omp for schedule( dynamic ) nowait
12: for each cell do
13:     scalars ← interpolator( gravity center of cell )
14:     opacity ← multi_dim_tf( scalars )
15:     density ← conv_opa2dens( opacity, visualization parameters )
16:     num_particles ← density * volume of cell
17:     c = 0
18:     while c < num_particles do
19:         point ← randomly sampled point in each cell.
20:         scalars ← interpolator( point )
21:         opacity ← multi_dim_tf( scalars )
22:         density ← conv_opa2dens( opacity, visualization parameters )
23:         r ← random number from 0 to 1.
24:         if density > r then
25:             vector ← gradient( point )
26:             color ← multi_dim_tf( scalars )
27:             particle_object ← ( point, color, vector )
28:             c++
29:         end if
30:     end while
31: end for
32: Write( particle_file ) ← particle_object
33: if mpi_rank == 0 then
34:     Write( time_step_file )
35: end if
```

with this control sequence, when Daemon and Sampler access the visualization parameter file simultaneously and the update of the visualization parameter file is not completed, Sampler often fails because of incomplete visualization parameters. To avoid this error, the exclusive control of write and read sequences of the visualization parameter file is designed, so that Daemon writes "END_OF_FILE" statement at the end of the visualization parameter file, and before Sampler reads the visualization parameters, it waits until this statement is detected in the file.

On the other hand, when the current time step in the time step file is updated by the master process of Sampler, Daemon starts to gather the particle files. This file I/O is thread parallelized via OpenMP. Here, their file names include the information of the current time step, the rank ID, and the number of MPI processes. By using this information, Daemon detects the completion of the file gather, before it launches transfer of the merged particle data to Viewer.

3. Performance Evaluation

In-Situ PBVR was originally developed on the ICEX, which consists of 2,510 Intel Xeon E5-2680v3 connected via the Infiniband FDR interconnect, and its performance showed excellent strong scaling up to 3,456 cores [6]. In order to estimate its performance on the latest many core platforms, which have significantly different hardware characteristics from the ICEX, we conduct a numerical experiment on the Oakforest-PACS, in which 8,208 computing nodes with Intel Xeon Phi7250 (Knights Landing) are connected via the Intel Omni-Path Host Fabric Interface.

Sampler is connected to the JUPITER code [22], which computes relocation behavior of molten debris in reactor pressure vessels based on thermal-hydraulic equations and multiphase simulation models. The code is based on structured grids with 3D domain decomposition, and is highly parallelized using a hybrid MPI and OpenMP parallelization model. A typical simulation duration of the JUPITER code is ∼3,000,000 time steps, and visualization is processed at every 1,000 steps. However, in this numerical experiment, Sampler is called at every time steps. A strong scaling test is performed with the fixed problem size of $240 \times 240 \times 1,920 \sim 10^8$ grids. Sampler generates $\sim 10^7$ particles (∼250MB), which is typically used for high quality image with $1,024 \times 1,024$ pixels. In the strong scaling test, the number of MPI processes per node is fixed to 4, and the number of threads per MPI process is chosen to be 16. Therefore, 64 cores per node are used without hyper-threading, while the Oakforest-PACS has 68 cores per node. The multi-channel dynamic random access memory (MCDRAM) is used in a cache mode. In the experiment, the number of nodes (cores) is increased from 24 (1,536) to 1,536 (98,304). The computational cost is measured over 20 time steps, in which visualization parameters are updated once.

The strong scaling test is summarized in Tab. 1 and in Fig. 4. In Tab. 1, "Solver" and "Sampler" mean the costs of the JUPITER code and the In-Situ PBVR, respectively. "Write" and "Update" are respectively the costs of updating visualization parameters and writing particle data, which are included in Sampler. The result shows that the cost of Sampler is suppressed between $\sim 10\%$ and $\sim 45\%$ of the JUPITER code (Solver). If one calls Sampler less frequently, this cost is negligibly small. Both Solver and Sampler scale up to $\sim$ 100k cores. However, at 1,536 nodes, the problem size per thread is reduced to $15 \times 15 \times 5$, which is almost the upper limit of strong scaling, and Solver suffers from significant performance degradation. As a result, the acceleration of Solver between 24 and 1,536 nodes (the peak performance ratio of $\times 64$) is limited to $\times 3.9$. Although Sampler shows better acceleration ratio $\times 9.5$, it is still far from the peak performance ratio. This performance degradation may be attributed to the cost of

writing the particle data, and the load imbalance inherent to the particle distribution reflecting the (synthesized) multivariate volume data. The overhead of file I/O increases from $\sim 1\%$ to $\sim 79\%$ of the total Sampler cost. It is noted that the update of visualization parameters includes MPI_Bcast and MPI_Reduce, and thus, the overhead of synchronization among all nodes was anticipated. However, the result shows very small impact from the update.

Another important result is the memory usage summarized in Tab. 2 where "Solver" and "Sampler" mean the memory usage of the JUPITER code and the In-Situ PBVR, respectively. The memory usage of Solver is based on the maximum memory size reported from the job scheduler, while the memory usage of Sampler is calculated from the difference of the memory usage before and after the memory allocation for the particle generation. Here, the instantaneous memory usage is obtained from "/proc/[process ID]/stat" file. Even with the fixed problem size, the memory usage of Solver shows explosive growth due to increasing halo regions and MPI buffers. On the other hand, the memory usage of Sampler shows moderate growth, and the memory usage per MPI process is suppressed between ~ 3.3MB and ~ 92MB. This extreme low memory usage is an important advantage of In-Situ PBVR.

The particle data was transferred to client PCs via socket communication, and the multivariate volume data was rendered on Viewer, in which the viewpoint is changed at about 10 fps.

Table 1. Distribution of computational costs per time step observed in the numerical experiment on Oakforest-PACS

Nodes	24	96	384	1,536
Cores	1536	6,144	24,576	98,304
Solver [sec/step]	26.7	10.3	7.3	6.8
Sampler [sec/step]	7.6	2.8	1.0	0.8
Write [sec/step]	0.078	0.064	0.124	0.645
Update [sec/step]	0.05	0.03	0.04	0.03

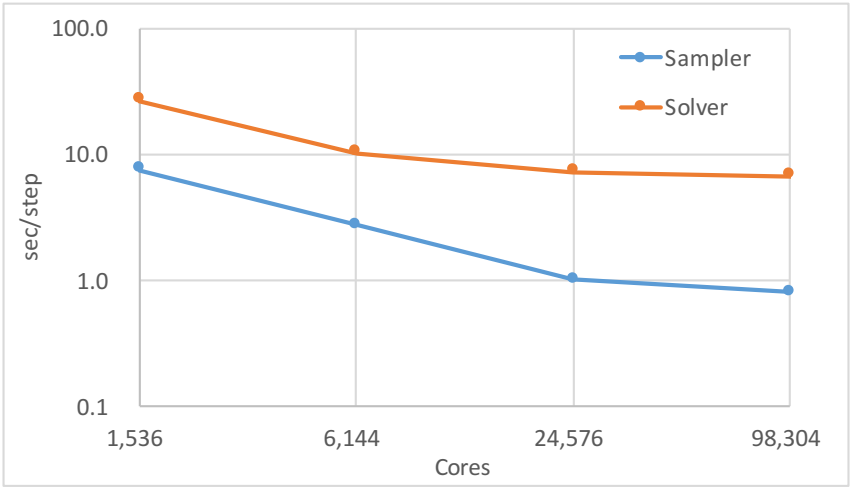

Figure 4. Strong scaling of the JUPITER code (Solver) and the In-Situ PBVR (Sampler) on Oakforest-PACS

Table 2. Total memory consumption observed in the numerical experiment on Oakforest-PACS

Cores	1536	6,144	24,576	98304
Solver [GB]	106.7	135.0	256.7	773.2
Sampler [GB]	8.9	9.5	11.6	20.4

Conclusion

We have examined the performance of the In-Situ PBVR framework on the latest many-core platform based on Knights Landing processors. The proposed framework is designed to process parallel in-situ visualization with the minimum memory usage and to enable interactive in-situ data exploration. Sampler is parallelized by MPI for each decomposed subdomain and by OpenMP for each cell, respectively. The result of strong scaling test shows that Sampler performance scales up to $\sim$ 100k cores with extremely low memory usage. These are promising features for future exa-scale in-situ visualization frameworks. Interactive data exploration is realized by the following two features. Firstly, we designed asynchronous file-based control sequences for interactive update of visualization parameters between Sampler and Daemon, so that the performance of Sampler is not affected by the interactive procedure. Secondly, In-Situ PBVR is based on view-independent rendering primitives, which are given as relatively small particle data. The particle data is easily transferred to remote PCs via socket communication, and is rendered at interactive frame rate.

Acknowledgments

This work is partially supported by "Joint Usage/Research Center for Interdisciplinary Large-scale Information Infrastructures" and "High Performance Computing Infrastructure" in Japan. This research was supported by MEXT as "Post-K priority issue No.6: Development of Innovative Clean Energy".

References

1. Visit user's manual. Tech. rep., Lawlence Livermore National Laboratory (2005), `https://wci.llnl.gov/simulation/computer-codes/visit/manuals`

2. Henderso, A.: Paraview guide, a parallel visualization application. Tech. rep., Kitware Inc. (2005), `http://www.paraview.org`

3. Howison, M., Bethel, E.W., Childs, H.: Hybrid parallelism for volume rendering on large-, multi-, and many-core systems. IEEE Trans. Vis. Comput. Graph. 18(1), 17–29 (2012), DOI: 10.1109/TVCG.2011.24

4. Kawamura, T., Idomura, Y., Miyamura, H., Takemiya, H.: Algebraic design of multi-dimensional transfer function using transfer function synthesizer. Journal of Visualization

20(1), 151–162 (Feb 2017), DOI: 10.1007/s12650-016-0387-1

5. Kawamura, T., Idomura, Y., Miyamura, H., Takemiya, H., Sakamoto, N., Koyamada, K.: Remote visualization system based on particle based volume rendering. In: Proceedings of the conference on VDA. Visualization and Data Analysis 2015, SPIE (2015), DOI: 10.1117/12.2083501

6. Kawamura, T., Noda, T., Idomura, Y.: In-situ visual exploration of multivariate volume data based on particle based volume rendering. In: Proceedings of the 2Nd Workshop on In Situ Infrastructures for Enabling Extreme-scale Analysis and Visualization. pp. 18–22. ISAV '16, IEEE Press, Piscataway, NJ, USA (2016), DOI: 10.1109/ISAV.2016.9

7. Kawamura, T., Sakamoto, N., Koyamada, K.: Level-of-detail rendering of a large-scale irregular volume dataset using particles. JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY 25(5), 905–915 (2010), DOI: 10.1007/s11390-010-9375-4

8. Knoll, A., Wald, I., Navrátil, P.A., Papka, M.E., Gaither, K.P.: Ray tracing and volume rendering large molecular data on multi-core and many-core architectures. In: Proceedings of the 8th International Workshop on Ultrascale Visualization. pp. 5:1–5:8. UltraVis '13, ACM, New York, NY, USA (2013), http://doi.acm.org/10.1145/2535571.2535594

9. Larsen, M., Meredith, J., Navrátil, P., Childs, H.: Ray-Tracing Within a Data Parallel Framework. In: Proceedings of the IEEE Pacific Visualization Symposium. pp. 279–286. Hangzhou, China (Apr 2015), 10.1109/PACIFICVIS.2015.7156388

10. Larsen, M., Brugger, E., Childs, H., Eliot, J., Griffin, K., Harrison, C.: Strawman: A batch in situ visualization and analysis infrastructure for multi-physics simulation codes. In: Proceedings of the First Workshop on In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization (ISAV), held in conjunction with SC15. pp. 30–35. Austin, TX (Nov 2015), http://doi.acm.org/10.1145/2828612.2828625

11. Levoy, M.: Display of surfaces from volume data. IEEE Comput. Graph. Appl. 8(3), 29–37 (May 1988), DOI: 10.1109/38.511

12. Moreland, K., Sewell, C., Usher, W., ta Lo, L., Meredith, J., Pugmire, D., Kress, J., Schroots, H., Ma, K.L., Childs, H., Larsen, M., Chen, C.M., Maynard, R., Geveci, B.: Vtk-m: Accelerating the visualization toolkit for massively threaded architectures. IEEE Computer Graphics and Applications 36(3), 48–58 (2016), DOI: 10.1109/MCG.2016.48

13. Peterka, T., Yu, H., Ross, R., Ma, K.L.: Parallel volume rendering on the ibm blue gene/p. In: Proceedings of the 8th Eurographics Conference on Parallel Graphics and Visualization. pp. 73–80. EGPGV '08, Eurographics Association, Aire-la-Ville, Switzerland, Switzerland (2008), DOI: 10.2312/EGPGV/EGPGV08/073-080

14. Peterka, T., Yu, H., Ross, R., Ma, K.L., Latham, R.: End-to-end study of parallel volume rendering on the ibm blue gene/p. In: Proceedings of ICPP 09. Vienna, Austria (2009), DOI: 10.1109/ICPP.2009.27

15. Sakamoto, N., Kawamura, T., Koyamada, K., Nozaki, K.: Improvement of particle-based volume rendering for visualizing irregular volume data sets. Computers & Graphics 34(1), 34–42 (2010), DOI: 10.1016/j.cag.2009.12.001

16. Sakamoto, N., Koyamada, K.: Kvs: A simple and effective framework for scientific visualization. Journal of Advanced Simulation in Science and Engineering 2(1), 76–95 (2015), http://doi.org/10.15748/jasse.2.76

17. Sakamoto, N., Nonaka, J., Koyamada, K., Tanaka, S.: Particle-based volume rendering. Asia-Pacific Symposium on Visualization 2007 pp. 129–132 (2007), DOI: 10.3154/tvsj.27.7

18. Tu, T., Yu, H., Bielak, J., Ghattas, O., López, J.C., Ma, K., O'Hallaron, D.R., Ramírez-Guzmán, L., Stone, N., Taborda-Rios, R., Urbanic, J.: Analytics challenge - remote runtime steering of integrated terascale simulation and visualization. In: Proceedings of the ACM/IEEE SC2006 Conference on High Performance Networking and Computing, November 11-17, 2006, Tampa, FL, USA. p. 297 (2006), DOI: 10.1145/1188455.1188767

19. Tu, T., Yu, H., Ramirez-Guzman, L., Bielak, J., Ghattas, O., Ma, K.L., O'Hallaron, D.R.: From mesh generation to scientific visualization: An end-to-end approach to parallel supercomputing. In: Proceedings of the 2006 ACM/IEEE Conference on Supercomputing. SC '06, ACM, New York, NY, USA (2006), DOI: 10.1145/1188455.1188551

20. Wald, I., Johnson, G.P., Amstutz, J., Brownlee, C., Knoll, A., Jeffers, J., Gunther, J., Navrátil, P.A.: Ospray - A CPU ray tracing framework for scientific visualization. IEEE Trans. Vis. Comput. Graph. 23(1), 931–940 (2017), DOI: 10.1109/TVCG.2016.2599041

21. Woop, S., Feng, L., Wald, I., Benthin, C.: Embree ray tracing kernels for cpus and the xeon phi architecture. In: ACM SIGGRAPH 2013 Talks. pp. 44:1–44:1. SIGGRAPH '13, ACM, New York, NY, USA (2013), DOI: 10.1145/2504459.2504515

22. Yamashita, S., Yoshida, H., Takase, K.: Development of numerical simulation method for relocation behavior of molten debris in nuclear reactors (1) preliminary analysis of relocation of molten debris to lower plenum. In: Proceedings of 21st International Conference on Nuclear Engineering (ICONE-21). vol. 4, p. V004T09A109. Nuclear Engineering Division (2013), DOI: 10.1115/icone21-16604