Interactive Computing–Virtual Planning of Hip Joint Surgeries with Real-Time Structure Simulations

Jovana Knežević, Ralf-Peter Mundani, and Ernst Rank

***Abstract*—Current interactive computing state-of-the-art environments, libraries, and frameworks open the door for engineers to run their simulation codes in an interactive mode, i. e. allowing for estimations of the state and tendency, as well as modifications, during a simulation program runtime without the necessity for their own expertise in efficient algorithms and data structures, high-performance computing, and visualisation. Nevertheless, when it comes to the real-time response of the simulation to this interaction – namely, keeping the connection between the user's change and its effect intuitive or at least observable – these environments are still limited in their possible application and, furthermore, often entail heavy code changes in order to be coupled to existing codes. Therefore, we introduce an integration framework applicable to different engineering applications, which with only minor code modifications involved supports distributed simulations as well as visualisation on-the-fly and enables real time interactive computational steering. Furthermore, we present its integration into a previously existing pre-operative planning environment for joint replacement surgery, which makes possible an interactive patient-specific selection of the optimal implant design, size, and position. The environment is supposed to enable the real-time surgeon's interplay with virtual models of bones and implants in 3D, thus, simultaneous computation and visualisation of the load transfer between the bone and the implant. Moreover, we tackle the problem of long communication delays which occur in the case of rigid coupling of simulation back-ends with visualisation front-ends and handicap a surgeon in observing which of his modifications leads to which outcome.**

***Index Terms*—Bone Mechanics, computational steering environment (cse), human femur, interactive computing, message passing interface (mpi).**

I. Introduction

In general, interactive computing is the practice of the real-time intervening of a user with a program during the program runtime in order to estimate or influence its course and final outcome. It is often associated with numerical simulation experiments, especially where the pre-processing phase is time consuming and, thus, the opportunity to modify interactively either the geometry of the simulated scene, or boundary conditions, or individual parameters represents an indispensible feature. On the front end, a graphical user interface and the visualisation of results on demand are desirable, while on the back-end, an interruptible, often time- and memory-consuming simulation is running on a high-performance cluster (Fig. 1).

Nowadays, many tools which "provide an environment in which researchers themselves can build interfaces and visualisations to the simulation" [1] are available, however, mostly having limited scope of application and/or requiring significant code invasion during the integration phase.

Magellan assumes the export of monitoring and steering objects from an application. Afterwards, for instance, a collection of instrumentation points, such as so-called *actuators*, knows how to change an object without disrupting application execution. Pending update requests are stored in a shared buffer until an application thread polls for them [9].

In *EPSN* API [10], XML description of simulation scripts is introduced to handle data and concurrency at instrumentation points. Here a steering server, when receiving requests, determines their date, thus, the request is executed after the first date that fulfils a condition. Reacting on a request consists of releasing the predetermined blocking points.

Steereo [11] is a light-weight steering framework, not the complete steering environment, where the client sends requests and the simulation side will execute them and send some response. However, the requests are not processed immediately in the simulation, but rather stored in a queue and executed at predefined points in the simulation. A user has to determine when this queue should be processed in his code.

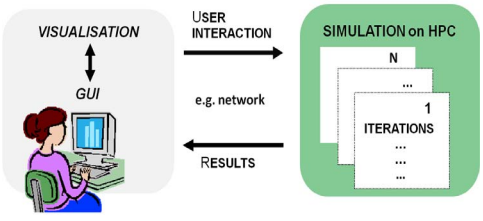

Fig. 1. Framework layout – user interaction with the running simulation: On the front-end a user is performing changes via graphical user interface, the information is sent to the simulation via the network.

G-HLAM [12], on the other hand, focuses more on fault tolerance, i. e. monitoring and migration of the distributed federates. The group of main *G-HLAM* services consists of a *Broker Service* which coordinates management of the simulation, a *Performance Decision Service* which decides when performance of a federate is not satisfactory, thus, migration is required, and a *Registry Service* which stores information about the location of local services. It has been tested on the application supporting surgeons with

Manuscript received October 5, 2011; revised October 31, 2011. This work was supported by the Munich Centre of Advanced Computing (MAC) and International Graduate School of Science and Engineering (IGSSE) at Technische Universität München (TUM).

The authors are with the Chair for Computation in Engineering, Technische Universität München, Arcisstraße 21, 80333 Munich, Germany (e-mail: knezevic@bv.tum.de), (e-mail: mundani@tum.de), (e-mail: rank@bv.tum.de)

simulations of vascular reconstruction, using distributed federations on the Grid for the communication among simulation and visualisation components.

Further comparison of different frameworks, computational steering environments, libraries, and tools is given in [2].

In the previous years, within the Chair for Computation in Engineering and with our cooperation partners at the Chair for Computer Graphics and Visualisation at Technische Universität München, an environment for pre-operative implant planning for hip joint replacement has been developed [6]. The ultimate goal of this medical procedure is to keep the stress distribution after insertion of an implant as close as possible to the physiological state, since removal of stress from certain regions in the bone due to the insertion of an implant might cause osteoporosis, degeneration of bone tissue, and lead soon unavoidably towards a new surgical intervention.

The developed analysis tool allows for implant selection and positioning based on prediction of response of patient-specific bone to a load that is applied. For this, two indispensable components have been coupled.

One is a simulation engine based on the models of femur, i. e. thigh bone geometry constructed by CT/MRI-data and using the Finite Cell Method (FCM), a variant of the high order *p*-FEM code with fictitious domain approach, as proposed in [3]. With this method, models with complicated geometries or multiple material interfaces can be easily handled without an explicit 3D mesh generation. The basic idea is an extension of the weak form of the partial differential equation beyond the physical domain up to the boundary of an embedding domain, which can easier be meshed.

The other is a sophisticated visualisation platform that allows the intuitive exploration of the bone geometry and particularly the mechanical response to various load situations of the physiological state and the post-operative state of an implant-bone situation in terms of stresses and strains [4, 5]. For this purpose, after sending an update of the settings – either after insertion/moving an implant, or testing a new position/magnitude of the forces applied to the bone – for each element and corresponding tensor a scalar value, i. e. the so-called von Mises stress norm, can be calculated and visualised as shown in Fig. 2.

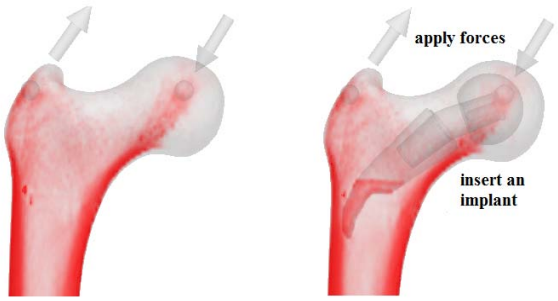

Fig. 2. Von Mises stresses (calculated for a polynomial degree $p = 6$) of a healthy bone (left) and after a virtual surgery (right) under load of 1500 N and 1125 N exerted at the femur's head and the great trochanter, resp.; darker colours refer to regions with higher stress magnitude, thus, providing an overview of how the implant changes the stress distribution in the surrounding bone tissue.

The challenges in developing such a two-component analysis tool are described in more detail in [6], [4], [5]. Unfortunately, due to the rigid communication pattern between the components, the new setting could be considered by the simulation only after the result for the previous one has been calculated and sent to the user. Therefore, the higher polynomial degrees were used, the longer became the total time for computing the outdated result plus the new ones until one could finally perceive the effect of his last change.

Hence, the central topic of this paper is the way in which these two components are glued in a new approach via our framework in order to allow for instant feedback about the changes performed by a surgeon.

II. General Idea of the Framework

In order to achieve an immediate response of any simulation back-end to changes made by the user, the regular course of the simulation coupled to our framework is being interrupted, using software equivalents of hardware interrupts, i. e. signals, in small, user-defined cyclic intervals followed by a check for updates [2].

If there has been any change on the user side, the new data is received and simulation state variables are manipulated in order to make the computation stop and then restart from an adequate point, according to the updated settings (new geometry, boundary conditions, etc.). It is the responsibility of a user to instruct the simulation program how the received data should be matched to the simulation data.

After the check for updates has been done, independently from whether any has been received, the control is given back to the simulation which continues from the state saved at the previous interrupt-point. However, this unconditionally happens only until the values of the simulation state variables can be compared earliest. Consequently, if the result of the comparison indicates so, the upcoming computation steps are skipped, meaning automatically re-starting the computation with new settings again.

As elaborated in [7], a significant remark is that, 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. Namely, ensuring atomicity of certain operations on the data is crucial for deterministic behaviour of the program. In addition to this, insuring memory consistency is necessary, not only in the sense of accessing always the correct values of variables in the main memory instead of potentially outdated values in the cache due to certain compiler optimisations, but also in the sense of releasing all allocated memory which is not supposed to be accessed anymore, or which is even not possible to access as soon as the new computation starts.

With some intermediate (one iteration in case of an iterative solver, e. g.) or the complete computation (in case of a direct solver, e. g.) being finished without an interrupt, new results are handed on to the user process for visualisation. Nevertheless, due to the fact that the framework is intended to be integrated in various application scenarios, hence it cannot be predicted in which way the results should be interpreted in each of them, it is again user's responsibility to prescribe to the front-end process how to interpret the

received data so that it can be appropriately visualised.

As given in more detail in [2], since many applications are amenable to concurrent execution, they are programmed nowadays using either shared memory, message passing, or these combined in hybrid parallel algorithms, thus, the design of our framework takes into consideration and supports all of them. These results in an extra effort to ensure correct program execution and avoid synchronisation problems when using threads.

In 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 by a signal 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 a change has been made, both the receiving and the other threads instantly obtain information about it due to the manipulated state variables, all of them becoming 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 in [7].

The same is valid for the case of hybrid parallelisation of a simulation (i. e. MPI and OpenMP), where not only a random thread in each active MPI process is being interrupted by signals to check for the updates, but also all the processes have to be explicitly notified about the changes performed by a user, which involves additional communication overhead.

Nevertheless, to prevent one master process, the direct interface of the user's process to the computing-nodes, i. e. slaves, from becoming a bottleneck, a hierarchical non-blocking broadcast algorithm for transferring the signal to all computing nodes has been implemented (Fig. 3), where all the computing nodes have their own signals invoked for their own fixed intervals.

Although all the simulation processes have to invoke their own signals to do checks for updates, this, due to the very small intervals in-between the two checks, still does not cause intolerable delays, even despite the necessary synchronisation among the processes.

III. Test Case–the Bone

A. The Communication Pattern

The main functionality provided to aid the pre-operative planning consists of the insertion of different implants, changing their position, applying forces at different places and with different intensity. The result a surgeon receives in terms of stresses distribution is accordingly visualised.

So as to achieve receiving of any feedback in real-time with the FCM simulation running on standard consumer-class hardware, and this even for higher accuracy, *i. e.* polynomial degrees of basic functions used in FCM higher than 4, our framework with several valuable features has been utilized.

To profit from all the features of the framework and overcome the problem of long communication delays, the initial structure of the components to be coupled has been slightly adapted to our needs.

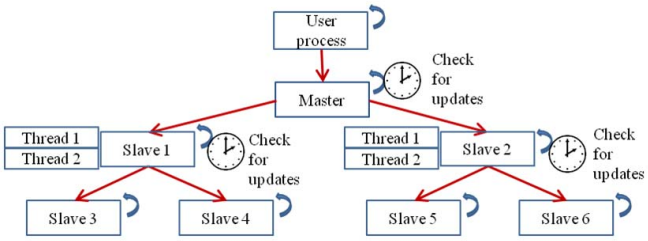

Fig. 3. Hierarchical communication pattern and transfer of the signal to all the processes. User process sends new settings to master of the simulation; master process checks for those updates in small, cyclic simulation specific intervals until an update is received, when it is transferred to all of the slaves in the communication hierarchy. Meanwhile, the slaves are doing their own checks in their own fixed intervals.

On the front-end, the main thread is in charge of fetching user interaction data and rendering. It is important to make sure that sending of the update information is done only, and also immediately, when the user is actually intervening via graphical user interface.

Consequently, in a second thread a loop for sending fetched updates is implemented. For sending of updates we use non-blocking MPI routines, giving the user an opportunity to provide for the computation the information about all of his modification requirements either immediately or in timely fashion, i. e. in specified intervals.

To our advantage, this thread is completely independent and, thus, not synchronised with the remaining third thread, which is dedicated for waiting to receive results as soon as these are available. In the first thread the rendering loop is continuously being executed, thus, this data becomes immediately visible to the user. A simplified communication pattern between the modules on the front-end and simulation on the back-end is illustrated in Fig. 4.

In order to benefit from this pattern, what becomes a challenge, is exploring the way in which the simulation running on the back-end can simultaneously become aware of the changes made on the front-end, i. e. when and how to interrupt the simulation kernel so that it can instantly receive an update and re-compute the solution for a new system of equations. To describe the challenge in more detail, we provide a basic overview of the simulation kernel structure.

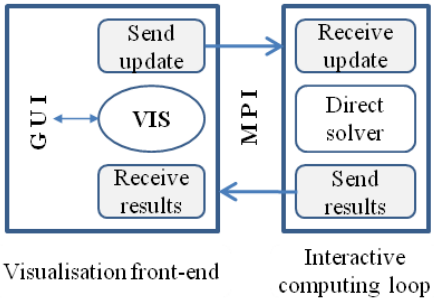

Fig. 4. Communication pattern between the two components. From the user front-end the changes of the data are recognised on a per-frame basis and sent immediately to the simulation via non-blocking routines. The simulation results, in terms of stresses, are then calculated and sent back to the user.

B. The Simulation Kernel

At the beginning, the femur voxel information generated on the visualisation side based on the quantitative computed tomography (QCT) scans and indicating the bone strength is

transmitted over the network, thus, the rectangular domain which embeds the entire femur is generated. The domain is then divided into sub cells of the same size. For the aforementioned FCM simulation, the polynomial degree of the shape functions p and the number of voxels in each direction are read from the user input file and are not dependent on the visualisation. The computational domain is kept fixed during the whole runtime of the simulation as the time-consuming discretisation is done only at the beginning, making the kernel convenient for interactive computing.

Driven by external forces f, a deformed solid is governed by the well-known equations from static elasticity theory, resulting in a linear equation system $K \cdot u = f$, where K is known as the stiffness matrix, u the displacement vector of all vertices, and f the force vector applied to the system. The stiffness matrix K is assembled from the element stiffness matrices, referring to individual elements lying inside the femur's physical domain.

Described initialisation and meshing steps are followed by an interactive computing loop, which consists of receiving user updates, pre-processing steps, solving the aforementioned system of equations, post-processing, and finally sending results to the user (Fig. 5).

Concerning the system of equations, due to its poor condition number sophisticated iterative solvers fail to be efficiently deployed and special treatment which allows for both the design of sophisticated solvers as well as for advanced parallelisation strategies is required. Therefore, a direct solver with hierarchical concepts, i.e. exploiting an octree data structure based on a nested dissection of the 3D domain (see Fig. 7) is used [13]. The main advantage here is that when inserting the implant, the stiffness matrices of the cells that experience change are updated locally and reassembly step is done only for a modified part of the system.

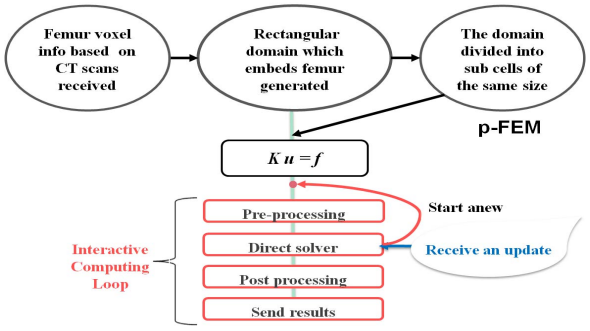

Fig. 5. Simulation process – execution flow. After receiving the data necessary for defining the problem and setting up the p-FEM kernel, a system of equations needs to be solved. The interactive computing loop consists of the pre-processing phase, an efficient direct solver based on nested dissection scheme where displacements of all the elements' vertices are calculated, the post processing phase, and finally returning computed stresses to the user's front-end for visualisation.

Despite the overall better performance of the solver in comparison to other direct solvers, i.e. Gauss and relatives, the current reassembling step which is computationally most expensive is undoubtedly worth being interrupted or skipped as soon as a surgeon on his interface changes actual settings.

C. Towards Interactive Computing – Interrupting the Simulation

As already implied, in order to further improve the proposed system towards an interactive simulation and visualisation environment, we have integrated functionality of our framework for instant interrupting the current computation in case of an update. On this occasion, due to the update, obtained stiffness matrices are supposed to be assembled, step by step traversing an octree bottom-up, into the global stiffness matrix. Afterwards, the solution for the system of equations at root, i.e. zero, level of the octree is done and all the solutions are recursively passed for each node to the nodes one level lower in the hierarchy for their own local solutions, as shown in Fig. 6. All the partial solutions are finally assembled into the final solution vector. The described algorithm, as presented in [13] has shown excellent scalability values in case of hybrid parallelisation [8].

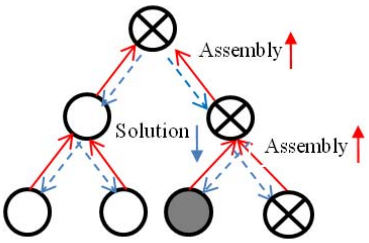

Fig. 6. Nested dissection solver. Dashed arrows indicate the solution sequence and the solid ones the assembly. For instance, in case of an interrupt being caught while processing the filled node at the bottom of the hierarchy, supposing the tree-like structure is being traversed in depth-first manner, the processing of the nodes marked with the cross is skipped.

Therefore, our intention is that the most time consuming phase, i.e. assembly, parallelised using shared or distributed memory concepts or both is being interrupted. Here, cyclically-repeating signals are used for frequent checks for updates. If there is an indicator of the upcoming message from the user side, this is recognised while processing one of the nodes in the previously mentioned hierarchical data structure and the simulation variables are set in a way which ensures skipping the rest of the nodes, as shown in Fig. 6. Thus, all the layers of the recursive assembly function call return immediately, the solution steps are skipped as well, and the new data is received at the beginning of the next step of the interactive computing loop.

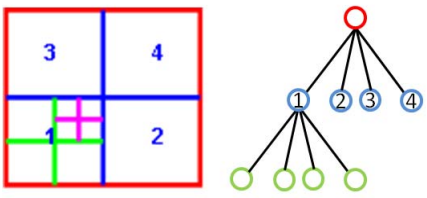

Fig. 7. Building a hierarchy of tasks based on nested dissection of the rectangular domain (for the sake of simplicity 2D case is shown). Dissection is done recursively and all the element stiffness matrices and load vectors are placed in the leaves of the tree structure.

In addition to the guaranteed data values consistency necessary for the correct program execution, mentioned at the beginning of the Section 2, sufficient steps to prevent

potentially introduced severe memory leaks before the new computation is started have to be taken. This is due to the interrupts and their possible occurrence before the memory allocated in the solver has been released. If assembly of different parts of the ochre is being processed by separate threads, i. e. the solver code is parallelized via Open MP, unexceptionally in this test case, it is ensured that when a new update is recognized by the thread catching a signal, all the other threads become immediately and automatically aware that they are supposed to skip the rest of their computations. As soon as the assembly has been completed without an interrupt, the stresses are sent back to the user process for visual update. Although precious time has been saved by skipping all the previous ones and calculating results only for an actual setting, unavoidable delay of any visual feedback especially for the higher p, i. e. higher than 4, is experienced as already expected, since the time needed for a new computation is dramatically increasing in case of increasing p. Here, we profit from a hierarchical approach.

IV. Hierarchical Approach for the Bone

The hierarchical approach used in this test case is based on the usage of several, chosen by the user, different polynomial degrees for corresponding parallel processes (Fig. 8). The voxels' data as well as the data referring to user interaction is being sent to all of them via MPI, and they can all start their own computation, naturally, for lower p finishing faster than for higher p. As soon as any of them is finished, the results are sent to the front end and visualised.

What is accomplished in this way is that while the user's interplay with the settings is very intensive, he is getting not the most accurate, nevertheless *immediate* feedback about the effects of his changes, i.e. results for lower p, more specifically $p = 1$ or $p = 2$, being able to see the more accurate results in addition to this only as soon as he stops interacting and lets the simulation finish one iteration in the interactive computing loop for higher p values. As soon as user interaction starts over again, the results for the lowest p are immediately being visualised and the general impression about the tendency can be instantly gained one more time, switching afterwards gradually to higher levels of hierarchy until either the user starts interacting again or the highest accuracy is achieved, i.e. results for the highest hierarchy are received (Fig. 9). The number of MPI program instances, being executed for different p, i.e. hierarchy, can be chosen by the user.

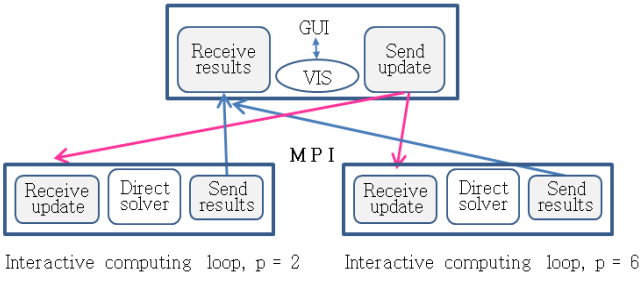

Fig. 8. Hierarchical approach – the communication pattern for the two chosen hierarchies – for instance $p = 2$ and $p = 6$. The updates are being sent to both of them, only the computation for $p = 2$ being able to finish and send back the result until the next update has been received. The computation for $p = 6$ is being interrupted and skipped all the time until the user stops interacting, when it has a chance to finish and send back the stresses for visualisation.

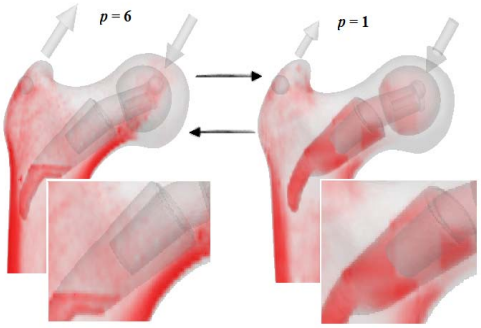

Fig. 9. Transition from $p = 6$ to $p = 1$ as soon as the user starts performing changes, i.e. changing the forces magnitude and direction, inserting an implant and moving it, etc. and getting the result again for $p = 6$ as soon as the interaction stops. In this way the user receives instantly feedback about the stress distribution, getting the finer result only when he stops interacting.

V. Results and Conclusions

The starting point of our work was a computationally efficient simulation and a sophisticated user interface with visualisation module, both opening the door for real-time interactive computing. The integration of our framework then comes into play not only to make more suitable for this purpose the way the data is communicated, but also to enable interrupting the simulation immediately and getting instant feedback ensued by any user interaction.

Evaluation of the performance on this particular test scenario, where the simulation is executed on multi-core architectures and connected to the visualisation front-end via a network still proved that this is yet another test case where the overhead caused by the framework itself is not significant.

The tests have been done in the past also on other multithreaded and distributed simulation test cases, where, as Fig. 10 and Fig. 11 show, we also got promising results.

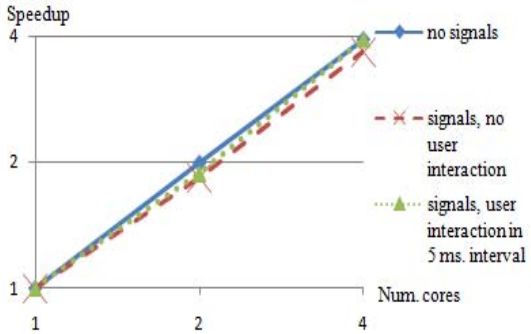

Fig. 10. Scenario with OpenMP parallelised Gauss-Seidel solver on a grid of size 500×500 without signals invoked, with signals invoked, but without any user interaction and with extremely intensive user interaction, i.e. each 5 milliseconds, shows excellent speedup results tested on 1, 2, and 4 cores.

In none of the test cases, even the one where the user interaction was invoked in 5-millisecond intervals, which is far more frequent than typically occurs in practice (the results of the measurements are shown in Fig. 10), did the integration of the framework significantly affect the overall execution time.

In the future, we will concentrate on testing the framework

in case of distributed and massively parallel version of this particular simulation. Load balancing techniques will be applied in order to involve all the available processes during the overall program runtime. This becomes especially challenging for the tasks organised in the hierarchy, where the number of processes involved is typically decreasing by the factor of 2^n on each level, where n is the dimension of the space. Thus, a sophisticated optimisation technique for the tasks with dependencies will be applied, which involves various heuristics in order to balance the work among processes in the optimal way.

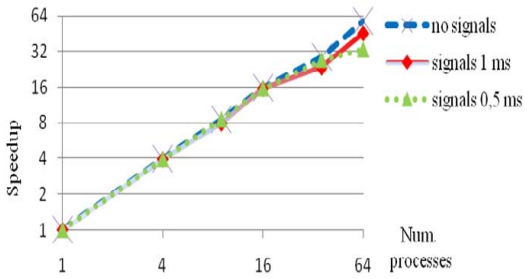

Fig. 11. Excellent speedup of a distributed Jacobi solver tested for up to 64 processes on AMD Opteron 850 processors at 2.4 GHz, without the integration of our framework and with it for different intervals in which signals occur (1 and 0.5 milliseconds); overhead introduced by the framework itself is negligible.

Also, the problem of data transmission for very high p will be tackled for minimising the amount of data which is being transferred in both directions.

REFERENCES

[1] J.D. Mulder, J.J. van Wijk, R. van Liere: A survey of computational steering environments, *Future Generation Computer Systems*, 15(1). 1999, pp. 119–129.

[2] J. Knežević, J. Frisch, R.-P. Mundani, E. Rank: Interactive computing framework for engineering applications, *Journal of Computer Science*, 7(5), 2011, pp. 591–599.

[3] A. Düster, Z. Parvizian, Z. Yang, E. Rank: The finite cell method for three-dimensional problems of solid mechanics, in *Proc. Computer Methods in Applied Mechanics and Engineering*, 2009, pp. 3768–3782.

[4] C. Dick, R. Georgii, R. Burgkart, R. Westermann: Computational steering for patient-specific implant planning in orthopedics, in *Proc. Visual Computing for Biomedicine*, 2008, pp. 83–92.

[5] C. Dick, R. Georgii, R. Burgkart, R. Westermann: Stress tensor field visualisation for implant planning in orthopedics, *IEEE Transactions on Visualization and Computer Graphics*, 15(6), 2009, pp. 1399–1406.

[6] Z. Yang, C. Dick, A. Düster, M. Ruess, R. Westermann, E. Rank, "Finite Cell Method with Fast Integration – An Efficient and Accurate analysis method for CT/MRI Derived Models," in *Proc. ECCM*, 2010.

[7] J. Knežević, R.-P. Mundani: Interactive computing for engineering applications, in *Workshop Proc. 22nd Forum Bauinformatik*, 2010, pp. 137–144.

[8] R.-P. Mundani, A. Düster, J. Knežević, A. Niggl, E. Rank: Dynamic load balancing strategies for hierarchical p-FEM solvers, in *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, LNCS 5759, Springer, 2009, pp. 305–312.

[9] J. Vetter, K. Schwan: High performance computational steering of physical simulations, in *Proc. 11th Int. Parallel Processing Symposium*, 1997.

[10] R. Nicolas, A. Esnard, O. Coulaud: Toward a computational steering environment for legacy coupled simulations, in *Proc. Sixth Int. Symposium on Parallel and Distributed Computing*, 2007.

[11] D. Jenz, M. Bernreuther: The computational steering framework steereo, in *Proc. of PARA 2010 Conf.: State of the art in Scientific and Parallel Computing*, 2010.

[12] K. Rycerz, M. Bubak, P. Sloot, V. Getov: Problem solving environment for distributed interactive applications, in *Proc. CoreGRID Integration Workshop*, 2006, pp. 129–140.

[13] R.-P. Mundani, H.-J. Bungartz, E. Rank, A. Niggl, R. Romberg, Extending the p-version of finite elements by an octree-based hierarchy," in *Domain Decomposition Methods in Science and Engineering XVI*, LNCSE 55, Springer, 2007, pp. 699–706.

Jovana Knežević was born in Užice, Serbia. In 2009, she has obtained M.Sc. degree in computer science at the Faculty of Mathematics, University of Belgrade. Since 2009, she is doing her PhD in computer science at the Chair for Computation in Engineering, Technische Universität München. Her main research interest covers interactive computing with the special focus on computational steering, parallelisation, and high-performance computing.

Ralf-Peter Mundani was born in Munich, Germany. In 2000, he obtained his diploma in computer science at the Faculty of Informatics, Technische Universität München (TUM), in 2005 his PhD at the Institute of Parallel and Distributed Systems, University of Stuttgart.

Since 2007, he has a post-doctoral position at the Chair for Computation in Engineering, Technische Universität München (TUM) and he is managing director of the Center for Simulation Technology in Engineering, part of TUM's International Graduate School of Science and Engineering.

Ernst Rank was born in Traunstein, Germany. In 1980, he obtained his diploma in mathematics at the Faculty of Mathematics, Ludwig-Maximilians-University Munich, in 1985 his PhD at the Faculty of Civil Engineering and Surveying, Technische Universität München.

In 1990, he was appointed professor for the Chair of Numerical Methods and Data Processing at University of Dortmund Germany, since 1997 he is heading the Chair for Computation in Engineering, Technische Universität München (TUM), and since 2009 he is also director of TUM's Graduate School. From 2002–2008 he was vice president of TUM.