Chapter 14

Advanced Visualization Research Team

14.1 Members

Kenji Ono (Team Leader)

Jorji Nonaka (Researcher)

Mikio Iizuka (Researcher)

Chongke Bi (Postdoctoral Researcher)

Daisuke Sakurai (Postdoctoral Researcher)

Kazunori Mikami (Technical Staff)

Tomohiro Kawanabe (Technical Staff)

Seiji Fujino (Senior Visiting Scientist)

Naohisa Sakamoto (Visiting Scientist)

Masahiro Fujita (Visiting Technician)

Kentaro Oku (Visiting Technician)

Steve Petruzza (Student Trainee)

Yukiko Hayakawa (Assistant)

Keiko Matsuoka (Assistant)

14.2 Research Activities

Although the research activities of this team cover a broad range of the end-to-end simulation pipeline, including simulation, visualization and analysis, the main activities are centered on the study, design, and development of effective tools and mechanisms for visualization and analysis of large-scale parallel simulation results generated by the K computer. In this fiscal year, the team has worked on some core technologies necessary for building a production-level visualization and analysis framework. Taking into consideration the heterogeneous hardware infrastructure for post-processing, which includes the K computer itself, some post-processing servers, a visualization oriented GPU cluster, and local computer devices, we have worked on a framework named **HIVE (Heterogeneously Integrated Visual analytics Environment)** [21]. Some of the core technologies, for enabling large data visualization and analysis, include a scalable parallel image compositing library for massively parallel rendering environments **(234Compositor)** [3, 10, 9], a visual data analysis mechanism for multivariate data **(Fiber Surface)**, a performance monitoring library for scientific applications **(PMlib)** [24], a multi-display management library for building scalable cooperative workspace on tiled wall displays **(ChOWDER)** [16], a parallel-in-time integration framework **(PinT)** based on *Parareal* algorithm, and a sparse modeling approach based data compression for *in-situ* visualization **(Sparse Modeling)**.

Effective visualization and analysis of large-scale data sets generated by modern leading-edge supercomputers such as the K computer is widely recognized as a difficult and challenging problem. There is still a lot of discussion among researchers about this topic, and we can cite the Dagstuhl-style *Shonan Meeting Seminar* focusing on *"Big Data Visual Analytics"* [12], and also a panel discussion entitled *"Top Computational Visualization R&D Problems 2015"* [6] during the *Symposium on Visualization in High Performance Computing*. Our research activities have focused on a sustainable long-term development lifecycle for the HIVE framework, trying to assure the easiness for enhancement, maintenance and support, and also mechanisms for absorbing the heterogeneity of the computer platforms normally found on modern HPC operational environments. In addition to these items, we have also considered the following topics: easy usability; customizability for enabling diverse visualization scenarios; interactivity for finding the best visualization parameters; and in-situ processing mechanisms.

Most of the results of aforementioned research and developemnt efforts have already been released as open source libraries and applications, which are continuously maintained and updated. Following are the full list of those libraries and applications: *PDMlib* [23]; *UDMlib* [27]; *HDMlib* [20]; *CDMlib* [15]; *Polylib* [25]; *Cutlib* [17]; *LPTlib* [22]; *TextParser* [26]; and *JHPCN-DF* [18].

14.3 Research Results and Achievements

14.3.1 Visual Analytics Framework (HIVE)

We integrated some of the core technologies listed in the previous section, and developed a prototype of the HIVE framework focusing on the operational environment of the K computer. Considering the hardware heterogeneity of this environment, which includes the K computer itself, some post-processing servers, visualization oriented GPU cluster, and local computational devices, HIVE was designed to run on *SPARC64fx CPUs*, *x86 CPUs*, and *GPUs*. Figure 14.1 shows the Web-based User Interface(UI) designed for accessing the HIVE functionalities, and for small data sets it is possible to execute the entire visualization and analysis on the local computational side. For larger data sizes, this Web-based UI can be useful for preparing the visualization scenes to be used as a parameter during the batch job execution of large-scale parallel visualization processing. As shown in the Figure 14.1, HIVE adopted the sort-last parallel rendering approach, which requires the image compositing **process** right after the parallel rendering stage. It is worth noting that the scalability target of both parallel processing modules were the full node of the K computer.

In order to enable the sustainable long-term development and maintenance life cycle of the HIVE framework, we focused on a minimum set of external libraries for avoiding the software dependency as much as possible. For instance, the **SURFACE** rendering module adopted the *GLES 2.0* (OpenGL

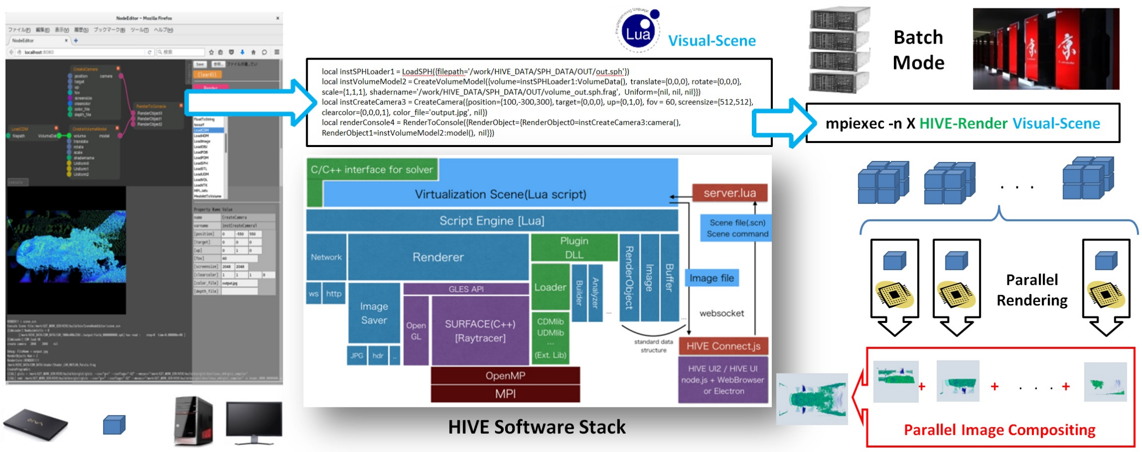

Figure 14.1: The software stack and the Web-based user interface of the HIVE framework designed for the visualization and analysis of large-scale simulation results generated from HPC platforms.

for Embedded Systems Version 2.0) compatible *API* (Application Programming Interface). The *GLES* is a subset of the original cross-language and multi-platform *OpenGL API* for rendering 2D and 3D vector graphics. This compatibility facilitates the development of both CPU and GPU based implementations. HIVE has designed to have a minimum but necessary set of functionalities, and the **Plugin DLL** module facilitates the extensibility and maintainability of the HIVE system by separating the core modules and other external or third party modules. This feature is possible, in part, due to the adoption of the light-weight and multi-platform scripting language called **Lua**. This scripting language is also used for describing the entire dataflow of a given visualization scene (**Visual-Scene** in the Figure 14.1), which can be easily converted to a batch job script for executing directly on the K computer. Examples of the third-party modules already incorporated to the HIVE include the large-scale parallel image compositing module **(234Compositor)**, and a set of large-scale data management libraries **(xDMlib)**. Other modules which are in the process of integration include those for visual analysis: **(Fiber Surface)** and Performance Monitoring **(PMlib)**.

14.3.2 234Compositor

The core rendering engine of the HIVE is a highly scalable *Raytracer* named **SURFACE** (Scalable and Ubiquitous Rendering Framework for Advanced Computing Environments), which adopted the *Sort-last parallel rendering* approach, that is, after the parallel rendering stage the entire set of rendered images should be gathered and merged into the final single image as shown in the Figure 14.1. To meet the performance and scalability demands of the SURFACE, we have developed the *234Compositor*, a hybrid MPI/OpenMP parallel image compositor library, based on the well-known *Binary-Swap* parallel image compositing algorithm. Although Binary-Swap has proven highly scalable, there exist some issues including the requirements for power-of-two number of processes, and the final sub-image collecting cost by using the *MPI_Gatherv* collective operation.

There already exist some extensions for the Binary-Swap to enable the use with non-power-of-two number of processes. One of these extensions is *Telescoping* method where the number of processes is gradually converted to the largest power-of-two number of processes, and was implemented on the *Ice-T* parallel image compositing library adopted by some parallel visualization applications such as *ParaView* and *VisIt*. Although it has proven efficient on specific supercomputer hardware architectures, we focused on a single-stage conversion approach to minimize the conversion overhead, and utilized the *3-2* and *2-1 Eliminations* techniques proposed by *Rabenseifner et al.* for MPI communications involving non-power-of-two odd number of processes. We extended the original algorithm for enabling the use to the even number of processes by applying the *234 Scheduling* [9], as shown in Figure 14.2. The communication pattern of the *3-2 Elimination* was also extended to enable the overlapping of communication and computation.

The final stage of the Binary-Swap image compositing algorithm is the gathering of image fragments, distributes among the image compositing nodes, to the master node and the assembly of the final image. In order to minimize the performance degradation when using a large number of node counts, we developed the *Multi-Step Image Composition* approach[3], which works by recursively

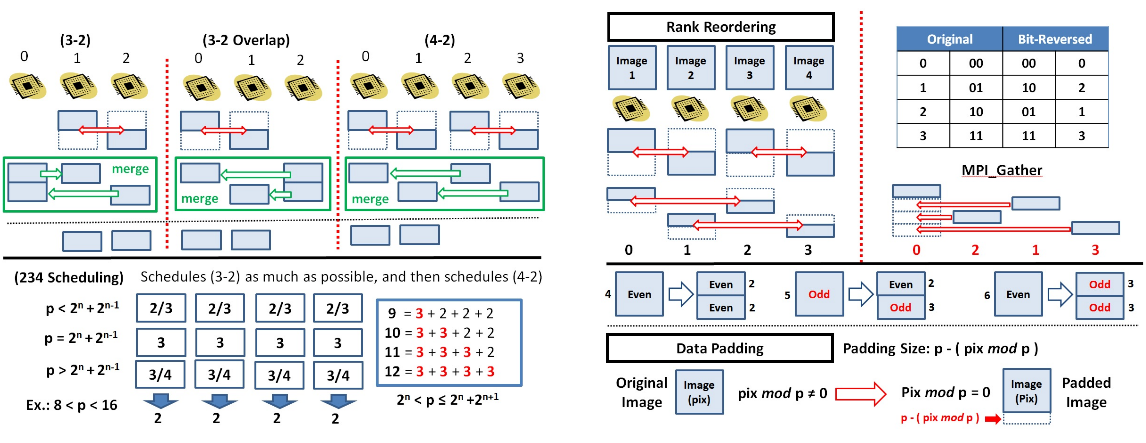

Figure 14.2: *3-2*, *3-2 Overlap* and *4-2 Eliminations* with the *234 Scheduling* mechanism (Left side). *MPI Rank Reordering* and *Data Padding* for enabling the use of *MPI Gather* (Right side)

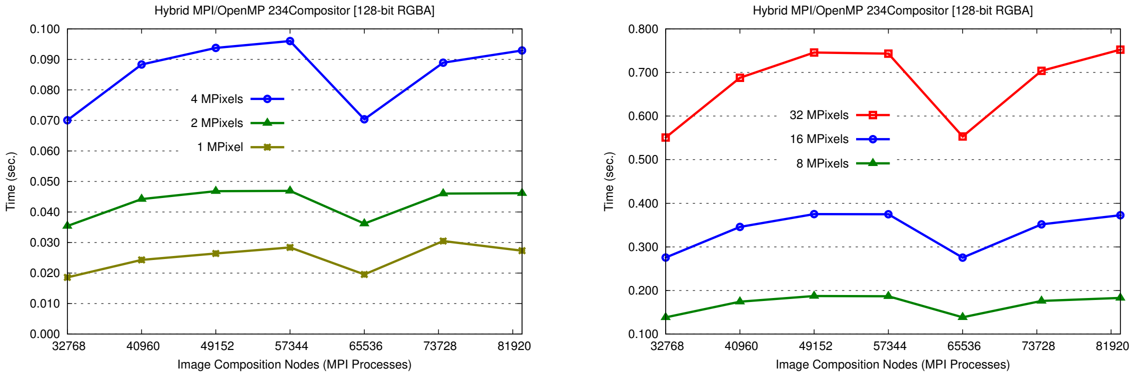

Figure 14.3: Scalability evaluation of the *234Compositor* for image sizes from 1 to 32 Mega Pixels (RGBA 128-bit) using up to the full node of the K computer.

dividing the composition nodes into smaller groups with a pre-defined threshold. We also worked on *MPI rank reordering* and *image data padding* techniques for enabling the use of *MPI_Gather*, in substitution to the traditional *MPI_Gatherv*, for optimizing the final image gathering stage[10]. Figure 14.3 shows the performance evaluation results of the *234Compositor*, which integrates the aforementioned techniques, and we could confirm the scalability up to the full node of the K computer when compositing 128-bit RGBA images with sizes varying from 1 to 32 MP (Mega Pixels).

14.3.3 Fiber Surface

With our *fiber surface GUI* (Fig. 14.4), a user can analyze multivariate data to quickly extract 3D features such as vortex structures in meteorology data and objects in CT scans. When the user is familiar with the popular isosurface analysis, it is not difficult to understand the concept of fiber surface – it is a straight-forward generalization of isosurface. The GUI is integrated into HIVE so that the user can analyze large-scale datasets in various computational architectures. In fact, multivariate datasets are often the output of simulations in supercomputers like the K computer. The user explores data by querying data values of interest. The query is as simple as drawing a polygon in a scatterplot. Our GUI then visualizes the queried samples in the 3D coordinate as 2D polyhedra. Thanks to fiber surface, the data size becomes magnitudes smaller compared to the original 3D data. This means that once the fiber surface is extracted, even computers with weaker performance can visualize the simulation outputs efficiently. The GUI resembles to traditional isosurfacing tools, while respecting today's GUI design for multifield exploration. The GUI also offers a scripting environment in order to support flexible data analysis.

14.3.4 Development of PMlib (Performance Monitoring Library)

PMlib is an open source software library designed for monitoring the computational performance of scientific applications. PMlib has designed to support hybrid parallelism where the parallel distributed memory and shared memory computing are exploited by the applications through specific libraries such as MPI and OpenMP. This library is suitable for monitoring computational workloads of applications which can change dynamically over the time and space, depending on their initial conditions and state variables, such as computing particle trajectories over the subdomains in parallel fluid simulations.

The performance statistics for the computational workload such as floating point operations per second (flops) and memory load/store operations per second (bandwidth) can either be defined as the algorithmic workload, that is, theoretical requirements decided by the source program, or as the actually executed system workload. The latter varies depending on the system conditions including the compiler optimization, prefetch strategy and data movement on hierarchical cache components, and numbers of pipeline stages for arithmetic operations.

This library allows the users to select the best policies to obtain the statistics during runtime executions. The statistics policies can be explicitly declared by the users as API arguments inside the source program, or can be automatically acquired from the processor built-in hardware counters. PMlib will produce the report in text format at a post processing phase as shown in Figure 14.5.

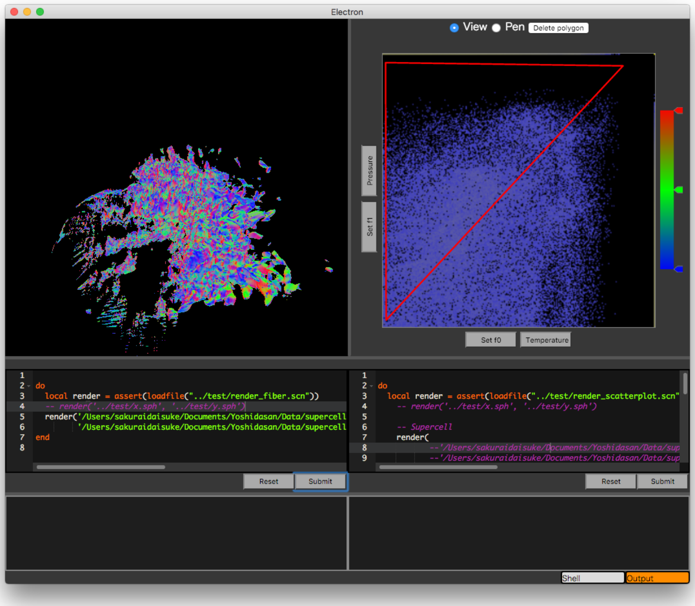

Figure 14.4: Fiber surface GUI visualizing water cohesion in the atmosphere (simulated with SCALE, which is developed by the Computational Climate Science Research Team of RIKEN AICS).

```
Parallel Mode:   Hybrid (8 processes x 2 threads)
The environment variable HWPC_CHOOSER=FLOPS is provided.
Total execution time            = 2.270602e+00 [sec]
Exclusive sections statistics per process and total job.
Inclusive sections are marked with (*)

Section           |call|        accumulated time[sec]           | [hardware counter flop counts]
Label             |    |     avr    avr[%]      sdv    avr/call |      avr        sdv    speed
------------------+----+----------------------------------------+----------------------------
First section     :    1  9.086e-01  30.29  5.52e-01  9.086e-01   5.909e+09  7.25e+08   6.50 Gflops
Second section(*) :    1  1.996e+00  66.54  4.18e-01  1.996e+00   5.752e+09  3.94e+08   2.88 Gflops(*)
Subsection X      :    1  7.349e-01  24.50  1.89e-01  7.349e-01   6.586e+09  5.32e+08   8.96 Gflops
Subsection Y      :    1  6.123e-01  20.41  1.26e-01  6.123e-01   5.896e+09  3.49e+08   9.63 Gflops
------------------+----+----------------------------------------+----------------------------
Sections per process      2.256e+00    -Exclusive CALC sections-  1.839e+10             8.15 Gflops
------------------+----+----------------------------------------+----------------------------
Sections total job        2.256e+00    -Exclusive CALC sections-  1.471e+11            65.22 Gflops
```

Figure 14.5: Output example of PMlib.

In this fiscal year, the development efforts for PMlib was focused on the following items:

- Fortran API support
- Multiple MPI group support
- Addition of example programs and update of the user document
- Addition of an optional sorted output
- Initial study of a graphical interface for the Open Trace Format

14.3.5 ChOWDER

ChOWDER (*Cooperative Workspace Driver*) is a Web browser based multi-monitor controller system designed to assist collaborative work among multiple users and applications. The development of *ChOWDER* has been conducted by the financial support of a cross-ministerial Strategic Innovation promotion Program (SIP) grant. *ChOWDER* delivers the functionality of virtual display driver for tiled wall display, and can be used for building scalable cooperative workspace environment for designers and engineers.

As shown in Figure 14.7, *ChOWDER* system is composed of the following three subsystems:

- Client Controller
- Client Display
- Display Server

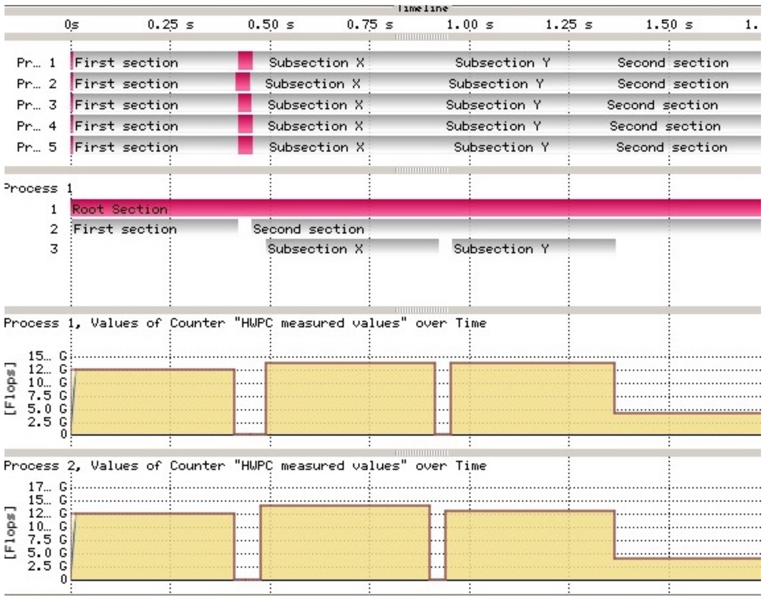

Figure 14.6: Statistical view of PMlib's output.

Client Controller is used for positioning the client displays on the user defined virtual display space. The controller is also used for positioning the displaying objects inside the virtual display space, via Web-based User Interface (UI) since *Client Controller* works inside the Web browser.

Client Displays are the physical displays where the contents in the virtual display space are actually displayed. Figure 14.7 shows an example of 8 *Client Displays* forming a single virtual display, and the user can make a huge virtual display space by simply adding new *Client Displays* to these existing *Client Displays*. These subsystems can work on a variety of devices based on the following Operational Systems: Windows, Mac, Linux, iOS, and Android. *Display Server* controls the communications between *Client Controller* and *Client Displays*, and manages the display objects to be displayed on *Client Displays*. *Display Server* was implemented by using the *node.js*, *socket.io*, *websockets*, and *redis* technologies. As shown in the Figure 14.7, *Display Server* possesses an API for cooperative behavior, which enables a seamless communication with HIVE system, that is, the graphics output of the HIVE can be sent to *ChOWDER's Client Displays*.

14.3.6 Parallel-in-Time Integration

Parallel-in-time integration (PinT) is desired for effectively exploiting the concurrency of the post peta-scale parallel computing environments which are expected to increase the currently achievable degree of concurrency. The study of PinT advances rapidly after *the Parareal method*(Fig. 14.8(a)) was proposed by Lions at the beginning of the 2000's. The Parareal method has the characteristic that it is easy to reuse existing codes and it doesn't demand large memory capacity. In other words, Parareal method provides us a general and practical framework to describe parallel-in-time integration algorithms for a variety of simulation codes. Therefore, we have investigated a general-purpose PinT framework, which employs the *Parareal* method, to reduce the developer efforts for describing the PinT codes (Fig. 14.8(b)).

We have investigated and found that *Parareal* method works well for parabolic PDEs, but does not work so well for hyperbolic PDEs. Therefore, we performed some investigations on the trends and prospects of PinT methods, and then we set a research direction for the PinT study. As the next steps, we will intensively study the Reduced basis methods (Fig. 14.9(a)) which use the sparse modeling technique, phase alignment methods (Fig. 14.9(b)), adjoint-based methods which use symmetrization of the time-integrator and Parareal/SDC hybrid methods.

14.3.7 Sparse Modeling

Large-scale simulation usually needs several days, or even several months to be executed on current supercomputers. It is strongly desired by researchers to observe the visualization results of such kind

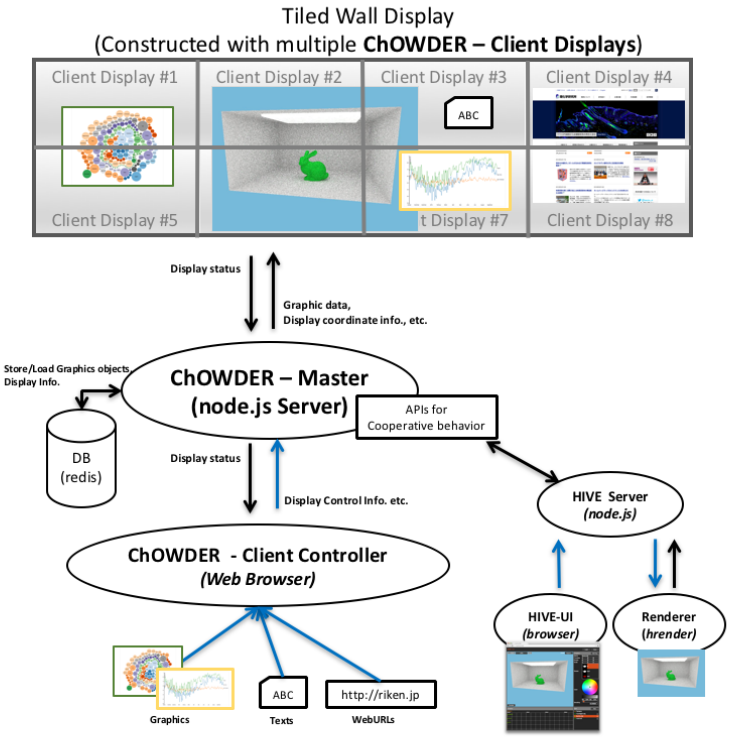

Figure 14.7: System diagram of ChOWDER.

of simulations in real time for analysis and adjustment of initial simulation parameters. However, due to the limitation of memory size, currently most of such kinds of simulation datasets are analysed as post processing such as visualization. It is usually necessary to execute these large-scale simulations several times for selecting a set of good initial parameters for simulations and visualizations, and this leads to a time consuming process for scientists and engineers. In this research, we investigated a sparse modeling-based interactive *in-situ* visualization method, in order to assist the users to visualize the simulation results along with the running simulation. By analysing these visualization results in real-time, the users will be able to adjust the simulation parameters and restart the simulation without the need for waiting to the conclusion of the entire simulation. As a result, it becomes possible to avoid the repetitive execution of the entire simulation just for the parameter selection process. The Proper Orthogonal Decomposition (POD) algorithm is used to design a sparse model for getting a good compression ratio for assisting the *in-situ* visualization. We also investigated a data sharing approach between the simulation code and the compression code to obtain the best performance. The visualization framework shown in Figure 14.1 is employed for the rendering and interactive visualization. HIVE is a hardware independent parallel visualization system with a scalable rendering capability.

14.4 Schedule and Future Plan

The next big step for the *HIVE* framework is its transition from the prototype stage to a solid and robust production-level system. During the early stage of the development, we have already held an initial presentation through the AICS Open Source Workshop Program to present an overview, and at the same time, to obtain a feedback from the potential users. In the next fiscal year, we are planning to take more advantage of this kind of AICS internal events, as well as other external events, not only for disseminating *HIVE* system, but also to verify and discuss potential future enhancements based on the real-world requirements and needs. Focusing on the AICS internal demands, we are collaborating with some AICS teams involved in the computational science fields for utilizing their real simulation results. In this initial stage, we are using some datasets from the Computational Climate and Computational Fluid Dynamics(CFD) simulations, and have investigated some directions for further enhancements. We are planning to gradually enlarge this group of collaboration teams for aggregating new functionalities based on real world demands. The loosely coupled modular

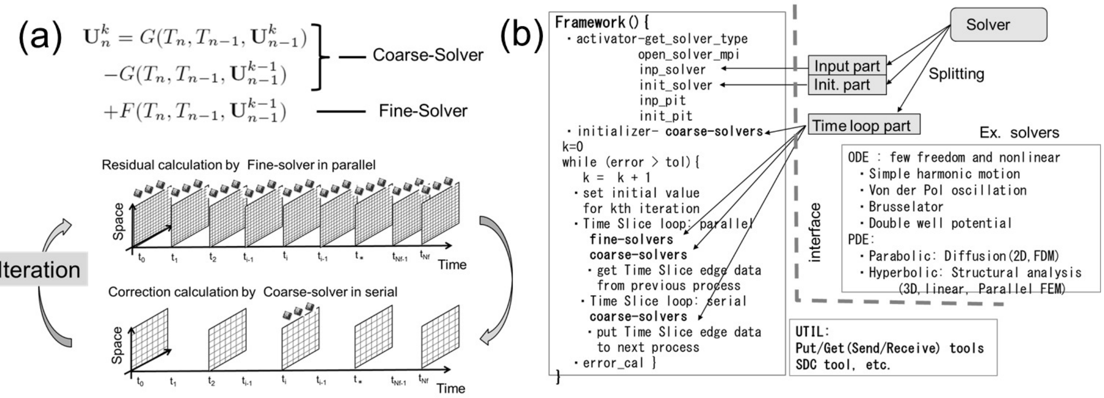

Figure 14.8: *PinT* framework: (a) *Parareal* algorithm, (b) Prototype of a *PinT* framework.

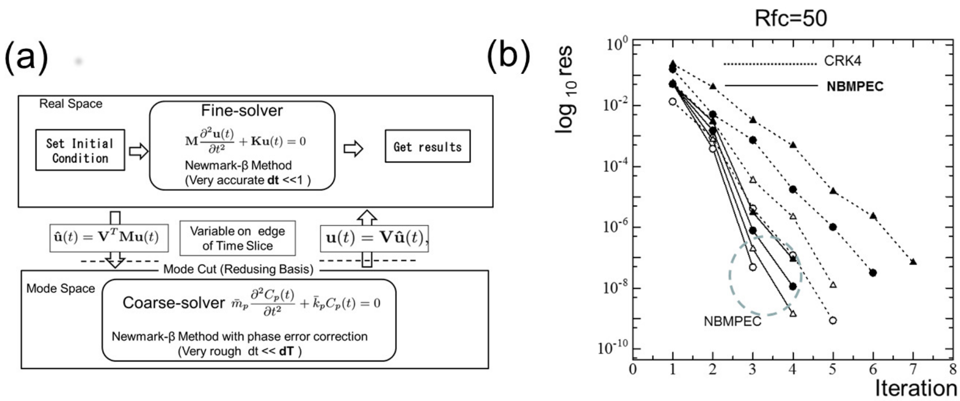

Figure 14.9: *PinT* methods: (a) *Parareal* process based on Reduced basis methods, (b) Convergence of *Parareal* method using *Newmark-β* method with phase error correction(NBMPEC) vs CRK4(Classical 4th-order Runge-Kutta method). Number of cycle = 100 (○), 1,000 (△), 10,000 (●), 100,000 (▲).

approach adopted by the HIVE system greatly facilitates the aggregation of new functionalities as native or third-party modules. Our initial priorities are: **Fiber Surface**, **FFV/C**[19], **PBVR**, and **PIDX**. In the next fiscal year, we are planning to aggregate the large-scale **PBVR** (Particle-Based Volume Rendering) functionality being developed under the Grant-in-Aid for Scientific Research (KAKENHI) in collaboration with Kyoto University and Kobe University. **PIDX** is a data format for streaming based visualization developed by the University of Utah, which our team has a signed MOU (Memorandum of Understanding). In the next fiscal year, we will accept a graduate student from this university as a student trainee, in order to investigate the potential of this approach on the K computer environment and a future aggregation to *HIVE* system.

14.5 Publications

Journal Articles

[1] Kei Akasaka et al. "Rapid Simulation Method for Airflow around Production Vehicle Using Immersed Boundary Method and Local Mesh Refinement". In: *Journal of Automotive Technology* 46.5 (2015), pp. 963–968.

[2] Koki Isobe et al. "Effectiveness of Two-dimensional Medical Image Interpolation in the Transverse Plane in the Voxel Analysis of Nasal Air Flow and Temperature". In: *Medical and Biological Engineering* 53.3 (2015), pp. 160–167.

[3] Jorji Nonaka, Masahiro Fujita, and Kenji Ono. “Multi-Step Image Composition Approach for Sort-Last Massively Parallel Rendering”. In: *Journal of Advanced Simulation in Science and Engineering* 2.1 (2015), pp. 108–125.

[4] Kenji Ono et al. “Low Byte/Flop Implementation of Iterative Solver for Sparse Matrices Derived from Stencil Computations”. In: *High Performance Computing for Computational Science – VECPAR 2014*. Vol. 8969. Lecture Notes in Computer Science. 2015, pp. 192–205.

[5] Huajian Xue et al. “Modeling Uyghur Speech Phenomena with Morphological Rules”. In: *Recent Patents on Computer Science* (2015), pp. 58–66.

Conference Papers

[6] Issei Fujishiro et al. “Top Computational Visualization R&D Problems 2015: Panel”. In: *SIGGRAPH Asia 2015 Visualization in High Performance Computing*. SA ’15. 2015, 20:1–20:13.

[7] Seigo Imamura, Kenji Ono, and Mitsuo Yokokawa. “Performance Evaluation of Iterative Solver for Multiple Vectors Associated with a Large-Scale Sparse Matrix”. In: *27th International Conference on Parallel Computational Fluid Dynamics Conference Abstract*. Montreal, Quebec, Canada, May 2015, pp. 124–125.

[8] Peng Li et al. “Person Re-Identification Using Color Enhancing Feature”. In: *The 3rd IAPR Asian Conference on Pattern Recognition*. 2015.

[9] Jorji Nonaka, Masahiro Fujita, and Kenji Ono. “234 Scheduling of 3-2 and 2-1 Eliminations for Parallel Image Compositing Using Non-Power-of-Two Number of Processes”. In: *The 2015 International Conference on High Performance Computing & Simulation (HPCS 2015)*. Amsterdam, Netherlands, 2015, pp. 421–428.

[10] Jorji Nonaka, Masahiro Fujita, and Kenji Ono. “Rank Reordering and Data Padding for Optimizing Large-Scale Parallel Image Composition”. In: *IPSJ High Performance Computing Symposium (IPSJ HPCS2015)*. Tokyo, 2015, pp. 64–72.

[11] Jyunya Onishi, Kenji Ono, and Soichiro Suzuki. “Parallelization of a Block-Based Hierarchical Cartesian CFD Code”. In: *Proceedings of the ASME-JSME-KSME Joint Fluids Engineering Conference 2015 AJK2015-FED*. Seoul, Korea, July 2015, AJK2015–29581.

[12] Kenji Ono et al. “High-Performance Extreme-Scale Visual Analytics”. In: *Part of NII Shonan Meeting Report No. 2015-7 (Big Data Visual Analytics)*. 2015, pp. 5–7.

[13] Ken Uzawa and Kenji Ono. “Validation of Local SGS Models Implemented in High-Performance CFD Solver: FFV-C”. In: *Proceedings of the ASME-JSME-KSME Joint Fluids Engineering Conference 2015 AJK2015-FED*. Seoul, Korea, July 2015, AJK2015–29634.

[14] Yucong Chris Ye et al. “In Situ Depth Maps based Feature Extraction and Tracking”. In: *5th IEEE Symposium on Large Data Analysis and Visualization, LDAV 2015, Chicago, IL, USA, October 25-26, 2015*. 2015, pp. 1–8.

Patents and Deliverables

[15] *CDMlib - Cartesian Data Management Library*. `http://avr-aics-riken.github.io/CDMlib`.

[16] *ChOWDER: Cooperative Work Driver*. https://github.com/SIPupstreamDesign/ChOWDER.

[17] *Cutlib - Cut Information Library*. `https://github.com/avr-aics-riken/Cutlib`.

[18] *Data compression library based on Jointed Hierarchical Precision Compression Number - Data Format*. `http://avr-aics-riken.github.io/JHPCN-DF`.

[19] *FFVC - FrontFlow Violet Cartesian*. `http://avr-aics-riken.github.io/ffvc_package`.

[20] *HDMlib - Hierarchical Data Management Library*. `https://github.com/avr-aics-riken/HDMlib`.

[21] *HIVE (Heterogeneously Integrated Visual analytics Environment): A Visualization Framework for Large-Scale Datasets*. `http://avr-aics-riken.github.io/HIVE`.

[22] *LPTlib - Lagrangian Particle Tracking Library*. `http://avr-aics-riken.github.io/LPTlib`.

[23] *PDMlib - Particle Data Management Library*. `http://avr-aics-riken.github.io/PDMlib`.

[24] *PMlib: Performance Monitoring Library*. `http://avr-aics-riken.github.io/PMlib`.

[25] *Polylib - Polygon Management Library*. `http://avr-aics-riken.github.io/Polylib`.

[26] *Text Parser Library*. `http://avr-aics-riken.github.io/TextParser`.

[27] *UDMlib - Unstructured Data Management Library*. `http://avr-aics-riken.github.io/UDMlib`.

[28] *V-Isio - VCAD Visualizer*. `http://avr-aics-riken.github.io/V-Isio`.