DOI: 10.14529/jsfi170303

Development and Integration of an In-Situ Framework for Flow Visualization of Large-Scale, Unsteady Phenomena in ICON

Michael Vetter[1], *Stephan Olbrich*[2]

With large-scale simulation models on massively parallel supercomputers generating increasingly large data sets, in-situ visualization is a promising way to avoid bottlenecks. Enabling in-situ visualization in a simulation model asks for special attention to the interface between a parallel simulation model and the data analysis part of the visualization, and to presentation and interaction scenarios. Modifications to scientific workflows would potentially result in a paradigm shift, which affects compute and data intensive applications generally. We present our approach for enabling in-situ visualization within the highly parallelized climate model ICON using the DSVR visualization framework. We focus on the requirements for generalized grid and data structures, and for universal, scalable algorithms for volume and flow visualization of time series. In-situ pathline extraction as a technique for the visualization of unsteady flows has been integrated in the climate simulation model ICON and verified in first studies.

Keywords: DSVR, ICON, in-situ visualization, visualization framework.

Introduction

High-resolution simulation of climate phenomena tends to produce very large data sets, which hardly can be processed in classical post-processing visualization applications. Typically, the visualization pipeline consisting of the processes data generation, visualization mapping, and rendering is distributed into two parts over the network or separated via file transfer [4, 6, 20]. Within most traditional post-processing scenarios, the simulation is executed on a supercomputer whereas data analysis and visualization is done on a graphics workstation. That way temporary data sets with huge volume have to be transferred over the network, which leads to bandwidth bottlenecks and volume limitations. A solution to this issue is the avoidance of temporary storage, or at least significant reduction of data complexity. This can be achieved by in-situ visualization, where the visualization is tightly coupled to the data generation [9]. In our work we focus on this topic, as well as on further challenges in extreme-scale visual analytics [19].

One actual climate simulation model is the ICON (Icosahedral non-hydrostatic) general circulation model, which was initiated by the Max Planck Institute for Meteorology (MPI-M) and the German weather service "Deutscher Wetterdienst" (DWD) [21]. Within the Climate Visualization Lab – as part of the Cluster of Excellence "Integrated Climate System Analysis and Prediction" (CliSAP) at Universität Hamburg, in cooperation with the German Climate Computing Center (DKRZ) – we enhance our in-situ approach and integrate it into the ICON framework.

The article is organized as follows. Section 1 introduces our DSVR framework [13] in the context of state-of-the-art visualization approaches. In section 2 we present the development steps to support generic grids. Section 3 addresses the integration and application of DSVR based flow visualization in ICON. The Conclusion summarizes the study and points directions for further work.

[1]Centrum für Erdsystemforschung und Nachhaltigkeit (CEN), Universität Hamburg, Germany
[2]Regional Computing Center (RRZ), Universität Hamburg, Germany

1. Parallel Data Extraction and Visualization in "DSVR"

As the data exchanged between the processes of the visualization pipeline decreases in volume, the level of exploration also decreases. So the partitioning of the visualization pipeline while designing a visualization system is always a trade-off decision between those two parameters. Common in-situ approaches fulfill the full visualization task on the supercomputer running the simulation and just store 2D pixel data within a movie. In another common approach, the visualization framework let a remote user connecting directly into the running simulation in order to allow live visualization of the running simulation. Well-known visualization systems utilizing one or both of these approaches are: Paraview Catalyst [1, 5] or VisIt [18]. An overview of most common in-situ visualization frameworks is given at [2]. We decided to follow the second approach, separating the process chain between the mapping and the rendering processes, since this enables real-time streaming while preserving user interactions like explorative camera changes, setting of lighting or time shifting within the visualization (see fig. 1).

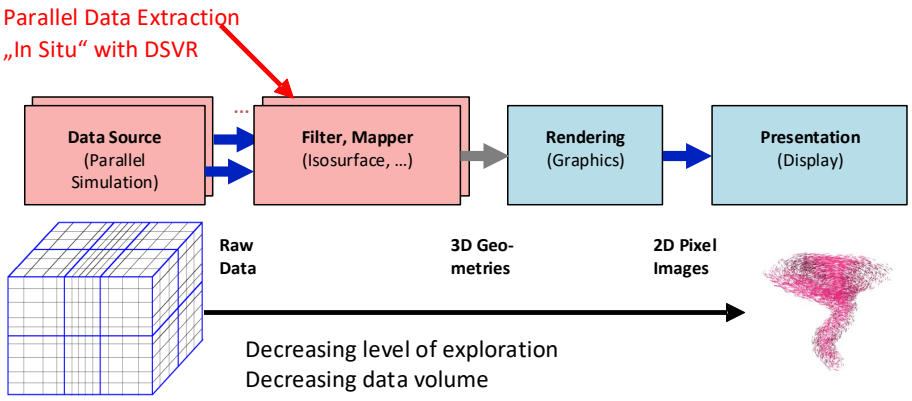

Figure 1. The visualization pipeline of DSVR

1.1. Distributed Streaming and Visualization Framework

In contrast to other visualization frameworks, where in-situ capabilities typically are built on top of existing post processing visualization applications, the DSVR framework [7, 13] was designed for in-situ visualization from scratch. This includes design and development of distributed software components, as well as network protocols and interfaces. As shown in fig. 2, the DSVR framework consists of three main components:

3D Generator: To enable in-situ visualization, the mapping process is tightly coupled to the simulation by calling methods of a parallelized data extraction library called libDVRP. This way the visualization mapping algorithms have access to the transient raw data in memory, using the domain decomposition of the calling simulation. The visualization library also combines 3D data compression with the mapping algorithms. This includes polygon reduction by adaptive vertex clustering within the isosurface generation [10], as well as compression of pathlines. Pathlines could be enhanced by local properties of the simulation to allow interactive post-filtering [15]. This results in a time-based sequence of geometric 3D scenes, interleaved with samples of raw data or extracts, which can be streamed and visualized in 3D.

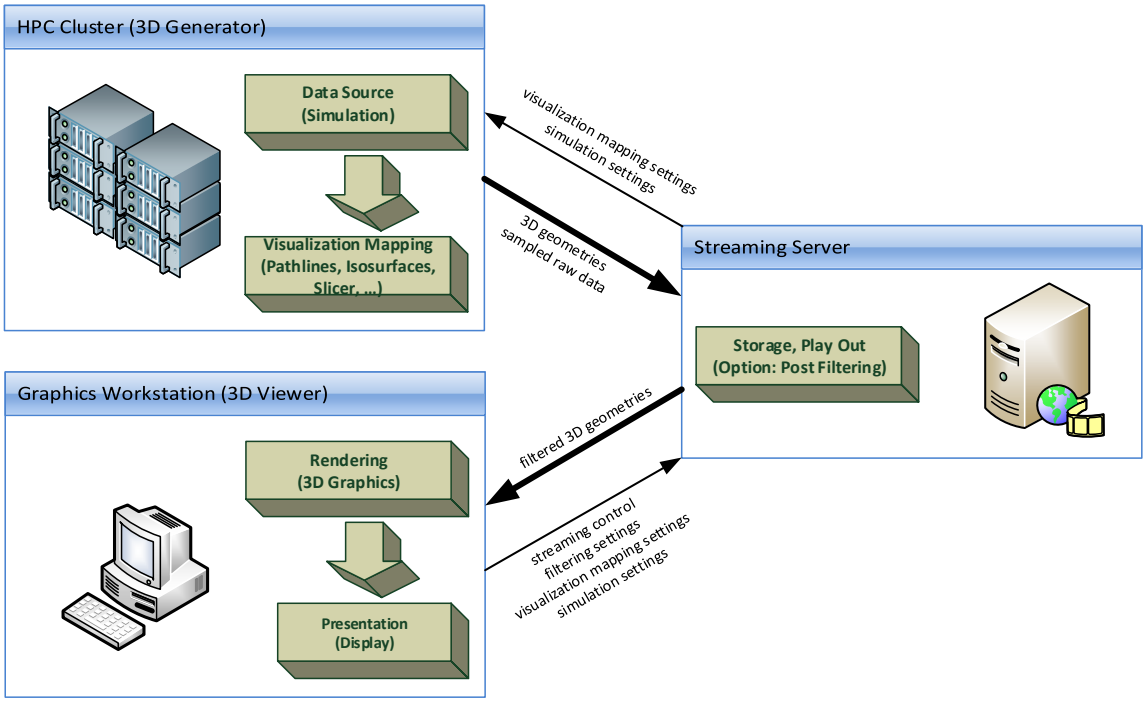

Figure 2. The DSVR framework consisting of libDVRP, streaming server, and rendering client

Streaming Server: A unique, advanced streaming feature of DSVR also has capabilities for further reduction of transferred data by server-side filtering of geometric objects based on sampled raw data as well as support of synchronization and control of frame rates. To support long running simulations, the 3D models can be stored on a separate persistent storage component, which is realized as a streaming server providing record and play functions and then played-out asynchronously.

3D Viewer: The 3D viewer client was first realized as a multi-platform browser plugin based on the NPAPI, and is now provided as a lightweight stand-alone version. Since the OpenGL based rendering is part of this viewer application, the scenes can be navigated interactively, optionally supported by stereoscopic 3D and tracking systems. In contrast to other in-situ approaches where 2D images are created as part of the simulation or a synchronous co-visualization takes place, our method supports interaction in 3D space and in time, as well as control of frame rates.

1.2. In-Situ Pathline Extraction

Techniques for visualization mapping of raw data in a three dimensional domain can be classified into volume visualization for scalar data and flow visualization for vector data. The integration of streamlines or pathlines using particle tracking algorithms is a common techniques for visualization of unsteady flow fields. Parallelization of particle tracking algorithms utilizes basically one of the following parallelization schemes: parallelization over seeds, parallelization over blocks (spatial domain and optional time), or hybrid approaches. For parallel performance of one or another parallelization scheme, key influencing factors are data set size, seed set size, seed distribution as well as vector field complexity [14]. Seeding strategies for pathline visualization

not only affect the algorithm's performance but also the visualization's insight. A lot of seeding strategies have been discussed for the integration of streamlines within steady flows, but they could not be applied straight forwardly to unsteady flows. Seed points can be evenly distributed all over the volume with some termination or filtering function to avoid visual cluttering. For better results the data set is preprocessed to find critical points of the vector field. An overview about geometric flow visualization including common seeding strategies is given in [12].

In our work we especially focus on techniques for visualization of time dependent scalar and vector fields. Some algorithms we implemented in DSVR are already well proven and highly optimized for the parallel visualization mapping and data reduction. Since the DSVR library was originally designed to be used with rectilinear grids, the implemented algorithms take direct advantages of the grid structure and are tightly bound to the grid. Parallel isosurface extraction with integrated flexible polygon simplification, as well as parallel pathline extraction for feature enhanced pathlines had been implemented and applied in oceanic and atmospheric simulations based on the parallel large-eddy simulation model PALM [11, 16]. Using the simulation's domain decomposition the parallelization of the pathline extraction is natively given as parallelization over blocks. Although time is a fourth dimension in time dependent flow visualization, parallelization over time steps is not an option due to mainly two reasons: (1) within an in-situ approach, only data of a few time steps can be hold in transient memory because of memory limitations, and (2) pathline extraction uses iterative, serial algortihms to trace particles over time steps. The seeding strategy would typically distribute a very large amount of seed points over the simulated volume, using a predefined pattern. Visual cluttering is reduced by interactive filtering during the streamed presentation, based on the enhancement of the pathlines with a set of properties gained from the simulation.

1.3. Support of Simulation Applications

To enable DSVR based in-situ visualization within a simulation, the data extraction library, called libDVRP, needs to be integrated within the simulation code. The library is written in C and provides an additional Fortran interface. libDVRP supports MPI-based parallel environments, and it encapsulates the handling of the transient data parts in 3D space and time, according to the given domain decomposition and iterative solution, and the serialization for file or streaming output of extracted 3D primitives or other data. The integration of the library into a simulation can be done easily by adding a few lines of code as shown in fig. 3. In most simulation codes a simple scheme can be found: First there is a bunch of initialization routines, then within a main loop for each simulated time step the model data is calculated, followed by finalization routines.

Within the simulation's initialization routines, libDVRP needs to be configured. First of all libDVRP needs to be initialized to set internal data structures to defaults and the output mode has to be selected. Here the options are writing 3D sequences directly to file or to the DSVR streaming server via TCP/IP. Then the simulation's grid configuration needs to be given to the library. After this the visualization has to be parametrized by setting seed points for pathlines or thresholds for isosurfaces for example.

At the end of the simulation's main loop, when all calculations are done, the data fields needs to be provided to the library. Then `DVRP_Visualize()` can be called to let libDVRP extract the 3D geometries. Within the simulation's finalization our library should be finalized.

```
PROGRAM simulation
    ! initialize simulation
    CALL DVRP_Initialize
    CALL DVRP_SetOutput
    CALL DVRP_SetGrid
    ! call methods for setting visualization parameters,
    ! eg. DVRP_Threshold, DVRP_Material, DVRP_Cubic_Seeding
    DO ! simulations main loop
        ! do the simulation step and calculate all those fields
        CALL DVRP_SetDataFields
        CALL DVRP_Visualize
    END DO
    CALL DVRP_Finalize
    ! finalize simulation
END PROGRAM simulation
```

Figure 3. Integration of DSVR in-situ visualization in a simulation code

2. Development of Visualization Methods on Generic Grids

ICON internally uses an icosahedral grid structure, which fits better for a spherical problem domain found in earth sciences. Since ICON also includes output modules for rectilinear grids, we had to do a design decision for integrating DSVR in ICON: Would we like to make usage of the ICON output routines and fit them for our needs or reimplement our algorithms for the ICON grid? Mapping of the ICON raw data onto rectilinear grids is attended by several issues: (1) grid transformation requires recalculating all needed data fields on all grid points, (2) using the same amount of grid points will lead to oversampling at the poles and to undersampling at the equator and (3) matching the grid point distance of the original grid with the rectilinear grid at the equator, won't lead to undersampling but needs a higher memory footprint due to oversampling. We decided to implement a highly generalized approach.

Given that other grid structures beside rectilinear and prism grids will get required in the future, we decided for a paradigm shift for the development of libDVRP. The overall goal is to implement visualization algorithms independent of the simulations' grid structure while preserving the possibility to optimize the in-situ visualization for the individual simulation data structures. This results in a gridAPI written in C, which abstracts the grid and data relevant operations from the visualization algorithms as known from object oriented programming approaches. For each new grid, a realization of each of the gridAPI's functions has to be implemented. The gridAPI will then act like a proxy redirecting the function calls.

The API is designed as an integrated software layer between the simulation and the visualization algorithms (fig. 4). It includes methods to be called by the libDVRP visualization routines as well as function to be called by the simulation. While the simulation uses "setter"-functions for setting grid type, grid parameters and data fields, the visualization routines typically use "getter"-functions to get data values or the MPI process, having the data at given coordinates or grid indices. In addition, it implements some sort of iterator to be used on the grid cells. The design of the gridAPI still allows for grid specific optimization, since the major getter methods allows unspecific optional optimization parameters. This could be used to store previously calculated cell indices or starting positions for search operations for example.

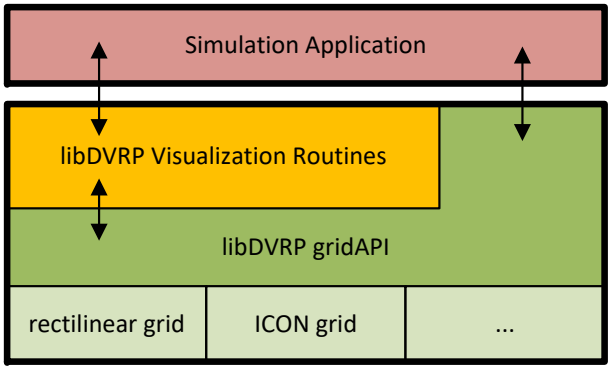

Figure 4. libDVRP gridAPI layout

In order to take usage of the gridAPI, a reimplementation of the visualization algorithms is needed. We implemented universal parallel algorithms for pathline extraction as well as for isosurface extraction, both applying the gridAPI.

2.1. Pathline Extraction

The algorithm for pathline extraction using the gridAPI was based on the MPI-parallelized pathline algorithm already introduced in [15, 16] with an optimized communication scheme as in fig. 5.

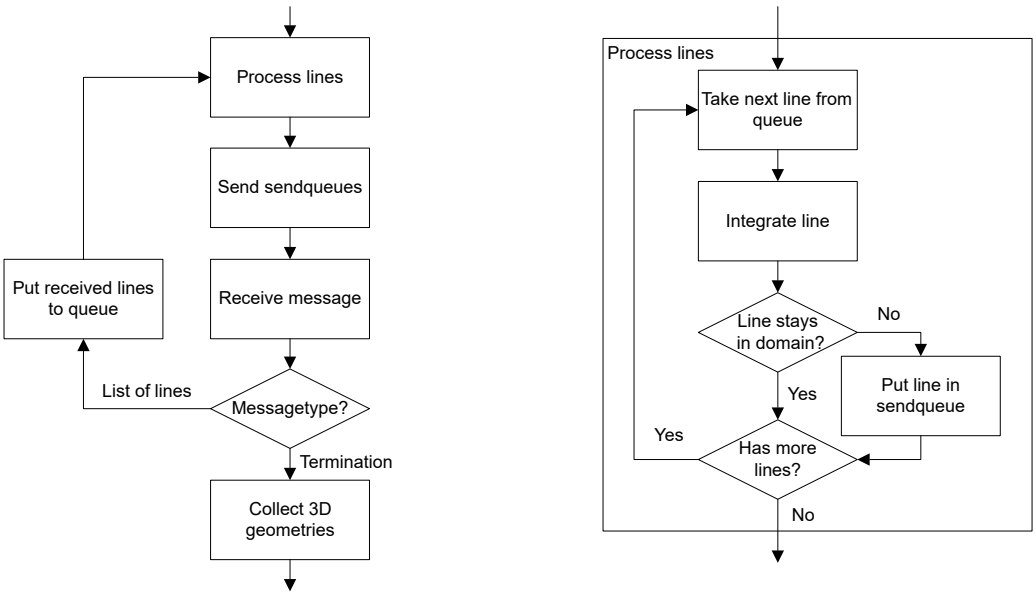

Figure 5. Process diagram of pathline extraction

As the parallelization is done using 1D, 2D, or 3D domain decomposition of the 3D data grid given by the simulation model, integration of pathlines is limited to the local domain by each MPI process. Parallelization of visualization over time steps is not possible (see 1.2), and generally not realized in simulation models. The data of at most two time steps is cached in libDVRP, to enable higher order numerical integration. On every call of `DVRP_Visualize()` each MPI process iterates over all lines within the domain and integrates the next supporting point for each line using Euler or Runge-Kutta integration. When a line leaves local domain of one process, it is

sent to the MPI process holding the needed data. For optimized MPI communication, traversing lines are buffered and asynchronously communicated after all local lines have been processed. After the new pathlines entering from neighbor domains are received, the line processing starts again.

Within this algorithm, only two functions of the gridAPI were used: while integrating the line, the data values at a given position are requested by calling `DVRP_gridAPI_getValAtPos()` up to four times for the Runge-Kutta 4th order integration. After this the MPI process handling the data at the resulting position is requested to find out if the line stays within the local domain. This is done by calling `DVRP_gridAPI_getMPIAtPos()`. This way the task of finding a specific grid cell, as well as the data layout of the raw data, is shifted into the gridAPI. This allows for optimizations regarding the grid and data access without touching the pathline algorithm anymore.

2.2. Isosurface Extraction

In order to be independent of the used grid structure, the isosurface algorithm has to be as general as possible. The marching cubes algorithm [8] used for isosurface extraction within libDVRP could not be abstracted from grid and data layout. Contemplable algorithms have to meet several constraints: parallelization based on domain decomposition as well as data volume optimized resulting 3D meshes are hard requirements. This leads to two possible algorithms: Complex-Valued Contour Meshing [17] and Isosurface Construction using Convex Hulls [3]. The first algorithm fractionizes all 3D cells into a bunch of tetrahedrons and creates an isosurface for each tetrahedron using a lookup table. The second algorithm would dynamically generate the appropriate mesh pattern lookup tables for the necessary cell configurations during the run-time initialization. Both algorithms have their own advantages and disadvantages. Though the first algorithm is more generalized since it is not limited to convex cells, in a comparison test on a prism grid it generates 6.7 times more triangles for the same isosurface. Therefore, we favored the second algorithm, as it would generate the appropriate mesh pattern lookup tables for the necessary cell configurations dynamically during the run-time initialization and result in lesser amount of 3D geometric primitives.

For this algorithm we need 6 additional functions within the gridAPI: `startCellIteration`, `getNextCellId`, `getCellPoints`, `getCellValues`, `getEdgeDefinition` and `getCellDefinition`.

We have evaluated the isosurface algorithm using the artificial tornado like swirl by Crawfis[3] generated by a self-written test application. The data field was generated and visualized on a prism grid and on a rectilinear grid using the same grid points. The new algorithm was compared with the Marching Cubes algorithm. Using our new algorithm on a prism grid with the same amount of grid point but double amount of cells, the algorithm produces 47 percent more triangles.

3. Integration of DSVR Based Pathline Extraction in ICON

In order to integrate an in-situ processing based on our DSVR framework and methods in the state-of-the-art climate simulation model ICON, we are continuously evolving data structures of the framework to support the ICON model's native grid structures. Therefore, we implemented

[3]http://www.cse.ohio-state.edu/ crawfis/Data/Tornado/tornadoSrc.c

a realization corresponding to the ICON grid within our gridAPI. ICON uses a dual grid, where the primary grid consists of a triangle mesh around the globe with parallel layers for the height axis. The secondary, indirectly stored grid consists of hexagons connecting adjacent triangles.

After reimplementation of the pathline extraction algorithm we have implemented a gridAPI realization for rectilinear grids as well as for the ICON grid. To get scientists a better understanding about what DSVR is capable of, we implemented a stand-alone NetCDF post-processor, based on libDVRP (fig. 6). By using the NetCDF post-processor the 3 processes simulation, visualization mapping, and rendering are separated completely: the data set is processed in a batch mode – e.g. using the same supercomputer on which the data is generated – and the interactive 3D rendering is done afterwards on the scientist's local system.

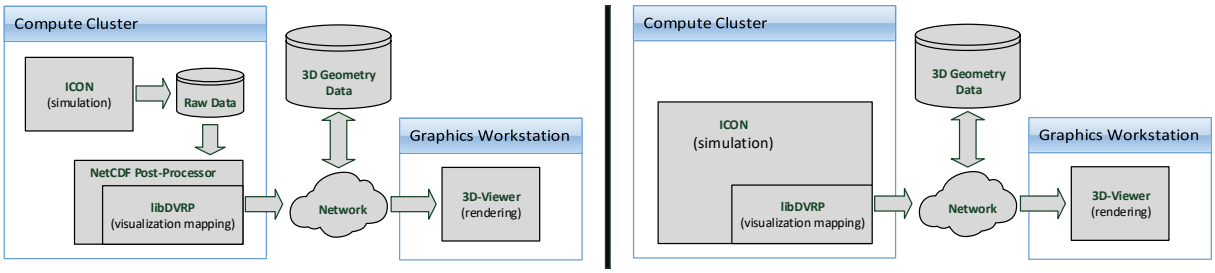

Figure 6. Architecture diagram of ICON visualization with DSVR using the NetCDF post-processor on the left side, and an in-situ visualization on the right

Since the NetCDF files do not necessarily contain any information about grid cells but only the coordinates of where the data is stored at, a gridAPI realization supporting leveled point sets was implemented. At the actual status of implementation, the post-processor supports the generation of isosurfaces and colored slicers on volume data set time series based on rectilinear grids as well as the visualization of pathlines on time varying flow fields based on either rectilinear grids or leveled point set grids (see fig. 7).

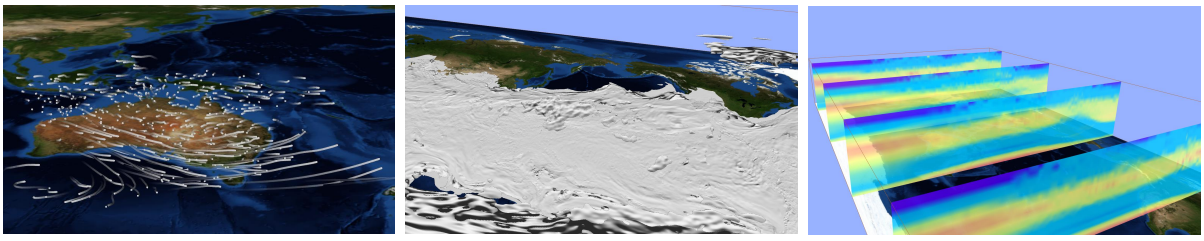

Figure 7. Sample visualizations of ICON data using the post-processor (from left to right): (a) pathlines of wind speed on ICON grid, (b) isosurface showing an atmospheric temperature of 273.15 K on rectilinear grid, and (c) colored slicer of atmospheric temperature on rectilinear grid

We have now implemented a new output module to ICON to take advantage of the DSVR visualization, which is called in the ICON simulation loop. The visualization can be configured as most output modules by using a specific namelist and is exemplarily integrated within the non-hydrostatic atmospheric model time loop. The module is initialized by a subroutine called `init_dvrp_output`. Here the ICON grid structure is collected, conditioned and set within libDVRP. Also the major configuration of the visualization is made here. The processing of every time step is done by a subroutine called `write_dvrp_output`, were data fields are copied to

libDVRP and the visualization routine `DVRP_Visualize` is called. ICON calculates most flow vectors like wind speed at the cell center point. So the original ICON prism grid could be used assuming the flow is the same everywhere within the cell. On the other hand, the hexagonal grid can be used, with the original cell center points becoming the new grid points. The second option would allow interpolation of flow values at every position leading to better visualization results, so the decision fell on the usage of the hexagonal grid for pathline extraction. Since each hexagonal cell can be broken down on triangles, we implemented the interpolation method as well as the cell-searching algorithm on prism cells, taking advantage of the known clustering.

With the integration of a DSVR-based in-situ pathline extraction within ICON, the next milestone is reached. The pathline algorithm as well as the grid data structures has been optimized for the domain decomposition used for the parallelization of ICON based on MPI and OpenMP. Software implementation and evaluation is also done on the supercomputer "Mistral" at DKRZ. All computation was done using Mistral's standard compute nodes with two 12-core Intel Xeon E5-2680 v3 processors at 2.5 GHz and 64 GB main memory each. In principle, the data complexity is reduced from $O(n^3)$ to $O(m)$, where n is the compute grid resolution of the simulation model and m is the number of supporting point of all pathlines. Since the amount of pathlines as well as the amount of supporting points per pathline are given by the users, m is adjustable. The number of supporting points per pathline should be choosen relating to the simulation's resolution. The number of pathlines, on the other hand, should be constant in order to prevent visual cluttering. Therefore, m will somehow be scaling with n, and the overall reduction of complexity can be estimated with n^2. The evaluation of stability and scalability is done using Atmospheric Model Intercomparison Project (AMIP) runs which were used for testing the model's changes.

Number of Nodes	ICON runtime without visualization	ICON runtime with visualization
1	437 s	486 s
2	231 s	257 s
4	131 s	149 s
8	78 s	96 s
16	68 s	86 s
32	47 s	65 s

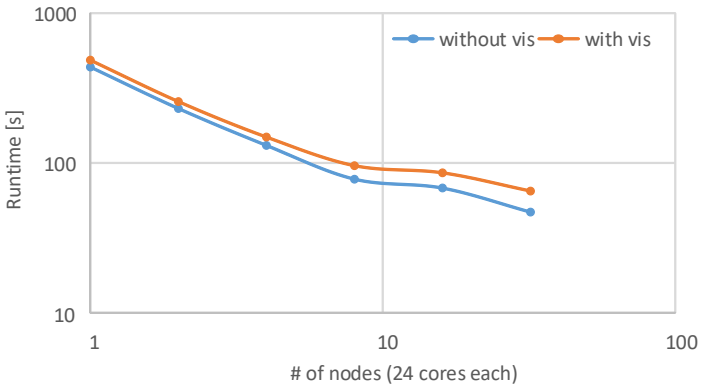

Figure 8. Time measurement of ICON ATM AMIP runs on a 20480 * 47 grid (160 km) for 7 days (1008 time steps). Simulation based on 2D domain decomposition. In-situ visualization of 2000 pathlines with 100 supporting points. Strong scaling

In fig. 8 the results of such test run are shown, comparing the overall run times of ICON with and without DVSR visualization. This case uses a low grid resolution of 160 km (20480 grid points per layer) and runs for 7 simulated days writing out 1008 time steps. Scalability tests have been done up to 32 compute nodes, which means a maximum of 768 cores. Pathline algorithms generally don't scale very well on domain decomposition, since the extraction of pathlines does not take that much computation time at all. Most time is consumed by finding the value for a given position, and this has only to be done four times per pathline assuming the

usage of the Runge-Kutta 4th order. Thereby the visualization time depends mainly on the MPI communication. Rising visualization time with increasing core count is a tribute to the domain decomposition, which may lead to more or less lines to alternate processes. Up to the tested 32 nodes, the visualization does not cause exceeding increase in the overall runtime of this strong scaling test. We expect that weak scaling should show better results, since the relation between computational load and communication cost would be better in that scenario.

Beside the runtime test within ICON, we have also tested the new pathline algorithm using an artificial tornado by Crawfis on a large prism grid containing 1.44 million grid points per layer multiplied with 200 layers to get an impression on the scalability (see fig. 9). This is approximately the same amount of grid points as the ICON run on the 20 km grid has. The prism grid was generated by cutting each voxel of a regular grid into two prisms.

Number of Nodes	runtime with visualization
1	1358 s
2	442 s
4	128 s
8	26 s
16	25 s
32	21 s

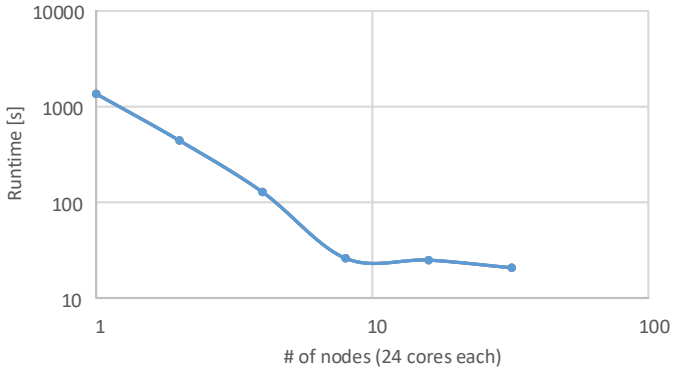

Figure 9. Visualization time measurement of an artificial tornado 1.44 million x 200 ICON grid. Simulation based on 2D domain decomposition. In-situ visualization of 2000 pathlines with 100 supporting points

The higher runtime is caused by the `DVRP_gridAPI_getValAtPos()` function scaling with the grid size. This also scales only up to 8 compute nodes, since the simulation part was not compute-intensive, and because of the strong scaling method.

Conclusion and Future Work

In order to enable in-situ visualization within the ICON climate model using our DSVR framework, we had to redesign one of its core components. With the design of the gridAPI, a fundamental generalization is introduced for the parallelized data extraction library libDVRP. This way, a support for actual and next-generation simulation models can easily be added. Also, redesign and implementation of our visualization algorithms are required. For a start, algorithms for pathline extraction as well as isosurface generation have been implemented. Both algorithms have already been tested on artificial test scenarios. The pathline extraction has been integrated in the NetCDF post-processor, as well as an in-situ visualization option in ICON.

In future work we plan to integrate the isosurface algorithm in ICON. Furthermore, large scale performance and stability evaluation of flow visualization and volume visualization will be done on high resolution scenarios.

Acknowledgement

We would like to thank Tim Rolff for supporting our work with his bachelor thesis, as well as Marco Giorgetta from Max Planck Institute for Meteorology for the successful cooperation. We also like to thank the DKRZ for providing access to their supercomputer as well as fruitful discussions. This work was supported through the Cluster of Excellence CliSAP (EXC177), Universität Hamburg, funded by the German Research Foundation (DFG).

References

1. Ayachit, U., Bauer, A., Geveci, B., O'Leary, P., Moreland, K., Fabian, N., Mauldin, J.: Paraview catalyst: Enabling in situ data analysis and visualization. In: Proceedings of the First Workshop on In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization. pp. 25–29. ISAV2015, ACM, New York, NY, USA (2015), DOI: 10.1145/2828612.2828624

2. Bauer, A.C., Abbasi, H., Ahrens, J., Childs, H., Geveci, B., Klasky, S., Moreland, K., O'Leary, P., Vishwanath, V., Whitlock, B., Bethel, E.W.: In situ methods, infrastructures, and applications on high performance computing platforms. In: Proceedings of the Eurographics / IEEE VGTC Conference on Visualization: State of the Art Reports. pp. 577–597. EuroVis '16, Eurographics Association, Goslar Germany, Germany (2016), DOI: 10.1111/cgf.12930

3. Bhaniramka, P., Wenger, R., Crawfis, R.: Isosurface construction in any dimension using convex hulls. IEEE Transactions on Visualization and Computer Graphics 10(2), 130–141 (Mar 2004), DOI: 10.1109/TVCG.2004.1260765

4. Brodlie, K.W., Duce, D.A., Gallop, J.R., Wood, J.D.: Distributed cooperative visualization. In: Proceedings of Eurographics'98 State of the Art Reports. pp. 27–50 (1998)

5. Fabian, N., Moreland, K., Thompson, D., Bauer, A.C., Marion, P., Gevecik, B., Rasquin, M., Jansen, K.E.: The paraview coprocessing library: A scalable, general purpose in situ visualization library. In: 2011 IEEE Symposium on Large Data Analysis and Visualization. pp. 89–96 (Oct 2011), DOI: 10.1109/LDAV.2011.6092322

6. Haber, R.B., Mc Nabb, D.A.: Visualization Idioms: A Conceptual Model for Scientific Visualization Systems. In: Visualization in Scientific Computing (1990)

7. Jensen, N., Olbrich, S., Pralle, H., Raasch, S.: An efficient system for collaboration in tele-immersive environments. In: Proceedings of the Fourth Eurographics Workshop on Parallel Graphics and Visualization. pp. 123–131. EGPGV '02, Eurographics Association, Aire-la-Ville, Switzerland, Switzerland (2002), `http://dl.acm.org/citation.cfm?id=569673.569695`

8. Lorensen, W.E., Cline, H.E.: Marching cubes: A high resolution 3d surface construction algorithm. In: Proceedings of the 14th Annual Conference on Computer Graphics and In-

teractive Techniques. pp. 163–169. SIGGRAPH '87, ACM, New York, NY, USA (1987), DOI: 10.1145/37401.37422

9. Ma, K.L., Wang, C., Yu, H., Tikhonova, A.: In-situ processing and visualization for ultra-scale simulations. Journal of Physics: Conference Series 78(1), 012043 (2007)

10. Manten, S., Breuer, I., Olbrich, S.: Parallel isosurface extraction including polygon simplification via self adapting vertex clustering. In: The Ninth IASTED International Conference on Visualization, Imaging and Image Processing (VIIP-2009). Cambridge, UK (2009)

11. Manten, S., Vetter, M., Olbrich, S.: Evaluation of a scalable in-situ visualization system approach in a parallelized computational fluid dynamics application. In: Brunnett, G., Coquillart, S., Welch, G. (eds.) Virtual Realities: Dagstuhl Seminar 2008. pp. 225–238. Springer Vienna, Vienna (2011), DOI: 10.1007/978-3-211-99178-7_12

12. McLoughlin, T., Laramee, R.S., Peikert, R., Post, F.H., Chen, M.: Over Two Decades of Integration-Based, Geometric Flow Visualization. Computer Graphics Forum (2010)

13. Olbrich, S., Pralle, H., Raasch, S.: Using streaming and parallelization techniques for 3d visualization in a high performance computing and networking environment. In: Hertzberger, B., Hoekstra, A., Williams, R. (eds.) High-Performance Computing and Networking: 9th International Conference, HPCN Europe 2001 Amsterdam, The Netherlands, June 25–27, 2001 Proceedings. pp. 231–240. Springer Berlin Heidelberg, Berlin, Heidelberg (2001), DOI: 10.1007/3-540-48228-8_24

14. Pugmire, D., Peterka, T., Garth, C.: Parallel integral curves. In: Bethel, E.W., Childs, H., Hansen, C. (eds.) High-Performance Visualization - Enabling Extreme-Scale Scientific Insight, chap. 6. CRC Press (2012), DOI: 10.1201/b12985-9

15. Vetter, M., Manten, S., Olbrich, S.: Exploring unsteady flows by parallel extraction of property-enhanced pathlines and interactive post-filtering. In: Proceedings of 14th Eurographics Symposium on Virtual Environments (EGVE 2007), Posters. pp. 9–12 (2008)

16. Vetter, M., Olbrich, S.: Scalability issues of in-situ visualization in parallel simulation of unsteady flows. In: Bischof, C., Hegering, H.G., Nagel, W.E., Wittum, G. (eds.) Competence in High Performance Computing 2010: Proceedings of an International Conference on Competence in High Performance Computing, June 2010, Schloss Schwetzingen, Germany. pp. 177–190. Springer Berlin Heidelberg, Berlin, Heidelberg (2012), DOI: 10.1007/978-3-642-24025-6_15

17. Weigle, C., Banks, D.C.: Complex-valued contour meshing. In: Proceedings of the 7th Conference on Visualization '96. pp. 173–180. VIS '96, IEEE Computer Society Press, Los Alamitos, CA, USA (1996), DOI: 10.1109/VISUAL.1996.568103

18. Whitlock, B., Favre, J.M., Meredith, J.S.: Parallel in situ coupling of simulation with a fully featured visualization system. In: Proceedings of the 11th Eurographics Conference on Parallel Graphics and Visualization. pp. 101–109. EGPGV '11, Eurographics Association, Aire-la-Ville, Switzerland, Switzerland (2011), DOI: 10.2312/EGPGV/EGPGV11/101-109

19. Wong, P.C., Shen, H.W., Johnson, C.R., Chen, C., Ross, R.B.: The top 10 challenges in extreme-scale visual analytics. IEEE computer graphics and applications 32(4), 63–67 (2012)

20. Wood, J., Brodlie, K., Wright, H.: Visualization over the world wide web and its application to environmental data. In: Proceedings of the 7th conference on Visualization '96. pp. 81–86. VIS '96, IEEE Computer Society Press, Los Alamitos, CA, USA (1996), DOI: 10.1109/VISUAL.1996.567610

21. Zängl, G., Reinert, D., Rípodas, P., Baldauf, M.: The icon (icosahedral non-hydrostatic) modelling framework of dwd and mpi-m: Description of the non-hydrostatic dynamical core. Quarterly Journal of the Royal Meteorological Society 141(687), 563–579 (2015), DOI: 10.1002/qj.2378