Analyzing an Ideas Used in Modern HPC Computation Steering

Pavel Vasev
Computer visualization lab
N.N. Krasovskii Institute of the Russian Academy of Sciences
Ekaterinburg, Russia
vasev@imm.uran.ru

***Abstract*—Computation steering is a scientific and technical area which provides methods of understanding a state of running high-performance computing (HPC) programs and performing interactive control over such programs. This work is an analysis of ideas and current status of computation steering. The main idea of computation steering is that computation and visualization are performed simultaneously. By the way, modern HPC systems approached the disbalance: they have much more computing power than storage power. Thus computations cannot output all meaningful data easily. To solve this, steering technologies are used – with idea to run in non-interactive mode, extracting program's in-memory data and saving it for ordinal post-hoc analysis. This lead to new demands on algorithms of correct data reduction, in purpose of correctness of various following data analysis. Another idea of modern steering is a configuration of parts of visualization pipeline in text files separate from simulation codes. This allows to relatively easy setup data transformations and connections between them and their placement on various locations, including computational nodes. Modern steering systems rely on model: describe simulation data's life using API and then configure in text files what to do with that data in purpose of visualization during simulation.**

Keywords—computation steering, on-line visualization, HPC visualization, in situ visualization, scientific visualization

I. Introduction

In high-performance computing (HPC), steering is a process of data extraction from running HPC program and also a process of controlling that program by modifications of it's state. An example of steering is presented on figure 1.

In short, with computation steering a researcher achieves the following possibilities:

1. Gain extra knowledge of model behavior by examining intermediate simulation state.

2. Prematurely stop simulation that fell into an uninteresting trajectory thus saving resources.

3. Perform faster "what if" simulations by changing running computation parameters.

The history of computation steering is as old as computations itself, but it's approaches changes along with HPC technologies evolution. This paper introspects modern HPC steering technologies and outlines ideas used in them.

II. Terms overview. Phenomena of "In situ" approach

There are a plenty of terms used.

First of all, "computation steering" sometimes denoted as "computational steering". Then, there are terms "online visualization", "interactive steering" and "dynamic steering". I suppose that all terms mentioned above are used to identify the same area – while naming difference underlines the most valuable aspect for particular point of view.

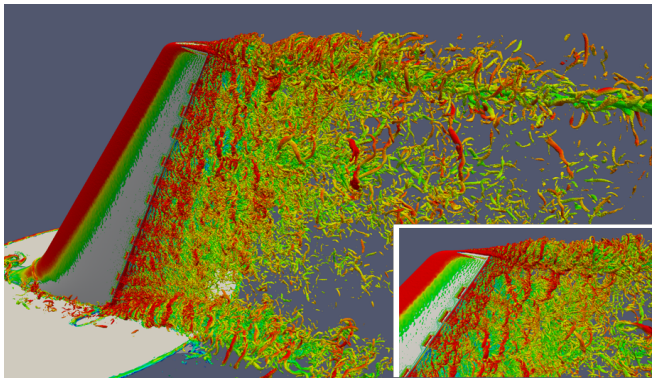

Fig. 1. The computation steering example: visualization during simulation. The simulation is performing on 256K computing cores [1].

In contrast, "in situ visualization" has a different sense.

For computation steering is does not matter where visualization stages (data processing in purpose of visualization, mapping and rendering) are performed. It is ready to deal with all combinations, looking on task of joining simulation and visualization as a whole.

However as stated in [1], [2] and fig. 2, modern HPC systems approached to the disbalance: they have much more computing capabilities than storage and I/O capabilities. In that disbalance, a researcher cannot save all computed simulation output data even in ordinal simulation pipeline with post-hoc processing. The problem increases if researcher is interested in intermediate simulation state.

This fact is complicated by another fact that visualization is sensitive to I/O bandwidth [3]. This means that even if simulation is able write out a data – visualization cannot read this data due to I/O bottleneck with storage.

To overcome stated problems, the "in situ visualization" [4] is used. It's idea is to put all or parts of visualization pipeline closer to computations. "Closer" almost always imply putting them into computation resources or even in simulation codes.

For example, data reduction might take place right in simulation, and reduced data is saved in ordinal way for following post-hoc analysis. Another example – data is copied out of computing memory onto dedicated nodes for processing and rendering images for online analysis.

Important aspect of "in situ" approach is given in [3]: "data reductions must be done carefully in order to preserve the integrity of the underlying data when post-hoc analysis will need to generate derived quantities from reduced data".

I suppose that in situ visualization is a computation steering where only some parts of visualization pipeline works simultaneously with simulation. Attention is paid to have pipeline parts fastest access to simulation data. In situ visualization deals with both variants of when analysis is performed: during or after simulation.

III. Steering systems

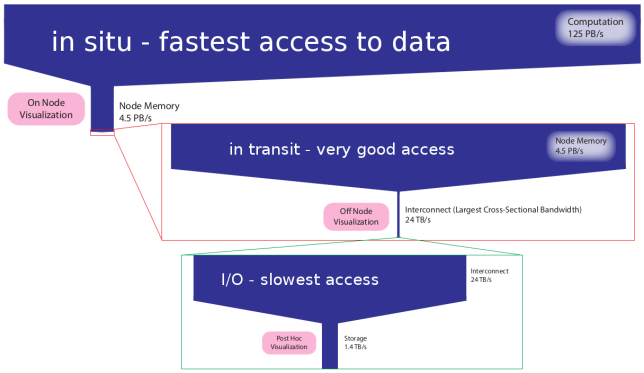

Fig. 2. A plot of the relative bandwidth of system components in a supercomputer. The widths of the blue boxes are proportional to the bandwidth of the associated component. [2].

Below we outline ideas that we extracted from some of modern steering systems. We provide idea list incrementally, with no repeats. Beside mentioned systems, other systems which we reviewed – Libsim and Ascent – contains in most similar ideas. Additional ideas may be found in ParaView Catalyst, ADIOS and PAVE and other systems.

A. The SENSEI framework

The Sensei [5] is a US-based open-source project available at sensei-insitu.org website. It suggest to instrument (extend) simulation code with 2 extra calls:

1. Describe computation data distribution in a memory of computing nodes (the data adapter in Sensei's terms)

2. Call synchronization function during computation iterations (the analysis adapter in Sensei's terms).

The step 1 provides information to Sensei about which data a program has and how is it distributed across it's processes memory. Such approach with description is widely used, as it will be seen in the following sections.

The step 2, however, is tricky. In Sensei's terms, this is analysis step, and it suggest to put any code that is desired for that purpose there. But in practice, the Sensei provides own universal analyze code which is configurable via external xml file. In that file a researcher may describe a such complex elements as:

a) Data reduction and transformation in-place (e.g. in simulation processes).

b) Transmission to outer space (from simulation) for other on-line processing.

c) Storage actions on extracted data and/or connection with on-line visualization tools, among with configuration for such tools.

d) Combinations of the above.

I suggest to look on steps 1-2 in metaphorical way of "embedding a representative" of visualization pipeline into simulation. In this point of view, these steps are parts of visualization pipeline which are physically placed into simulation code for purpose of overall coding simplicity. The stated metaphor in a wide sense was noted by Mikhail Bakhterev from N.N. Krasovskii Institute.

This representative may perform various tasks, including rendering on computing nodes. The Sensei, in particular, implemented a feature of interpreting user-defined external python scripts inside an analysis adapter.

Thanks to external textual configuration (e.g. xml file), the visualization and investigation tools might be easily reconfigured without any modification to simulation codes.

A user provides configuration, runs the simulation, and achieves his aim. When user's aim changes (for example, she decides to get new data extracts) – she modifies configuration file and achieves new results. Actually user reprograms a visualization pipeline when she changes that configuration file.

One additional interesting aspect to be noted is an implemented ability to "proxy" computation data out of computation processes memory. The Sensei has built-in analysis adapter type which transfers data to outside, and it's other process in outside re-publishes this data again (via special data adapter), and thus a data may be processed by another analyses codes in outside, like a chain.

This might be thought in turn as providing representative of computing program. Analysis codes connected with that representative even have no knowledge about where a real program is being running.

It is claimed that the Sensei successfully worked in configuration with 1 million of computing nodes [6]. By the way, the Sensei doesn't deal with transport, using for that purpose such frameworks as ADIOS, Ascent, Libsim or libIS, and playing a role of reconfigurable orchestrator.

B. LibIS

The libIS [7] library is connected with OSPRay rendering technology by Intel. Website: github.com/ospray/libIS.

The library uses almost same approach as Sensei: describe data and then provide access to that data using per-iteration calls to the library. When data is requested by clients, embedded per-iterations library calls perform asynchronous write-out operation.

LibIS provides following types for description of data and it's distribution across node's memory: 3d arrays (e.g. grid fields), 1D arrays, and arrays of structures (representing for example particle systems). This is not big amount in contrast to Sensei's, which uses vtkDataModel leading to more than dozen of data types. But it seems enough to be compatible with some simulation codes due to simplicity.

In contrast to Sensei's "in situ", libIS adopts "in transit" approach. In this approach, data for visualization purposes is processed outside of simulation processes. E.g. in per-iterations calls, the libIS provides access to described data only for possible transportation to outside. Forthcoming transformations of data are desired to be done somewhere in outer space by libIS "clients".

The networking with client is done in two stages: first, client sends connection request via ordinal IP port to computational process of rank 0, and then a communication is performed via newly-established dedicated MPI rank. There is no server or routing nodes in library: libIS assumes that client nodes resides in HPC network, probably on same type of computing nodes as simulation itself. LibIS assumes there are N client nodes accessing M simulation nodes and thus upon client's query each simulation node is instructed to send it's data to one of library-chosen client node in a point-to-point manner.

LibIS client's query language is simple. The query response consist of number of records (e.g. partial responses each from some computing node), where each record contains local coordinates (corresponding to that node), 0 or more field data records, and 0 or more particle array records.

LibIS may be used by Sensei as a transport layer. Moreover, libIS provides unpacking capabilities back to Sensei's data model. Thus simulation users may not only connect to Sensei-instrumented simulation using libIS, but connect other Sensei-compatible tools to libIS-instrumented simulations.

It is interesting that libIS uses following assumptions. 1) Each computation node owns one cubical part of common world, 2) Number of client nodes N is less or equal to simulation nodes M. These assumptions had simplified API of libIS and probably the library's code itself.

At the moment of writing this paper, it was known that libIS was successfully tested with simulation of 128 computing nodes and 2 client nodes [7].

C. ESPN

ESPN steering system [8] looks like libIS in it's internal spirit. ESPN is an abandoned project, but even in that state it outlines the following useful ideas:

1. A possibility to connect to simulation with different kind of clients simultaneously. For example, one client might perform parameters lookup and steering, and another client at the same time might visualize a data on tiled-display wall.

2. An access to computing processes should be somehow synchronized (to perform data access actions at logically same simulation iterations).

D. Damaris

The Damaris [9] is a product of INRIA, a French national research institution. Project website: project.inria.fr/damaris/.

The Damaris has very interesting approach on describing data layout inside simulation processes memory using external xml file. It also uses that xml file for describing Damaris system launch configuration (dedicated cores or dedicated nodes, so on). Moreover, Damaris asks user to describe important computation parameters (for example sizes of a grid) in that file, and these parameters may be used in other xml locations in arithmetic expressions, and also they are available in simulation code via read and write methods.

Damaris uses instrumentation cycle different from Sensei. Instead of "describing" data layout, a user has to "write" data of a given name with given pointer, and the name should correspond to one described in xml file (mentioned above). For some reason, during write operation Damaris copies data inside itself. It also provides special alloc and free procedures as alternative, but even in that case is is not so transparent for simulation codes as just the publishing of existing pointers (probably a shared memory approach used by Damaris is the reason for such behavior).

It is interesting that Damaris uses shared memory for inter-memory communications with simulation code. E.g. Damaris library code seems only publishes data to a shared memory, and transmit signals for accompanying in situ processes when data is ready to grab.

Damaris relies on assumption that simulation code is iterative, asking user to call *damaris_end_iteration* function at each cycle and also it internally counting those iterations. In Damaris, same as in libIS, each simulation process is assigned to one steering server process (on same node at some core or on dedicated node). Thus upon damaris_end_iteration (e.g. sync) call a corresponding server node is notified. It is tricky how Damaris assign server nodes: it chooses them among computation nodes and assumes that they will enter in forever loop (which is located in damaris_start code).

Damaris, same as Sensei, suggest a developers to write plugins. A plugin is supposed to provide response to some event, and the connection between event names and plugins is defined in xml file mentioned above. Moreover, developer might reside those plugin codes right in simulation processes. After a plugin code is called, it has access to Damaris internal API and may perform any actions.

Damaris is stated to work with 16K cores simulations [9].

IV. Remarkable efforts

In addition to ideas we found in steering systems, other ideas are located around them in various projects and other efforts. In this section we provide some interesting ones.

A. Real case: Data Reduction for Vector Field

In paper [3] scientists were interested on plasma field behavior between simulation checkpoints. Each iteration data was about 20TB and thus it was not subject to save it as is (at each iteration). It was decided to perform "in situ" data reduction: a data field was reduced to ~45x45 mesh which was in turn saved for each iteration, and due to it's size it was able to be visualized.

The reduction was made with big care, so scientist may still generate various derivatives from the reduced data and this derivatives were correct, e.g. same as derivatives from original data.

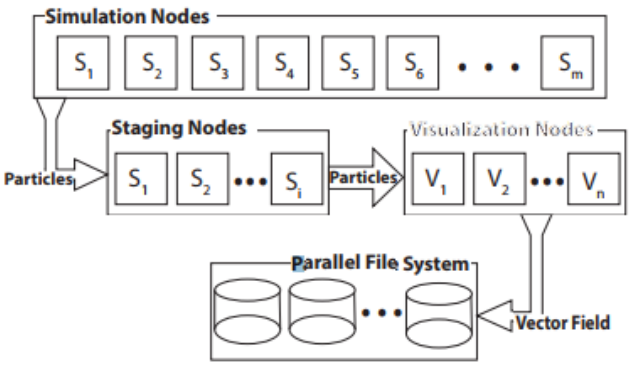

Fig. 3. Described fusion simulation and visualization data flow.

Simulation nodes were instrumented with ADIOS API routines. Preliminary work (e.g. particle sampling) is performed inside the simulation, and it's results are flushed out via ADIOS to special I/O-staging nodes. These staging nodes transit data to VTK-M parallel nodes, and latter in turn generate final reduced mesh. The solution was running on 1024 simulation, 12 staging and 32 visualization nodes (see fig. 3).

The referred work as a whole is a great example of how things are done. Interestingly, this work leverages only data extraction for post-hoc visualization purpose, abandoning online analysis and steering – which is probably enough.

B. NVidia Omniverse proposal for in situ

In its proposal, NVidia suggested to connect ParaView Catalyst steering system to Omniverse 3D scenes portal [10] using special software adapter. Thus a data from steered simulation may be uploaded via Catalyst to Omniverse's micro-service and updated during simulation runtime, and then visualized using various compatible rendering software kits – during simulation runtime or post-hoc. It uses Pixar Universal Scene Description format [11] for 3D scenes with

incremental patching feature, which is useful for runtime updates.

Current cave-at of suggested proposal is that data sent to Omniverse has to be already somehow reduced, because it's servers does not support data of HPC-scale sizes.

The proposal might be interesting if researcher will receive extra benefits from a data residing in Omniverse. Currently, for example, NVidia suggests to transparently transfer data to game engines: thus simulation might generate visualization on-line as an artifact for a bigger (gaming) world.

Other possible benefits might be in found if Omniverse will play a role of storage and transport to various tools. However even in that case it is unclear how it may compete with such thing as CinemaScience data format [12]. Probably it is better to create microservice-based hosting for CinemaScience databases, and even maybe incorporating the mentioned Pixar scenes inside those databases.

C. Comparing the efficiency of visualization placement

Two extreme approaches of in situ visualization, in-place and in-transit, were investigated in [13]. Some of conclusions of the work are:

1. Running visualization pipeline right in simulation processes in time slicing mode is superior when simulation iteration cycle is fast.
2. Sometimes it is more efficient to put visualization pipeline on dedicated nodes, for example reducing communications required by parallel rendering.

Remarkable that the work accounts all parts of visualization pipeline, including (parallel) rendering. The work provides very comprehensive analysis of various processes going on during in-place and in-transit schemes.

V. Conclusions

It is interesting that investigated computation steering systems neither rely on network protocol, nor have published conceptual protocols of their operation. I think this is a mismatching position in a long-term scale both for achieving new developers and for achieving new implementations for other infrastructures.

An exciting trend (or at least an approach) is to use external configuration files for computation steering operations. They are devoted for describing visualization pipeline: data voyage and transformation, visualization back-ends and their configuration, and even data layout in simulation (as in Damaris project). This fact means that whole computation steering process is programmed in those configuration files, including behavior and codes that are lately-bound into simulation processes.

It seems that big part in making simulation steerable is a description of various data types and memory layouts of simulation data in processes memory. Industry standard is required here, and it is still missing. Maybe vtkDataModel or Conduit [14] are a good staring point for that purpose.

It looks like all steering systems assume that data layout doesn't not change during simulation and is always present. But they seems do not restrict users to obey that – they are free to publish computation data at any time.

Also it seems that a lot of efforts will be spent onto finding methods of correct data simplification for extraction from computation process. An exciting examples of such efforts are [3] and [15].

Of course, deeper research of the subject should consider running mentioned software frameworks, comparing both their robustness and usability. This is subject to future work.

Acknowledgment

Pavel Vasev thanks Vladimir Averbukh [16], the head of Computer visualization lab at N.N. Krasovskii Institute, for unparalleled visionary and inspiring leadership during years.

Also Pavel Vasev thanks all the authors of referenced literature for their time, efforts, and detailed publications. Especially he thanks Kenneth Moreland for his brilliant ability to express ideas in a short and easy language.

References

[1] A. C. Bauer, H. Abbasi, J. Ahrens, H. Childs, B. Geveci, S. Klasky, K. Moreland, P. O'Leary, V. Vishwanath, B. Whitlock, and E. W. Bethel, "In Situ Methods, Infrastructures, and Applications on High Performance Computing Platforms", in *Computer Graphics Forum, 2016*, vol. 35, no. 3, pp. 577-597.

[2] Moreland K., "The tensions of in situ visualization", in *IEEE Computer Graphics & Applications*, March/April 2016, pp. 5–9, doi: 10.1109/MCG.2016.35

[3] Kress J., Choi J., Klasky S., Churchill M., Childs H., Pugmire D., "Binning Based Data Reduction for Vector Field Data of a Particle-In-Cell Fusion Simulation", in *High Performance Computing, ISC High Performance*, Yokota R., Weiland M., Shalf J., Alam S., Eds. 2018. Lecture Notes in Computer Science, vol. 11203. Springer, Cham, doi: 10.1007/978-3-030-02465-9_15

[4] Childs, H., "The in situ terminology project", [Online]. Available: https://ix.cs.uoregon.edu/~hank/insituterminology/

[5] "In Situ Analysis and Visualization with Sensei and Ascent" tutorial at *SC '19 International Conference*. [Online]. Available: https://sc19.supercomputing.org/program/tutorials/

[6] U. Ayachit et al., "Performance Analysis, Design Considerations, and Applications of Extreme-Scale In Situ Infrastructures", *SC '16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, Salt Lake City, UT, 2016, pp. 921-932.

[7] Will Usher, Silvio Rizzi, Ingo Wald, Jefferson Amstutz, Joseph Insley, Venkatram Vishwanath, Nicola Ferrier, Michael E. Papka, and Valerio Pascucci, "LibIS: a lightweight library for flexible in transit visualization", in *Proceedings of the Workshop on In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization (ISAV '18)*, pp. 33–38, doi: 10.1145/3281464.3281466

[8] EPSN Project. [Online]. Available: https://www.labri.fr/projet/epsn/

[9] "Damaris: In Situ Data Analysis and Visualization in Support of Large-Scale CFD Application" presentation, given by Hadi Salimi at *29th International Conference on Parallel CFD (ParCFD)*, May 2017.

[10] Hummel M., van Kooten K., "Leveraging NVIDIA Omniverse for In Situ Visualization", in *High Performance Computing. ISC High Performance*, Weiland M., Juckeland G., Alam S., Jagode H., Eds. 2019. Lecture Notes in Computer Science, vol. 11887. Springer, Cham.

[11] USD: Universal Scene Description format. [Online]. Available:, https://graphics.pixar.com/usd/docs/index.html

[12] CinemaScience visual data format. [Online]. Available: https://cinemascience.org/

[13] Kress, James, Matthew Larsen, Jong Choi, Mark Kim, Matthew Wolf, Norbert Podhorszki, Scott Klasky, Hank Childs and David Pugmire, "Comparing the Efficiency of In Situ Visualization Paradigms at Scale.", in *High Performance Computing. ISC High Performance*, Weiland M., Juckeland G., Trinitis C., Sadayappan P. , Eds., 2019, Lecture Notes in Computer Science, vol. 11501. Springer, Cham.

[14] Conduit: a model for describing hierarchical scientific data. [Online]. Available: https://llnl-conduit.readthedocs.io

[15] J. Ames et al., "Low-Overhead In Situ Visualization Using Halo Replay," 2019 IEEE 9th Symposium on Large Data Analysis and Visualization (LDAV), Vancouver, BC, Canada, 2019, pp. 16-26.

[16] Averbukh V.L, Vasev P.A., Gorbashevsy D.Y., Kazantsev A.Y., Manakov D.V., "An interactive visualization system for parallel computing", in *Proceedings of 14-th International Conference on Computer graphics and Vision GraphiCon'2004*, Moscow state university named after M.V. Lomonosov, pp. 291-294, in Russian.