Journal of Computer Science 7 (5): 591-599, 2011
ISSN 1549-3636


# Interactive Computing Framework for Engineering Applications

Jovana Knezevic, Jerome Frisch, Ralf-Peter Mundani, Ernst Rank
Department of Computation Engineering, Technische Universität München,
Arcisstraße 21, 80333 Munich, Germany

**Abstract: Problem statement:** Even though the computational steering state-of-the-art environments allow users to embed their simulation codes as a module for an interactive steering without the necessity for their own expertise in high-performance computing and visualisation, e.g., these environments are limited in their possible applications and mostly entail heavy code changes in order to integrate the existing code. **Approach:** In this study, we introduce an integration framework for engineering applications that supports distributed computations as well as visualization on-the-fly in order to reduce latency and enable a high degree of interactivity with only minor code alterations involved. Moreover, we tackle the problem of long communication delays in the case of huge data advent, which occur due to rigid coupling of simulation back-ends with visualization front-ends and handicap a user in exploring intuitively the relation of cause and effect. **Results:** The results for the first test cases are encouraging, both showing that we obtain excellent speedup in parallel scenarios and proving that the overhead introduced by the framework itself is negligible. **Conclusion/Recommendations:** Testing the case involving massively parallel simulation, as well as the integration of the framework into several parallel engineering applications are part of our imminent research.

**Key words:** Interactive computing, Computational Steering Environment (CSE), pure multithreading, integration framework, engineering applications, Problem Solving Environment (PSE), Message Passing Interface (MPI)

## INTRODUCTION

Interactive computing, in general, refers to the real-time interplay of a user with a program during the program runtime in order to estimate its actual state or tendency and to fetch an opportunity to react on variety of changes. Within numerical simulation experiments, specifically, this implies that the geometry of the simulated scene can be modified interactively altogether with boundary conditions or a distinct feature of the application, thus, the user can gain "insight concerning parameters, algorithmic behavior and optimization potentials" (Mulder *et al*., 1999). The commonly agreed central features of interactive computing in this case are: on the front end, a sophisticated user interface and the visualization of results on demand and, on the back-end, a separated steerable, often time- and memory-consuming simulation running on a high-performance cluster (Fig. 1).

Even though powerful tools such as Van Liere and Van Wijk (1996); McCorquodale *et al*. (2000) and Reality Grid (2003) allow users to embed their simulation codes as a module for an interactive steering without the necessity for their own expertise in algorithms and data structures, high-performance computing and visualization, these tools are limited in their possible applications and mostly entail heavy code changes in order to integrate the existing code.

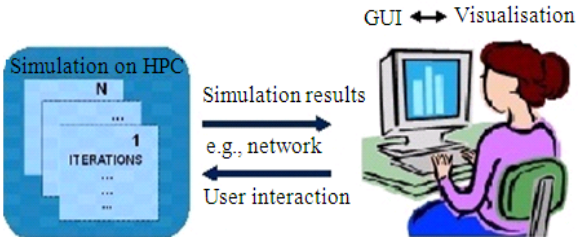



Fig. 1: At the development front-end, the user guides the simulation in building a solution to his problem via graphical user interface, while on the back-end, an often time-and memory-consuming program is being executed

**Corresponding Author:** Jovana Knežević, Lehrstuhl für Computation in Engineering, Technische Universität München, Arcisstraße 21, 80333 München, Germany

**The state of the art:** CSE is a computational steering environment whose kernel is designed to be very simple, flexible, minimalistic and all higher level functionality is pushed into the modular components, so-called satellites. It is based on the idea of the data manager informing all the satellites of changes made in the data and an interactive graphics editing tool allowing users to bind data variables to user interface elements.

CUMULVS is a middle layer between the application program and the visualization and steering front-end. It encompasses all the connection and data protocols needed to dynamically attach multiple visualization and steering front-ends to a running application. The user has to declare in the application which parameters are allowed to be modified or steered during the computation.

In the SCIRun Problem Solving Environment (PSE) for modeling, simulation and visualization of scientific problems, a user may, on the one hand, smoothly construct a network of required modules via a visual programming interface, while, on the other hand, changes with a deeper impact on the simulation require an automatic cancellation and restart of the simulation. In addition, this PSE has typically been adopted to support pure thread-based parallel simulations, consequently limiting the scale of the scientific computation that can be tackled to the shared memory environments.

Uintah is a component-based visual PSE that builds upon the best features of the SCIRun PSE, being, contrarily, designed to specifically address the problems of massively parallel computation on terascale computing platforms.

In the RealityGrid project, an application is structured into a client, a simulation and a visualization unit communicating via calls to the steering library functions. It involves insertion of check- and break-points at fixed places in the code where modified parameters are obtained and the simulation is to be restarted, respectively.

Within the Chair for Computation in Engineering at Technische Universität München, in the previous years, several successful Computational Steering research projects involving long-term cooperation with industry partners took place.

Valuable experience has been gained and advanced state reached in efforts to reduce the work required to extend an existing application code for steering. Performance investigations of several interactive applications, in regard to responsiveness to steering, have been done, as well as identifications of factors limiting performance. The focus at this time has been set to interactive Computational Fluid Dynamics (CFD), based on the Lattice-Boltzmann method, including Heating Ventilation Air-Conditioning (HVAC) system simulator (Borrmann *et al*., 2005), online-CFD simulation of turbulent indoor flow in CAD-generated virtual rooms (Wenisch *et al*., 2004), interactive thermal comfort assessment (Van Treeck *et al*., 2007). However, the developed concepts have been primarily adopted to this limited number of application scenarios, thus, they allow for further investigations so as to become more generic.

## MATERIALS AND METHODS

In the interest of widening the scope of the applications of the framework, our essential aim is an instant response of any simulation back-end to the changes made by the user. So as to achieve it, the regular course of the simulation coupled to our framework is being interrupted, using software equivalent of hardware interrupts, i.e., signals, in small, application-compatible, cyclic intervals, followed subsequently by a check for updates. If, meanwhile, there has been no user interaction, the control is given back to the simulation, which continues from the state saved at the previous interrupt-point, either until the results of the computation are complete and should be sent to the user, or until the end of yet another interrupt interval. Otherwise, the new data is received and simulation state variables are manipulated in order to make the computation stop and then start anew according to the updated settings (boundary conditions, simulation parameters).

As elaborated, to guarantee the correct execution of a program, one should use certain type qualifiers for the variables which are subjects to sudden change or objects to interrupts. First of all, one should ensure that certain types of objects which are being modified both in the signal handler and the main computation are updated in an non-interruptible way. Second of all, if the value in the signal handler is changed, one should take care that, due to compiler optimizations, the old value in the register is not used again instead of reading the updated value from the memory (which might result in undesired behavior of the program). Moreover, it is the responsibility of a user himself to instruct the simulation program how the received data should be matched to the simulation-specific requisites so as to reflect properly the outcome of the modifications.

Referring to the aforementioned idea, a relevant remark is that, under any circumstances, when the control of execution is given back to the main computation, it is obliged to continue at the point where

it has previously been interrupted. However, taking the pseudo code of an iterative solver for a system of linear equations (Fig. 2) as an example, this unconditionally happens only until the end of the current, most-inner loop iteration, where the earliest opportunity is used to compare the values of the simulation state variables and, if result of the comparison indicates so, consequently exit all the loops (i.e., starting with most-inner one and finishing with the most-outer one). This exactly means starting computation over again. Finally, with either one or several number of iterations being finished without an interrupt, new results are handed on to the user process for visualization. One more time it is user's responsibility to prescribe to the front-end process how to interpret the received data so that it can be coherently visualized.

Due to the complexity requirements and amount of data in numerical simulations nowadays, in order to fully exploit the general availability and increasing CPU power of high-performance computers, sophisticated parallel programming methods are inclined. The design of our framework, therefore, takes into consideration and supports different parallel paradigms, which results in an extra effort to ensure correct program execution and avoid synchronization problems when using threads, as explained further in the text.

**Multithreading parallelization scenario:** In the case of pure multithreading (with OpenMP/POSIX threads, e.g.,) used for the computations on the simulation side, the idea is that as soon as a random thread is interrupted at the expiration of the user-specified interval, it checks, via the functionality of the Message Passing Interface (MPI), if any information regarding the user activity is available. If the aforesaid probing of the user's message indicates that any change has been made, the receiving thread instantly obtains information about it. Furthermore, all the other threads become aware that their computations should be started over again and proceed in the way in which clean termination of the parallel region is guaranteed, as described in more detail.

**"Hybrid" parallelization scenario:** In the case of hybrid parallelization of a simulation (i.e., MPI and OpenMP), a random thread in each active process is being interrupted, hence, fetches an opportunity to check for the updates. The difference in comparison to the exclusively multithreaded parallelization is that now all the processes have to be explicitly notified about the changes performed by a user, which, matched up to pure multithreading, also involves additional communication overheads. If one master process, which is the direct interface of the user's process to the computing-nodes, i.e., slaves, is informing all of them about the user interaction, this may result in the master process becoming a bottleneck.

Therefore, a hierarchical non-blocking broadcast algorithm for transferring the signal to all computing nodes has been implemented (Fig. 3).

What is more, in efforts to interrupt one thread per process, an inevitable trade-off between ensuring a minimal number of checks per process and allowing for receiving the data promptly has to be faced, thus, as a next step, an optimal interval between the interrupts on different levels of the communication hierarchy is going to be estimated. In addition, a possibility of distributing the tasks among several user processes, each in charge of a certain group of simulation processes will be examined.

**Test case:** To evaluate our concepts, we have coupled our framework, on one side, to a C++ 2D simulation of heat conduction (described by Laplace heat equation) in a given region over time. Solutions of the heat equation are characterized by a gradual smoothing of the initial temperature distribution by the heat flow from warmer to colder areas of a domain.

This means that different states and starting conditions will tend toward the same stable equilibrium. After discretising using a Finite Difference scheme for updating the values, we come up with a five-point stencil. The system of linear equations is then solved using the Gauss-Seidel iterative method.

On the other side, we have coupled our framework to a graphical user interface using the wxWidgets library. The temperature is represented along the z-axes, pointing upward, hence, showing the variations of the temperature in the corresponding 2D domain. The simulation and the visualization are implemented as separate MPI processes.

```
Iterative_solver() {
    for (t ← T0 to TN) do      # iterations over time
    # iterations over a domain
     for (idx1 ← X10 to X1N )
       for (idx2 ← X20 to X2N)
           Process(data[idx1][ idx2])
}
```

Fig. 2: Pseudo code: An example of an iterative solver

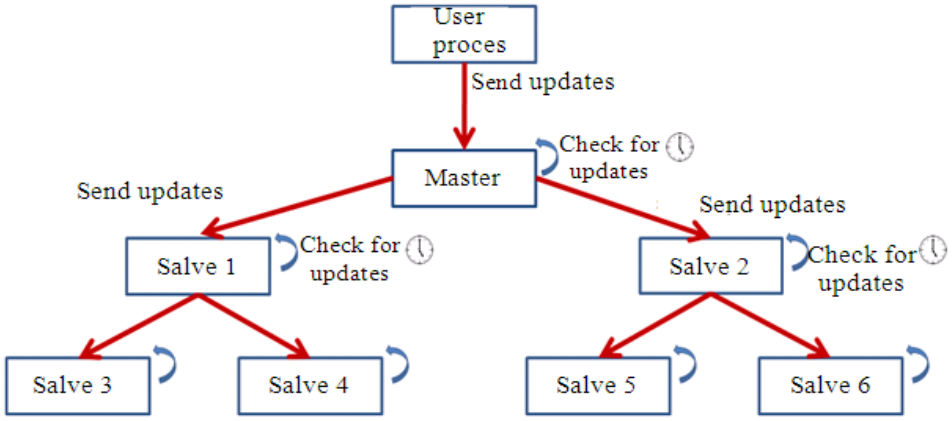



Fig. 3: User process sends the data about the update to solely master process on the simulation side; master process checks for the updates in small fixed intervals and signal is transferred from master to all the slaves via communication hierarchy. All the slaves then do their own checks in their own fixed intervals

**Initial settings:** The very first settings, which include grid generation, error tolerance and maximal number of iterations for the simulation, are specified by a user via graphical user interface (Fig. 4a). Immediately after defining these parameters, one can define the boundaries of the domain and set points with certain fixed values of the temperature, so-called pillars (Fig. 4b).

**User interaction:** When it comes to interplay with the program during the simulation, there are a few possibilities available-one can interactively add, delete or move pillars, add, delete, or move boundary points, change maximal possible number of iterations and error tolerance.

However, every time the change is performed by a user and the simulation becomes aware of it, the computation is restarted. Thus, what is unfortunately not feasible on a 300×300 grid, due to the short intervals between two restarts in the case of "hand over fist" user interaction, is an instant estimation of the equilibrium state for points of the domain far away from pillars, as shown in Fig. 5b. In this case, we profit from the hierarchical approach, introduced already in the next paragraph.

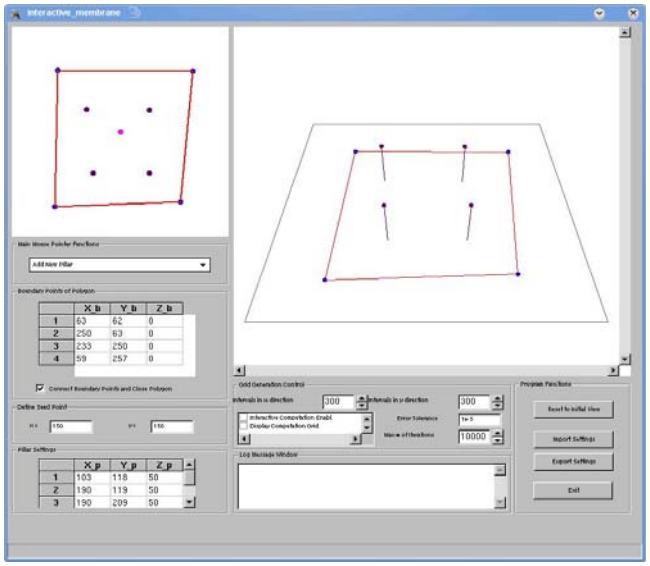
**Hierarchical approach:** Our hierarchical approach is based on the usage of several different grids depending on the frequency of the user interaction. The principle on the example of the test case is as follows: At the beginning, the initial grid is used for the computation. In the case that some user interaction occurs, the simulation process recognises it and, as soon as it restarts the computation with the updated settings, the coarser grids are used, the level of coarseness being determined based on the frequency of user's activity.

In this particular test case, we are using three different grids. Besides the initial 300×300 grid (Fig. 6a), the four times smaller, intermediate one (Fig. 6b), is used in the case of lower pace of interactions-adding/deleting pillars or boundary points, e.g. Finally, the coarsest, 75×75 (Fig. 6c) is brought into play for the occasion of the very high frequency of moving boundary points or pillars over the domain.

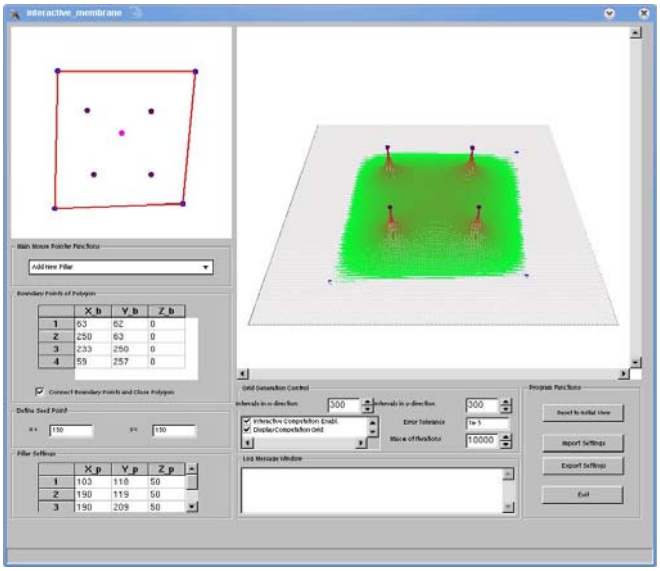
Although at this point, one does not have completely accurate results, the tendency of the running simulation in the overall domain can be easily and instantly observed, independently on the number and rate of changes applied. At last, when the current settings satisfy user's requirements, no more interaction is involved and the stage of calculating more accurate results is reached again.

Namely, as soon as the simulation at the back-end realizes that there has been no front-end intervention for a user-predefined time slot, it stops, switches back to one of the finer grids, depending on the slot and starts a new computation. In this case, the results of the previous computing on the coarser grid are discarded. To speed up reaching the heat equilibrium on finer grids, a multi-level method, where the previous precious results are reused, is exploited, as commented on in more detail further in the article.

**Multi-level approach:** In order to avoid wasting the computational cycles within the runtime while the user is interacting, we employ a multi-level algorithm, i.e., the results of the computation on the coarsest grid are not disposed of when switching to the finer one.




(a)



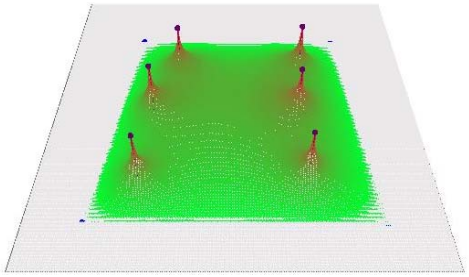

(b)

Fig. 4: (a) Initial settings; (b) simulation running with initial border and pillar settings on a grid 300×300, with error tolerance e-05 and maximal number of iterations set to 10,000

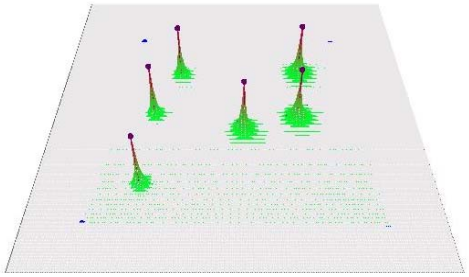
Namely, our concept already involves a hierarchy of discretisations as in multigrid algorithm and we profit from the analogous idea. Nevertheless, instead of accelerating the convergence of a basic iterative method by global correction from time to time, accomplished by solving a coarser problem, i.e. descending to the coarser grids and calculating an error, as in multigrid algorithm, our scheme starts with the solution on the coarsest grid and only uses the result we gain as an initial guess of a result on a finer one (Fig. 7).

As expected, the results show clearly that the speed of convergence is significantly higher with the new approach.

For the examples of the settings we tested including an initial 300×300 grid and several different pillar $P_i$ ($x_i$, $y_i$) and boundary $B_j$ ($x_j$, $y_j$) points with corresponding ordered pairs ($x_k$, $y_k$) of x- and y-axes indices respectively and the error tolerance set to e-05, the number of iterations needed for convergence both on the intermediate and the initial grid can be significantly improved.



(a)



(b)

Fig. 5: (a) An initial scenario; (b) moving pillars/boundaries rapidly leads to the continual restart of the computation and inability to estimate the equilibrium temperature in the region farther away from the pillars, i.e. reached in later iterations

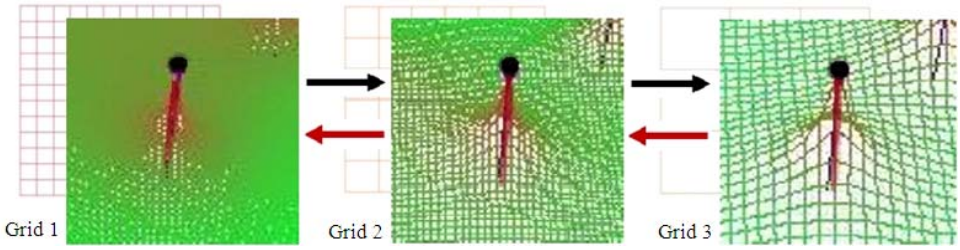



Fig. 6: Switching from the finest grid (grid 1: 300×300) by adding/deleting pillars/boundaries to the intermediate (grid 2: 150×150) and, finally, when moving pillars/boundaries, to the coarsest one (grid 3: 75×75) and reversely to the initial grid when there is an interval without any interaction

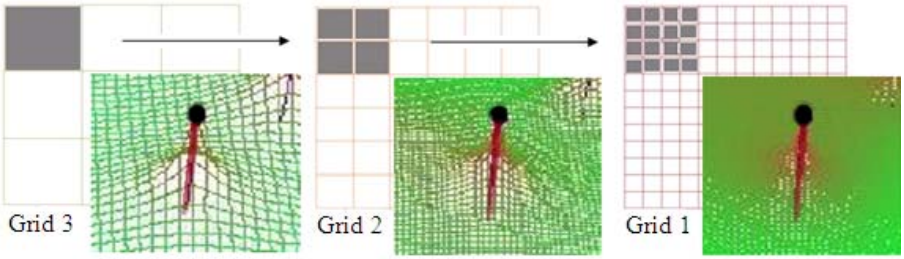



Fig. 7: Copying the results (temperature values) from the computation on the coarser grid to the initial vectors of the finer one.

**Code modification requirements:** To integrate the framework into any application scenario, a few modifications of the code have to be made by the user. Since these modifications are only minor, we list all of them.

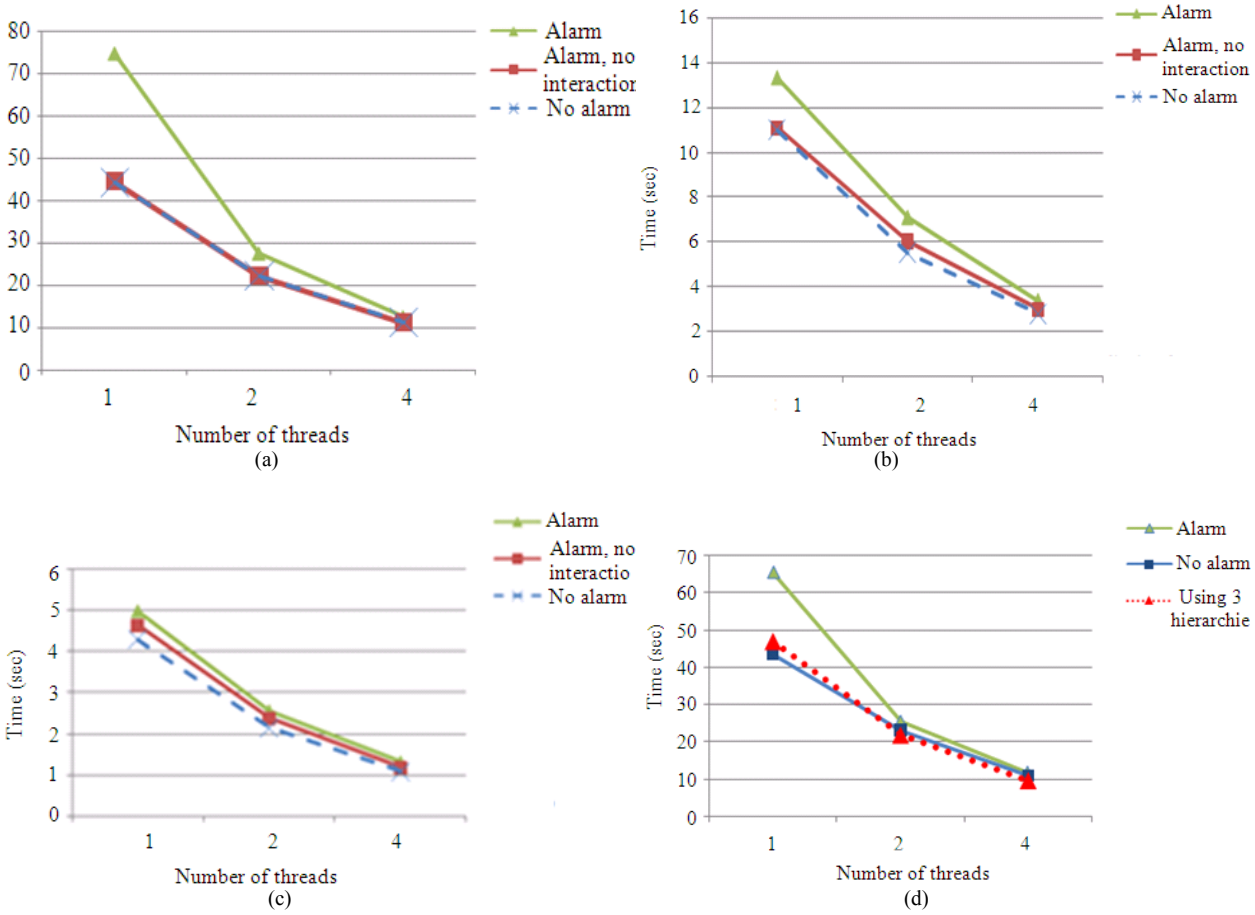



Fig. 8a-c: Non-hierarchical approach: estimations of the overall execution time in seconds (vertical axis) for 1, 2 and 4 threads (horizontal axis) doing the computations on grid 1000×1000, 500×500 and 300×300 respectively; square and cross markers represent respectively the values without and with checks for updates on simulation side, without any user interference actually occurring and almost completely overlap each other; triangle markers show the values with user interaction rate of 5 milliseconds. d) Hierarchical approach: measurements of the overall time in seconds (vertical axis) for 1, 2 and 4 threads (horizontal axis) in the case of series of user interaction occurring every 5 milliseconds, with 0.5 seconds break in between series; triangular and square markers connected with solid line show overheads of the time in the case of user interaction in comparison to the pure execution time, similarly as in a), while the markers connected with dotted line show how these overheads are minimized using hierarchical approach, compromising the accuracy

First of all, all the variables which will be affected by the interrupt handler in order to force the restart of the computation have to be declared both global (Fig. 8) and atomicity of their updates and prevention of the compiler optimizations which would lead to incorrect value references insured. Second of all, the integrity of each user-defined ‘atomic’ sequence of instructions in the simulation code has to be ensured. Furthermore, the calls to the provided send and receive functions which would be interface to our framework have to be included in the appropriate places in the code. Nevertheless, the user himself should instruct the interpretation of the data (in the receive buffers on both simulation and visualisation side, e.g.). Finally, he has to enable the regular checks for updates by including appropriate functions which will examine and change the default signal (interrupt) action, specifying in the same time the time interval in which the checks of the simulation process(es) should be made, as shown in the pseudo code example.

## RESULTS

So as to evaluate potential overheads caused by integration of our framework, we have done measurements, the average of several being graphically represented in Fig. 9.

First of all, concerning the non-hierarchical approach, what has been estimated for the cases of single-, two-and four-threaded simulation and the different initial problem sizes: 1000×1000 (Fig. 8a), 500×500 (Fig. 8b) and 300×300 (Fig. 8c), was the overhead caused only by cyclic interruption of the simulation every millisecond, realising that there is no update available, since on the front-end the user is absolutely not interfering. What is easily observable concerning the total execution time of the simulation for all the three aforementioned scenarios is that this kind of overhead, caused, as a matter of fact, only by raising interrupts which do a message probing, can be neglected.

In the same three graphs of Fig. 9, the runtime estimations which have been illustrated make it easy to compare the total execution time of the simulation with or without updates sent from the front-end. The measurements have been made for the case of user interaction occurring repeatedly, every 5 milliseconds. The conclusion is that in the case of two or four threads for the specified problem sizes one may only observe very small overheads, while in the case of a single thread being interrupted and restarting its computation with aforementioned high and one must point out very unlikely, frequency of user interaction, more significant overheads may be introduced.

```
# Function to override the default interrupt action
My_interrupt_action() {
    If(update_available) {
        Receive_update()
        Manipulate_state_variables()
    }
}

# declare here state variables global and volatile
main( ) {
    Set_interrupt_action (My_interrupt_action);
    Set_interrupt_timer (INTERVAL) # initiate 1st interrupt
    Iterative_solver ()  # interrupt can occur here at any point
}
```

Fig. 9: Pseudo code which exemplifies for an iterative solver the code modifications necessary to integrate the framework into any application

These results are as expected, concerning the fact that our tests show that the number of interrupts where interaction is recognized in the case of a single thread is almost three times bigger (ca. 2500) than in the case of two and almost 5 times than in the case of 4 threads. Thus, the amount of computational cycles we discard in this scenario is larger.

At the point at which we have introduced the hierarchical approach based on switching between three different grids, so as to compare the two approaches, we have decided to measure the execution time in the case of non-periodical user interaction, being performed as a series of 5 millisecond frequent changes, with half of a second long intervals in between. The overhead caused in non-hierarchical scenario for 1000×1000 grid and one thread, which is most challenging case, for this new predefined occurrence of user interference is proven to be similar as in Fig. 8a. Nevertheless, the hierarchical approach outcome show that the aforementioned overhead in the example with one thread is drastically reduced in comparison to the non-hierarchical. In other words, the time of executing the same number of iterations, but now making use of three different grids is very close to the execution time without any interrupts, although in some number of iterations, while the interaction is very frequent, the user would have to agree with lower accuracy consequences. For two or four threads, this overhead turns out to be even smaller (Fig. 8d).

## DISCUSSION

Referring to what has already been said, it is important to point out that the hierarchical approach we have in mind for the future test cases is not limited to recursive coarsening the grid. On the contrary, one can analogously utilize other simulation-specific hierarchies (different polynomial degrees of basis functions in Finite Element scheme approximation, e.g.,) and any user of the framework can, if needed, easily adopt it to his individual requirements. One of our most imminent intentions is the existing p-FEM code used for computational orthopedics.

## CONCLUSION

In this study, we have presented a generic platform which couples simulation codes and visualization tools in the way which allows a user to trigger a simulation during the runtime, based on a ‘minimal invasion’ principle, i.e. minor code changes necessary and receive prompt feedback. Although the results for the first test cases look very promising the question of the signal

transfer from a user to all the computing nodes in the case of massively parallel simulation is a part of the current research, as well as the integration and testing of the framework incorporated into several parallel engineering simulation scenarios.

## ACKNOWLEDGMENT

This study has been financially supported by Munich Centre of Advanced Computing (MAC) and the International Graduate School of Science and Engineering (IGSSE) at Technische Universität München.

## REFERENCES

Borrmann, A., P. Wenisch, C. Van Treeck and E. Rank, 2005. Collaborative HVAC design using interactive fluid simulations: A geometry-focused platform. Technische Universität München. http://www.inf.bv.tum.de/papers/uploads/paper_479.pdf

Davison de St. Germain, J., J. McCorquodale, S.G. Parker and C.R. Johnson, 2000. Uintah: A massivelly parallel problem solving environment. Proceedings of the Ninth International Symposium on High-Performance Distributed Computing, IEEE Computer Society Washington, DC., USA., pp: 33-41. http://portal.acm.org/citation.cfm?id=822085.823309

McCorquodale, J., S.G. Parker, C.R. Johnson, 2000. Davison de St. Germain, J., Uintah: A massivelly parallel problem solving environment. The Ninth International Symposium on High-Performance Distributed Computing, 2000, Proceedings, pp: 33-41.

Mulder, J.D., J.J. Van Wijk and R. Van Liere, 1999. A survey of computational steering environments. Future Generat. Comput. Syst., 15: 119-129. DOI: 10.1016/S0167-739X(98)00047-8

Reality Grid, 2003. RealityGrid: moving the bottleneck out of the hardware and back into the human mind. UCL. http://www.realitygrid.org/index.shtml (Online)

Van Liere, R. and J.J. Van Wijk, 1996. CSE: A Modular Architecture for Computational Steering, Virtual Environments and Scientific Visualization. Springer Verlag, Vienna, pp: 257-266.

Van Liere, R. and J.J. Van Wijk, 1996. CSE: A Modular Architecture for Computational Steering. Proceedings of the 7th Eurographics Workshop on Visualization in Scientific Computing, (EWVSC'96), Springer Verlag, New York, pp: 257-266. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.7.5698

Van Treeck, C., P. Wenisch, A. Borrmann, M. Pfaffinger and M. Egger *et al*., 2007. Utilizing high performance supercomputing facilities for interactive thermal comfort assessment. Proceedings of the 10th International IBPSA Conference Building Simulation, Bejing, China.