Towards Interactive HPC: Sliding Window Data Transfer

Ralf-Peter Mundani[1], Jérôme Frisch[1], and Ernst Rank[1]

Technische Universität München, Chair for Computation in Engineering, 80333 München, Germany

Abstract

Interactive high-performance computing is doubtlessly beneficial for many computational science and engineering applications whenever simulation results should be visually processed in real time, i. e. during the computation process. Nevertheless, interactive HPC entails a lot of new challenges that have to be solved – one of them addressing the fast and efficient data transfer between a simulation back end and visualisation front end, as several gigabytes of data per second are nothing unusual for a simulation running on some (hundred) thousand cores. Here, a new approach based on a sliding window technique is introduced that copes with any bandwidth limitations and allows users to study both large and small scale effects of the simulation results in an interactive fashion.

Keywords: interactive HPC, sliding window, computational fluid dynamics

1 Introduction and Motivation

Due to recent advances in supercomputing, more and more application domains such as medicine or geosciences profit from high-performance computing (HPC) and come up with (new) complex and computationally intensive problems to be solved. While solving larger problems is just one objective, another one is to solve problems much faster – even in real time, thus bringing them into the range of interactive computing. Meanwhile many supercomputing centres provide users not only batch access to their HPC resources, but also allow an interactive processing which is in many ways advantageous. The probably most prominent representative of such an approach is *computational steering* [1] where a simulation back end running on a supercomputer or compute cluster is coupled to a visualisation front end for interaction. Hence, users have the possibility to manipulate certain parameters (geometry, boundary conditions, algorithm control etc.) in order to experience immediate feedback from the running simulation.

Proceedings of the Third International Conference on Parallel, Distributed, Grid and Cloud Computing for Engineering (Paper 50), 2013
DOI: 10.4203/ccp.101.50

At the moment, users can choose from a widespread toolkit of libraries, frameworks, and problem solving environments related to computational steering such as *CUMULVS* [2], *RealityGrid* [3], *Magellan* [4], *SCIRun* [5], or *COVISE* [6] – just to name a few. While those steering approaches differ in the way they provide interactive access to those parameters to be steered/visualised, using check- and breakpoints, satellites connected to a data manager, or data flow concepts, e. g., they are usually of limited scope concerning different application domains and/or entail severe code changes. A somewhat generic and 'minimal invasive' concept based on the idea of signals and interrupts has been developed by our group that was successfully tested with different applications from various domains (amongst others physics, medicine, and biomechanics) [7,8]. This concept should now be extended by the idea of a sliding window technique for fast and efficient data transfer between back and front end, especially in case of high-resolution multi-scale simulations.

One major problem within computational steering or interactive computing is the handling of the data transfer between the simulation back end and the interaction front end for visual display of the simulation results. Especially in HPC applications, where a vast amount of data is computed every second, such data advent easily exceeds the capacities, i. e. bandwidth, of any underlying interconnect[1] and, thus, hinders any interactive processing. Hence, sophisticated techniques are inevitable for selecting and filtering the respective data already on the back end in order to cope with physical limitations and to transfer those data only that are really to be visualised. While such an approach is doubtlessly necessary to leverage interactive HPC, on the other hand any selection and/or filtering process discards delicate details of the results that in most cases were computed on a much finer resolution than they are displayed for visualisation. So that users can select any region of interest within the computational domain to be displayed in any granularity (i. e. the density of results contributing for visual output) we have developed a sliding window technique that allows for the interactive visual display ranging from coarse (quantitative representation of the entire domain) to fine scales (any details computed on the finest resolution) and which is one important step towards interactive high-performance computing.

By selecting the region of interest – the window – on the front end for data transmissions from the back end, a sufficient data structure on the back end becomes necessary, that provides simple and fast access to any subset of the data. Therefore, we have developed a distributed hierarchical data structure that intrinsically supports efficient data access concerning both random choice of subsets and random choice of density. In order to put everything into practice, we have further implemented a 3D flow solver that carries out its computations on the aforementioned data structure, running in parallel on medium and large size compute clusters and supercomputers, letting us investigate different flow scenarios from simple setups to complex multi-scale problems. In combination with the sliding window data transfer, users have now the choice to retrieve results from a running HPC application to study either large-scale effects (such as flow around a large building or entire city) or small-scale details (such as vor-

[1] An apposite statement of Bill Gropp to this is *"latency is physics, bandwidth is money"*.

tices or local effects) in an interactive fashion. This not only opens the door to many possible scenarios from computational science and engineering, but it also paves the way for interactive HPC which is certainly beneficial for many kinds of optimisation or design problems from various application domains.

The remainder of this paper is as follows. In chapter 2 we discuss the hierarchical data structure and its application for a parallel flow solver, in chapter 3 we introduce the concept of the sliding window and its usage for the interactive data access from a running application. Chapter 4 highlights first results obtained with the sliding window technique and chapter 5, finally, provides a short conclusion and outlook.

2 Fundamentals

2.1 Computational Fluid Dynamics

As implementation example for demonstrating the sliding window data transfer concept we have chosen a parallel computational fluid dynamics (CFD) code currently under development at the Chair for Computation in Engineering at Technische Universität München.

The fluid flow computation is governed by the 3-dimensional Navier-Stokes equations for an incompressible Newtonian fluid with no acting external forces:

$$\boldsymbol{\nabla} \cdot \boldsymbol{u} = 0 \quad , \tag{1}$$

$$\frac{\partial}{\partial t}\boldsymbol{u} + (\boldsymbol{u} \cdot \boldsymbol{\nabla})\,\boldsymbol{u} = -\frac{1}{\rho}\boldsymbol{\nabla} p + \nu \Delta \boldsymbol{u} \quad . \tag{2}$$

As spatial discretisation, a finite volume scheme is used. The temporal discretisation is realised by a finite difference scheme, namely an explicit Euler method of 2nd order. A collocated cell arrangement storing all values at the cell centre was chosen together with a pressure oscillation stabilisation method. Numerically, a fractional step method is applied, which is based on an iterative procedure between velocity and pressure during one time step. After an intermediate velocity v^* is computed by neglecting any influence of the pressure field, a Poisson equation is solved, guaranteeing a divergence free pressure distribution. The pressure correction is then added to the intermediate velocity resulting in the velocity at the next time step v^{n+1}. Detailed information about the numerical solver as well as validation results can be found in [9].

2.2 Data structure

The data structure of the fluid solver was built from adaptive non-overlapping block-structured equidistant Cartesian grids with a ghost layer halo that favours the sliding window data transfer concept enormously. The code is designed for a full 3D simulation, but for simplifying matters, only 2D grids and domains are drawn in this paper. Figure 1 depicts different grid levels with different discretisations and their linkage (upper part) as well as a view of the 'assembled' domain consisting of all those grids

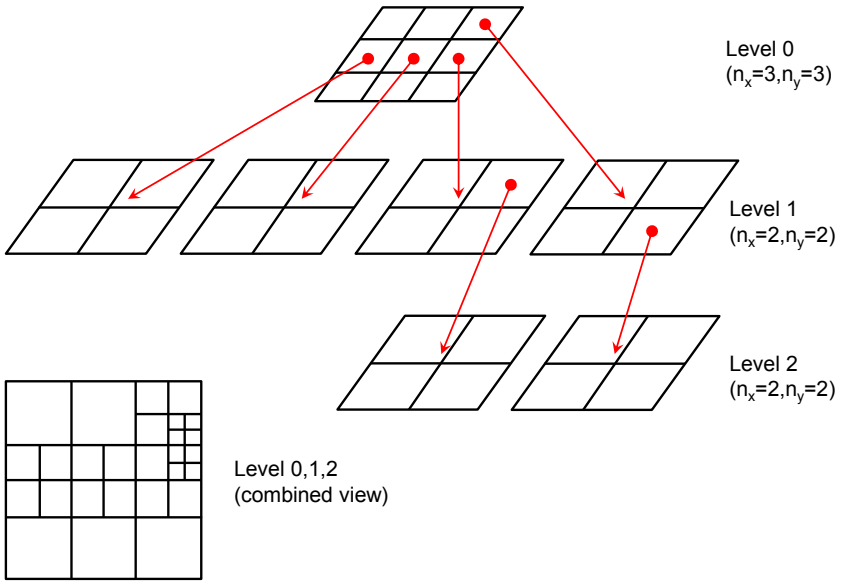


Fig. 1. Hierarchical grid construction over different levels using different discretisations (upper part), and a combined view of the 'assembled' domain including all grids of this structure (lower left part).

(lower part). A grid can be divided into $n_x \times n_y \times n_z$ sub-grids with different values for n_x, n_y, and n_z. Furthermore, for level 0 a different discretisation than for levels ≥ 1 can be chosen. This enables an easy creation of a tunnel shaped domain, e. g., while not losing too much 'non-computing' cells. A value $n_i = 1$ determines that in i-direction no refinement takes place over the different hierarchy levels. Obviously, this only makes sense if a 'pseudo'-2D computation should be performed using the 3D code. A value of $n_i = 2$ $(i = x, y, z)$ on all levels would describe a regular octree data structure.

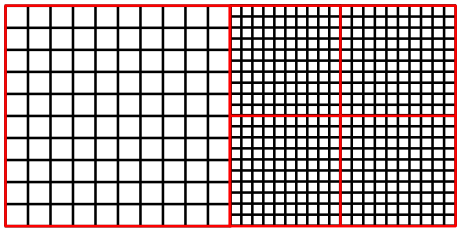


Fig. 2. Two adjacent grids (red boundaries) on different levels with their 10×10 cell discretisation (black).

Each grid consists of $nc_x \times nc_y \times nc_z$ cells stored in arrays at grid level that contain the actual computing values in a collocated arrangement (i. e. in the cell centre). In order to avoid non matching blocks, the following restraint is enforced: nc_i has to be dividable without remainder by n_i, $i = x, y, z$ on any level. This ensures that a matching on both sides can be found. In the case of Figure 2, the level discretisation is set to $n_x = n_y = 2$ with a cell amount of $nc_x = nc_y = 10$. This means, that one cell in the grid on a higher level has two neighbours in the grid on a lower level.

In our hierarchical data structure, grids and corresponding cells are not deleted when a parent grid is refined into child grids. This means, that the cell values are still present, but are not actively involved in the computation step. They merely get filled by interpolated values of the sub-grids during the communication and exchange step briefly described in the following sub-section.

2.3 Parallel Implementation and Communication Scheme

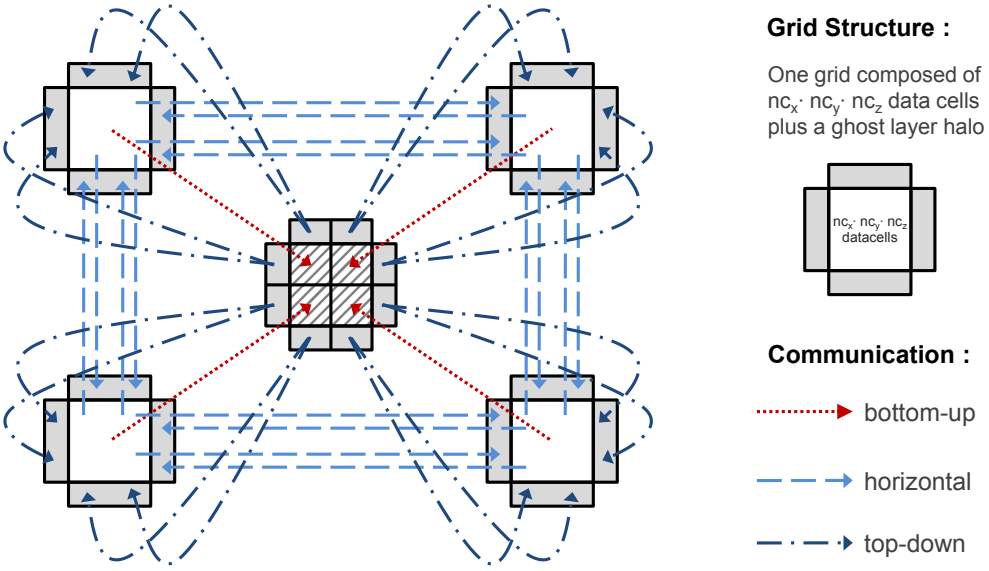


Fig. 3. Applied communication scheme for one grid at level 0 (middle grid) and 4 sub-grids on level 1 (outer grids). Each grid is composed of $nc_x \times nc_y \times nc_z$ data cells. The three different communication steps are depicted by different arrows and colours.

The parallel implementation is realised by applying a message passing paradigm, namely using MPI [10]. The code is constructed in such a way that a strict separation of computation and communication is ensured, thus enabling programmers with different specialities to focus on their respective area of expertise.

The communication depicted in Figure 3 can be divided into three different steps which require synchronisation in order to ensure a correct data flow. The first communication step can be described as a bottom-up approach. All grids average their data stored in the arrays and send them to the super-grid in order to get stored in the corresponding super-grid cell. Thus, the data travels from the bottom-most grids – where the actual computation is done – to the top-most grid which stores the most coarse representation of simulation data. Once the averaged data is stored at the right position, the next step can be applied.

The second step can be categorised as horizontal communication, as here only neighbouring grids on the same level communicate. The neighbouring information can be retrieved from a dedicated neighbourhood server acting as topological repository

and answering queries about possible neighbourhood relations in the grid structure [11]. If a grid has a neighbour, the data is send immediately from this grid to the respective ghost layer halo in the remote grid. Possible problems which will arise due to bottlenecks while increasing the amount of processes are also discussed and remedied in [11].

If there was no neighbour present for a certain grid, the ghost layer halo will be filled during the third and final step: the top-down approach. Here data from the parent grid is sent to the ghost layer halo of the sub-grid in order to fill all sides which have not received any data during the horizontal communication step. This procedure also ensures that the data is correctly arriving at grid boundaries, where more than one level difference is present.

The sending and receiving of data is performed by a mixture of blocking and non-blocking calls ensuring a correct synchronisation behaviour. Further detailed information can also be taken from [12].

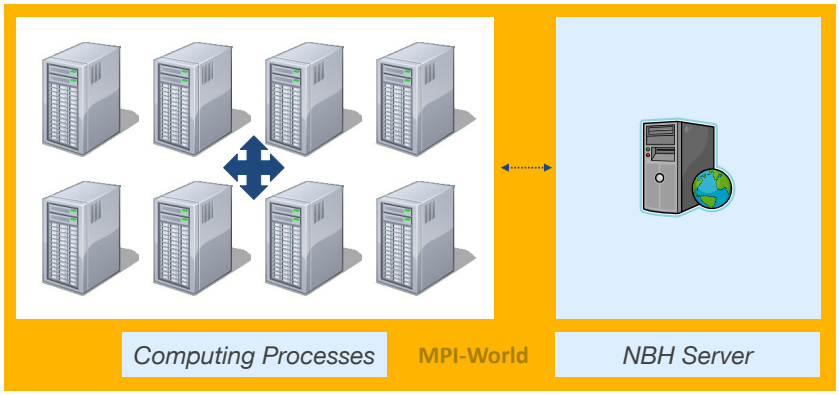


Fig. 4. Grouping of processes: Most processes will fulfill computational tasks only, but some might only work on scheduling and organisational tasks. Note that the thickness of the arrows is an indication for the amount of communication between the computing processes themselves and between them and the neighbourhood (NBH) server.

The different grids will now be distributed among the available processes in the MPI world (Figure 4). For the moment one process is reserved acting as neighbourhood server governing all the relations between the different grids. Please note that the neighbourhood server itself does not contain any computational data, it has a strictly topological view of all grids.

For distributing the grids, the Morton- or Z-order space filling curve is used, as it can be computed quite fast and easily by bit interleaving [13]. The grids are ordered according to the curve and distributed onto the available computing processes in order to ensure a good initial load distribution. If a lot of adaptive changes are performed in the computational domain, a balancing of the load has to be done at some point to ensure a good load distribution over time.

It can be concluded so far, that the structure and design of our hierarchical grid is very well suited for the sliding window concept as every intermediate level (with inter-

polated results) is fully present and, thus, a random selection process of data and granularity is strongly supported. A global server can further answer immediately which grids are involved on which processes solely by their IDs only.

3 Sliding Window

3.1 Concept

The basic idea of the sliding window concept is to limit the necessary data transfer between back and front end by selecting only subsets of the available computed results. Hence, any user should have the possibility to choose both a window, i. e. a desired region of the computational domain and a desired data density for the respective data to be visualised on the front end. This window can be slid over the computational domain as well as being increased or decreased in size, always determining which subset of data should be selected for the transfer. Whereas the window itself acts as a bounding box for the selection, the density defines the amount of data points inside that region to be considered for the visual display. Key feature of the entire approach is to keep the density constant while the window is being moved or resized, thus at any time the same amount of data is transmitted from the back to the front end. This allows us to optimally exploit the underlying network without exceeding any bandwidth limitations or causing unnecessary long transmission delays that would harm the experience of an authentic interactive computing.

In case of high-resolution data, a full window covering the entire computational domain would force the back end (depending on the chosen density) to skip lots of data points for the data transfer – for instance selecting only ever fifth or tenth data point in every dimension. Hence, the user would get a quantitative representation without too many details which more or less allows to catch global effects of the running simulation. When resizing the window, i. e. making it smaller, a smaller region of interest is covered and, thus, in order to keep the density constant, less data points are to be skipped, providing more and more details for the visual display. The window size can be further decreased until the resolution of the computational domain is met, i. e. now each single data point inside the selected region is shown on the visual front end. Obviously, the critical questions are how to select the right data points (cf. density) and in case of a parallel simulation how to select the right processes (cf. window).

We will see in the next section that our data structure is advantageous for the sliding window concept, as due to the hierarchy of grids together with their update scheme an implementation of this concept is straightforward and questions concerning the right selection of data points and processes are very easy to answer.

3.2 Implementation

The implementation of the sliding window concept is based on two major components, namely a server-side collector, responsible for collecting and gathering the requested

information on the back end, and a client-side visualisation and interpreter tool at the front end for sending data requests and receiving the corresponding data from the back end.

Server-Side Collector In order to keep the influence between the data collection process and the actual computation processes to a minimum, a special dedicated collector process was introduced into the above described CFD code. The main structure of the collector is an infinite loop listening on a TCP socket on a dedicated port for a single character describing the next task for execution. One possible command from the client-side to the server-side would be to do a sliding window visualisation.

Once the client send the task initialisation, it will continue sending the necessary information, such as visualisation data type, maximum amount of cells and the visualisation bounding box to the server. The collector receives this information and submits a query to the neighbourhood server, which has a topological and geometrical overview of the stored grids. The neighbourhood server performs for every grid an intersection test along the x, y, and z axis of the grid's bounding box and the transmitted sliding window bounding box. If the grid touches or lies completely inside the sliding window box, it is marked and added to the treatment list. If it is completely outside, the grid and all its sub-grids are ignored henceforth. The order of traversal is given by the same space filling curve used for load distribution. Thus, the grid identification is executed from the coarsest top grids to the more and more fine grids in deeper levels.

Once the maximum amount of cells to display is reached and the relevant grids have been identified, the neighbourhood server orders the corresponding processes containing the grids to send a data stream to the collector process. For this, MPI messages using the very fast dedicated cluster interconnect are exchanged. The collector then reforms and compresses the data in a way that it can be sent over a possibly slower TCP connection to the client. During the time the send operation is executed, the computing processes can continue with their regular simulation task without being blocked by the server-to-client transmission.

Instead of posting a visualisation task, the client can also interactively modify settings on the back end, such as ordering grid refinements, changing boundary conditions and cell types, or moving geometry through the domain. In this way, an interactive steering of the code is possible while at the same moment an interactive visualisation is able to switch through different scales from very large to very fine.

Client-Side Visualisation The current visualisation is realised using ParaView [14], an open source scientific visualisation tool capable of handling large data sets. ParaView is a graphical user interface for the visualisation toolkit VTK [15], an open source library for 3D computer graphics, image processing, and visualisation. In order to connect the visualisation immediately to the simulation side, a ParaView plug-in was written, incorporating the actual client implementation.

Figure 5 shows a screen shot of the ParaView GUI with the plug-in details on the lower left side. The user can enter the fluid server name and port as well as the data

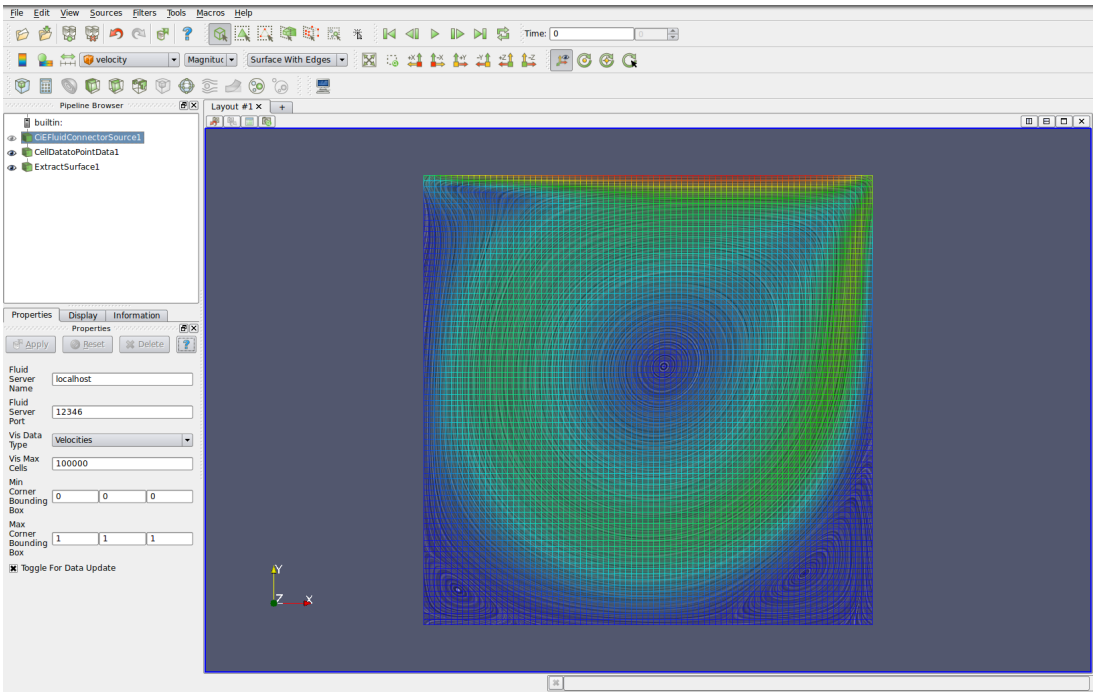


Fig. 5. ParaView as visualisation front end enriched by a plug-in for the direct connection to the simulation results computed on the back end (cluster or supercomputer).

type to transmit, the maximal amount of cells to display, and the bounding box coordinates of the sliding window visualisation. The plug-in connects via TCP sockets to the server and checks some prerequisites, such as the endianness and size of standard data types. If all prerequisites match, a connection can be established, and among other tasks, a visualisation can be requested (as described in the previous section). The bounding box coordinates, as well as the visualisation data type, and amount of cells to display are sent to the back end, which queries internally for the right data and delivers the results back to the front end for visualisation. Internally, the plug-in receives a long data stream which gets decoded and stored in the internal VTK unstructured grid data structure which then can be immediately visualised by ParaView.

4 CFD Results

For demonstrating the capabilities of the sliding window concept, the standard lid driven cavity benchmark was chosen. It is described in [16] and consists of a domain of one by one meter with four no-slip walls where the top no-slip wall is moving with a constant velocity of one, thus, exciting the internal fluid by shear stress only.

As geometrical discretisation, the grid subdivision was chosen to $n_x = n_y = 2$ up to level 3 and $nc_x = nc_y = 10$ as cell amount.

Figure 6 shows different visualisation areas selected in the sliding window display keeping the amount of data transferred over the network constant. Figure 6(d) shows a graphical representation of which bounding boxes, i. e. windows, were used for data

selection. 6(a) shows the complete domain with a coarse resolution of 400 cells. It is very important to stress the fact, that the resolution of the computation is different from the actual resolution of the visualisation given by the sliding window, because the computation is always carried out on the finest grids. After selecting a sliding window by defining a bounding box with half the size in each direction (Figure 6(b)), 400 cells are transferred for visualisation. Hence, the visualisation resolution in the physical domain got higher while reducing the size of the window for display.

In Figure 6(c) the size of the sliding window was even reduced further to 25% of the original size in each direction thus giving even more insight by having a higher physical resolution in this area.

Figure 7 shows the application of the sliding window concept for the same benchmark problem, now computed with a different Reynolds number $Re = 3200$. Here, one can clearly see the gain in visual details while zooming into the top left corner where a vortex has formed. While the vortex was not visible when displaying the entire domain on a coarse scale (even it was already there), it becomes more and more clear for smaller window sizes.

An additional benefit of this approach – which arose nearly for free out of the implementation of the sliding window – is shown in Figure 8. Here the sizes of the sliding window itself is not changed, but a different threshold for the cell selection is chosen. This means, that if a maximum of 100 cells should be displayed due to bandwidth limitations, the results look like Figure 8(a). By increasing the threshold continuously, more and more details get visible, until the computational resolution is reached, which is shown in Figure 8(d).

This easy benchmark simulation now highlights the way and the possibilities the sliding window can be applied to: assume a very large domain containing the power plant depicted in Figure 9. If an adaptive high performance multi-scale computation should be carried out on hundreds of thousands of processors, it will not be possible to transfer all the data in a reasonable amount of time for visualisation purposes to a dedicated front end node. If the visualisation should be performed in a reasonable amount of time or even in real time, only some data points of the grids will be sent to the visualisation node, thus showing a coarse picture of the complete domain. Assuming the time for sending these visualisation results is sufficient, one then can gradually zoom into the domain while transferring more and more accurate data and omitting less and less points until such a small part of the domain is reached, that every data point can be transmitted and the visual resolution meets exactly the computational one.

This is presented in Figure 10. Here, the same visual resolution for displaying a (geometric) detail is used as in the case where the complete power plant model was shown.

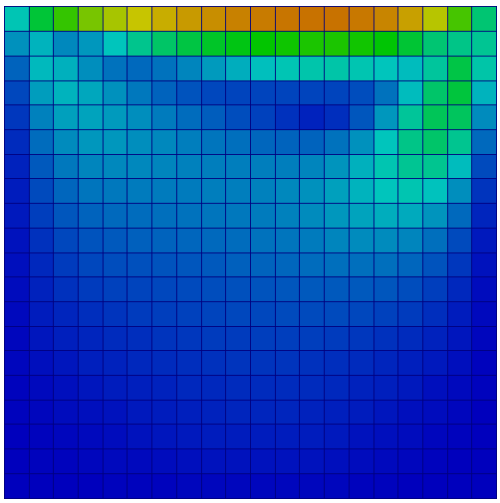
5 Conclusion

In this paper, we have presented a sliding window concept for interactive high-performance computing that allows a user to select and slide/resize a region of interest within

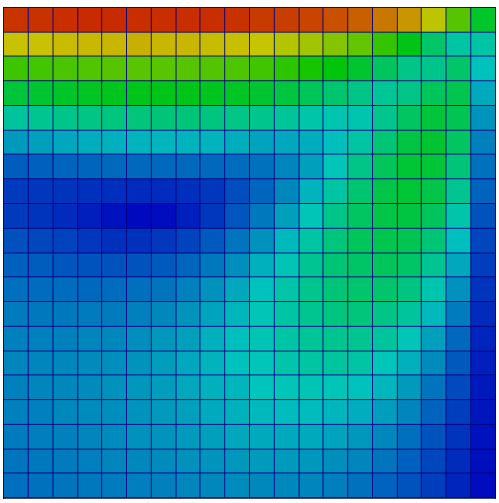
the computational domain for the visual display of simulation results in order to restrict and keep constant the data transfer between a simulation back end and visualisation front end. This becomes necessary as any complex simulation with several hundreds of thousands of data points easily exceeds the bandwidth of modern networks and, thus, hinders an interactive processing of the running simulation as required for computational steering. We have further shown first very promising results of this concept for a fluid flow solver, where a user out of ParaView can control and interact with a parallel CFD code, giving him the possibility to study large and small scale phenomenas as for instance to be observed in multi-scale simulations. Such large examples underline the need for sophisticated concepts like this in order to interactively process huge data sets and to leverage interactive high-performance computing.

References

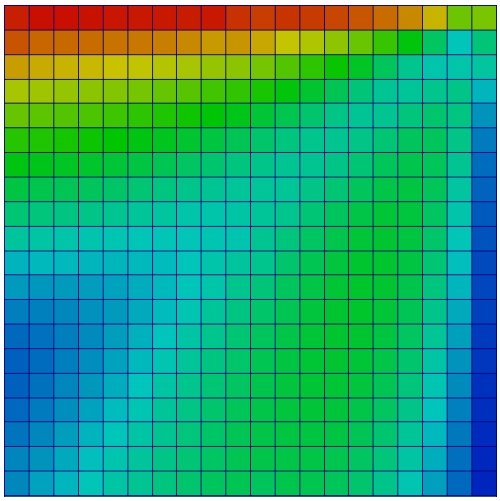
1. J. Mulder, J. van Wijk, R. van Liere, "A survey of computational steering environments", *Future Generation Computer Systems*, 13, 1998.
2. URL `http://www.netlib.org/cumulvs/`, CUMULVS: Computational Steering and Interactive Visualization in Distributed Applications.
3. URL `http://www.realitygrid.org/`, RealityGrid.
4. J. Vetter, K. Schwan, "High Performance Computational Steering of Physical Simulations", in *Proc. of the 11th Int. Symposium on Parallel Processing*. IEEE, 1999.
5. URL `http://www.scirun.org`, SCIRun: A Scientific Computing Problem Solving Environment.
6. URL `http://www.hlrs.de/organization/av/vis/covise/`, COVISE – COllaborative VIsualization and Simulation Environment.
7. J. Knežević, J. Frisch, R.P. Mundani, E. Rank, "Interactive computing framework for engineering applications", *Computer Science*, 7(5): 591–599, 2011.
8. J. Knežević, R.P. Mundani, E. Rank, "Interactive computing – virtual planning of hip-joint surgeries with real-time structure simulations", *Modeling and Optimization*, 1(4): 308–313, 2012.
9. J. Frisch, R.P. Mundani, E. Rank, "Adaptive Data Structure Management for Grid Based Simulations in Engineering Applications", in *Proc. of the 8th International Conference on Scientific Computing (CSC)*. Las Vegas, Nevada, USA, July 18-21 2011.
10. MPI Forum, "MPI: A Message-Passing Interface Standard. Version 2.2", September 4th 2009, available at: http://www.mpi-forum.org, Dec. 2009.
11. J. Frisch, R.P. Mundani, E. Rank, "Resolving Neighbourhood Relations in a Parallel Fluid Dynamic Solver", in *Proc. of the 11th International Symposium on Parallel and Distributed Computing - ISPDC*, pages 267–273. Munich, Germany, June 25-29 2012, DOI 10.1109/ISPDC.2012.43.
12. J. Frisch, R.P. Mundani, E. Rank, "Communication Schemes of a Parallel Fluid Solver for Multi-Scale Environmental Simulations", in *Proc. of the 13th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC)*, pages 391–397. Timisoara, Romania, September 26-29 2011, DOI: 10.1109/SYNASC.2011.7.
13. H. Samet, "The Quadtree and Related Hierarchical Data Structures", *ACM Comput. Surv.*, 16: 187–260, June 1984, ISSN 0360-0300.
14. URL `http://www.paraview.org`, ParaView.
15. URL `http://www.vtk.org`, Visualisation Tool Kit VTK.
16. U. Ghia, K.N. Ghia, C.T. Shin, "High-Re solutions for incompressible flow using the Navier-Stokes equations and a multigrid method", *Journal of Computational Physics*, 48(3): 387 – 411, 1982, ISSN 0021-9991.



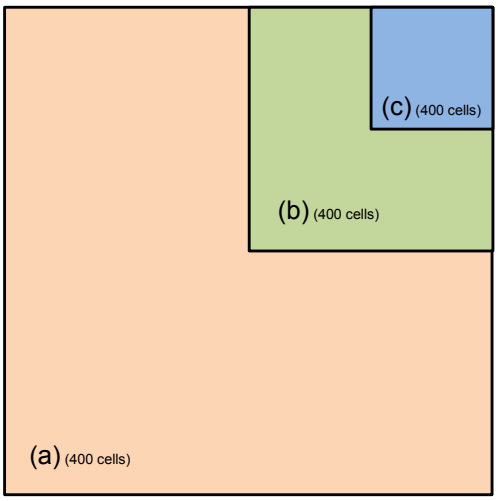
(a) Display of max. 400 cells in the sliding window bounding box (0;0) to (1;1)



(b) Display of max. 400 cells in the sliding window bounding box (0.5;0.5) to (1;1)

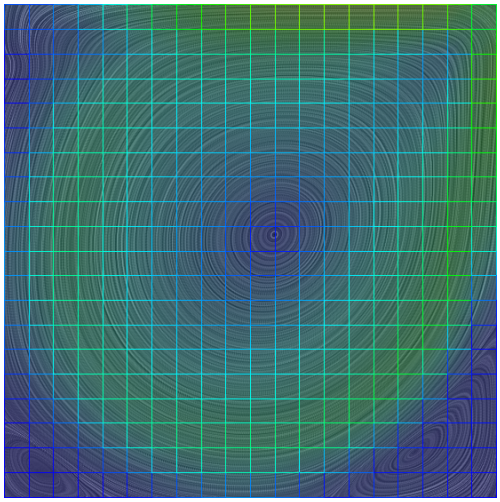


(c) Display of max. 400 cells in the sliding window bounding box (0.75;0.75) to (1;1)

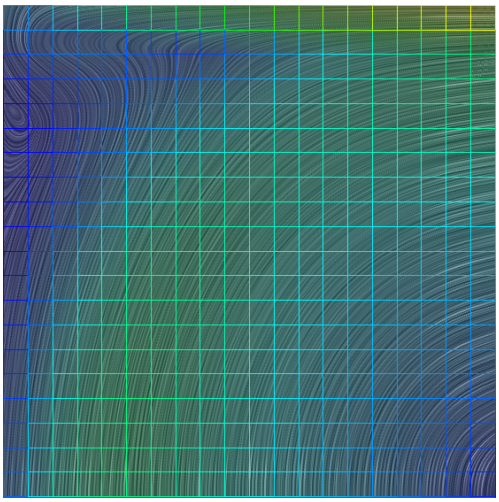


(d) Different positions of the selections from a, b, and c

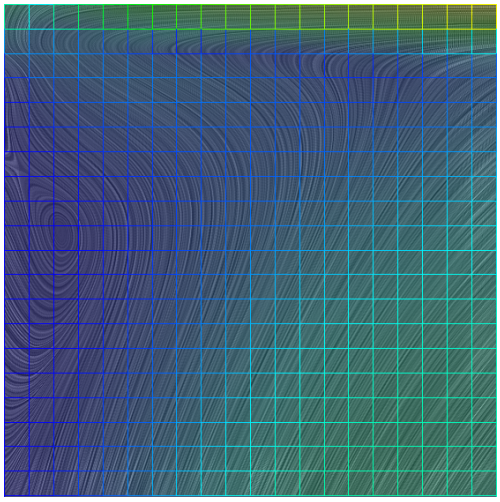
Fig. 6. Flow in a lid driven cavity for $Re = 100$, computed with the parallel implementation of our flow solver and visualised by the sliding window concept for different selections, always using a maximum of 400 cells (but always computed on the finest resolution of the computational domain).



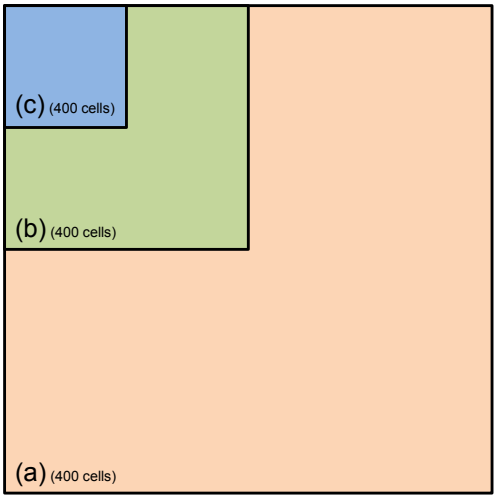
(a) Display of max. 400 cells in the sliding window bounding box (0;0) to (1;1)



(b) Display of max. 400 cells in the sliding window bounding box (0;0.5) to (0.5;1)

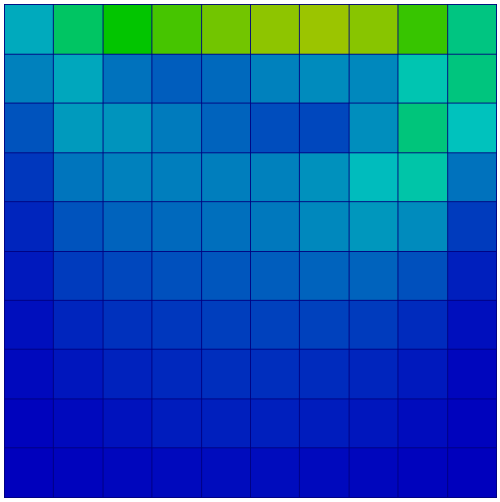


(c) Display of max. 400 cells in the sliding window bounding box (0;0.75) to (0.25;1)

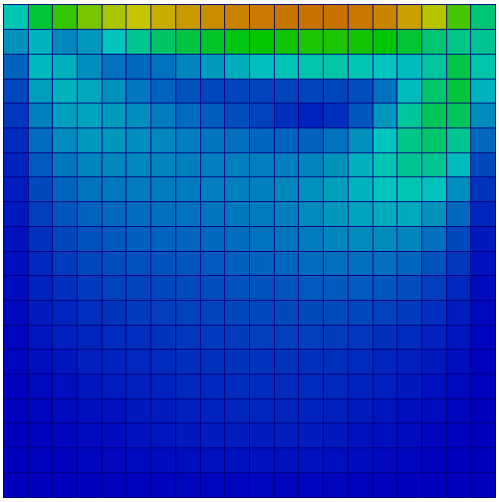


(d) Different positions of the selections from a, b, and c

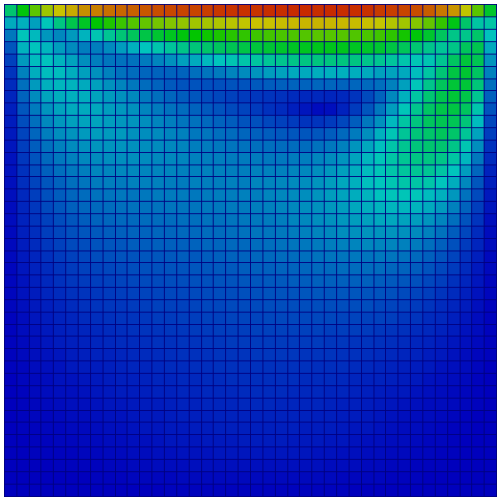
Fig. 7. Flow in a lid driven cavity for $Re = 3200$ (visualised using LIC – Line Integral Convolution), computed with the parallel implementation of our flow solver and visualised by the sliding window concept for different selections, always using a maximum of 400 cells (but always computed on the finest resolution of the computational domain). The vertex in the left upper corner cannot be seen in the coarsest visualisation, only after zooming in.



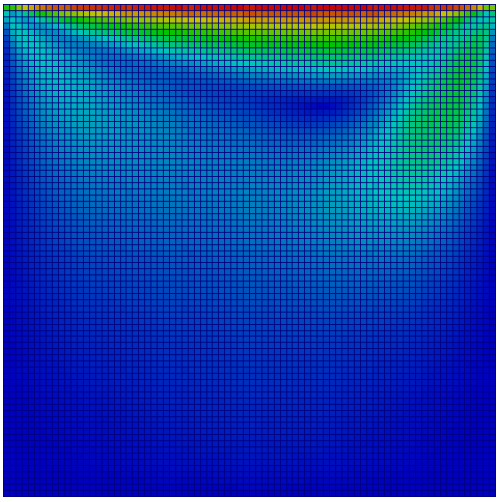
(a) Complete grid with max. 100 cells selected



(b) Complete grid with max. 400 cells selected



(c) Complete grid with max. 1600 cells selected



(d) Complete grid with max. 6400 cells selected

Fig. 8. Flow in a lid driven cavity for $Re = 100$, computed with the parallel implementation of our flow solver and visualised for different levels of detail (but always computed on the finest resolution of the computational domain).

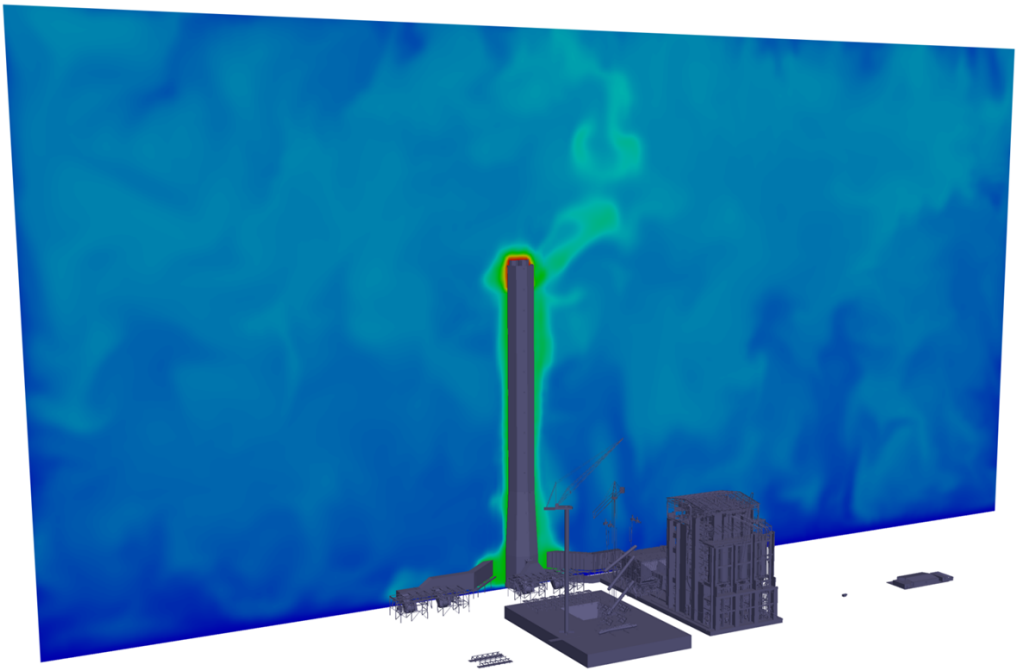


Fig. 9. Thermal simulation of a power plant on a very large scale.

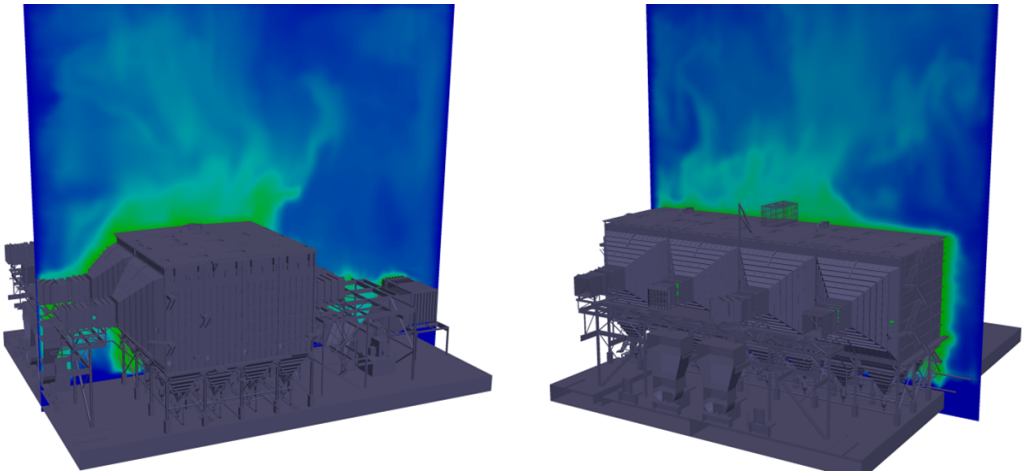


Fig. 10. Thermal simulation of a small part of the power plant on a small scale using as much degrees of freedoms as the full scale model.