2011 International Conference on Modeling, Simulation and Control
IPCSIT vol.10 (2011) © (2011) IACSIT Press, Singapore

Interactive Computing in Pre-operative Planning of Joint Replacement

Jovana Knežević [1 +], Ralf-Peter Mundani [1], Ernst Rank [1]

[1] Chair for Computation in Engineering, Technische Universität München

Abstract. 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 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-end with visualisation front-end and handicap a surgeon in observing which of his modifications leads to which outcome.

Keywords: Interactive Computing, Computational Steering Environment (CSE), Message Passing Interface (MPI), Bone Mechanics, Human Femur.

1. 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.

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. In *Magellan* a collection of instrumentation points know how to change an object without disrupting application execution. Here pending update requests are stored in a shared buffer until an application thread polls for them [9]. In *EPSN* API [10], 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 pre-determined blocking points. Further comparison of 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 Scientific 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 degeneration of bone tissue and lead soon unavoidably towards a new surgical intervention. The

+ Corresponding author. Tel.: +49 89 289-23044; fax: +49 89 289-25051.
E-mail address: knezevic@bv.tum.de.

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 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 instance, after sending an update of the settings, 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. 1.

The challenges in developing such a two-component analysis tool are described in more detail in [6, 4, 5]. Unfortunately, since, in the aforementioned version, 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, the higher polynomial degrees were used, the time until one could perceive the effect of his last change became longer.

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

Fig. 1. Von Mises stresses (calculated for polynomial degree of 6) of a healthy bone (left) and after a virtual surgery (right), under load exerted at the femur's head and the greater trochanter; darker colour refers to the regions with higher stress magnitude, thus, provides overview of how the implant changes the stress distribution in the surrounding bone tissue.

2. 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 equivalent of hardware interrupts, i.e. signals, in small, user-defined cyclic intervals, followed by a check for updates [2].

If there has been some 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 himself 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 starting computation with new settings over 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. With some intermediate (one iteration in the case of an iterative solver, e.g.), or the complete computation (in the case of a direct solver, e.g.) being finished without an interrupt, new results are handed on to the user process for visualisation. Nevertheless, 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] the design of our framework takes into consideration and supports different parallel paradigms, which results in an extra effort to ensure correct program execution and avoid synchronisation problems when using threads.

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 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, 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 a random thread in each active MPI process is being interrupted by signals to check for the updates, except that now all the processes have to be *explicitly* notified about the changes performed by a user, which involves additional communication overheads. 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. 2).

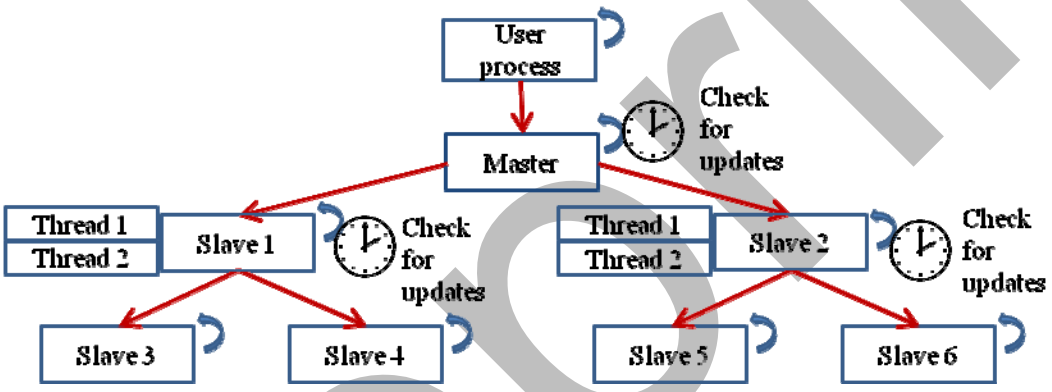

Fig. 2. Hierarchical communication pattern and transfer of the signal to all the processes. User process sends the new settings to the master process of the simulation; both master and slave processes check for updates in small, cyclic, simulation-specific intervals until an update is received, when it is transferred to all of the processes lower in the communication hierarchy.

3. Test case – the Bone

The main functionality provided to aid the pre-operative planning consists of the insertion of an implant, changing its 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 a standard consumer-class hardware, and this even for higher accuracy, i.e. polynomial degrees of basis functions used in FCM higher than 4, our framework with several valuable features has been utilised. 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.

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 the second thread, a loop for sending 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 also completely independent of and unsynchronised with the remaining, i.e. third, thread, which is dedicated to waiting to receive results as soon as these are available. A simplified communication pattern between the modules on the front-end and simulation on the back-end is illustrated in Fig. 3.

In order to benefit from this pattern, what becomes a challenge is exploring the way in which the simulation can simultaneously become aware of changes, i.e. when and how to interrupt the simulation kernel so that it can instantly receive an update. To describe the challenge in more detail, we provide a basic overview of the simulation kernel structure.

3. 1 The Simulation Kernel

At the beginning, the femur voxel information generated on the visualisation side based on the QCT scans is transmitted over the network, thus the rectangular domain which embeds the entire femur is generated. The domain is 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, 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 equation from static elasticity theory

$$Ku = f, \tag{1}$$

where K is known as the stiffness matrix, u displacement vector of all vertices, and f force vectors applied to the system. The stiffness matrix K is assembled from the element stiffness matrices, referring to individual elements laying inside the femur's physical domain.

Described initialization and meshing steps are followed by an interactive computing loop, which consists of receiving of the user update, pre-processing, solver of the aforementioned system of equations, post-processing and, finally, sending results to the user. Concerning the system of equations, due to its poor condition numbers, sophisticated iterative solvers fail to be efficient. Therefore, a direct solver with hierarchical concepts, i.e. exploiting an octree data structure based on a nested dissection of 3D domain, is used [8]. 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. 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.

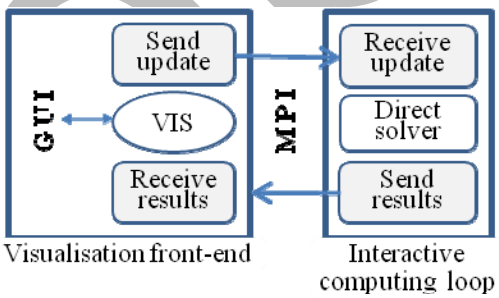

Fig. 3. Communication pattern between two components. From the user front-end the changes of the data are recognised on a per-frame basis and sent immediately to the simulation via non-blocking routines. The simulation results, in terms of stresses, are calculated and sent back to the user.

3. 2 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 the 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. 4. The described algorithm, as presented in [8] performs excellent in the case of hybrid parallelisation.

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 check 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 previously mentioned hierarchical data structure and the simulation variables are set in the way which ensures skipping the rest of the nodes, as shown in Fig. 4, thus all the layers of 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.

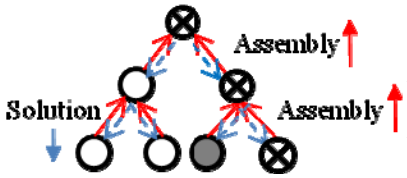

Fig. 4. Nested dissection solver. Dashed arrows indicate the solution sequence and the solid ones the assembly. In the case of an interrupt being caught while processing the filled node at the bottom of the hierarchy, supposing the tree-like structure is being traversed in depth-first manner, the processing of the nodes marked with the cross is skipped.

In addition to the guaranteed data values consistency necessary for the correct program execution, mentioned at the beginning of the Section 2, 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 octree is being processed by separate threads, i.e. the solver code is parallelised via OpenMP, unexceptionally in this test case, it is ensured that when a new update is recognised 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 complete assembly is done 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 if increasing p. In this case we profit from a hierarchical approach.

4. 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. 5). 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. In this way, while the user's interplay with the settings is very intensive, at least he is getting immediate feedback about the effect of his changes, i.e. results for lower p, nevertheless, being able to see the more accurate results only as soon as he stops interacting and lets the simulation finish one iteration in the interactive computing loop for a higher p. The number of MPI program instances being executed for different p can be chosen by the user.

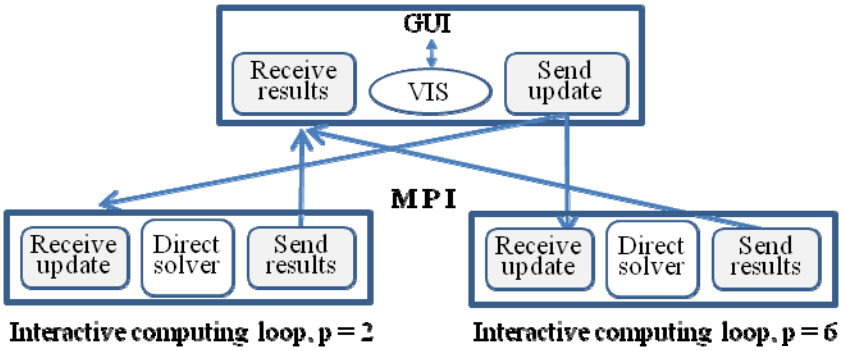

Fig. 5: Hierarchical approach – the communication pattern for 2 chosen hierarchies.

5. Results and Conclusions

Starting point of our work was a computationally efficient simulation and a sophisticated user interface with visualisation module, both opening all doors 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 of 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 architecture and connected to the visualisation front-end via 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 distributed simulation test cases, where, as Fig. 6 shows, we got promising results. In the future we will concentrate on testing the framework in the case of distributed, and massively parallel version of this simulation. Also, the problem of data transmission for very high p will be tackled and minimising the amount of data which is being transferred in both directions.

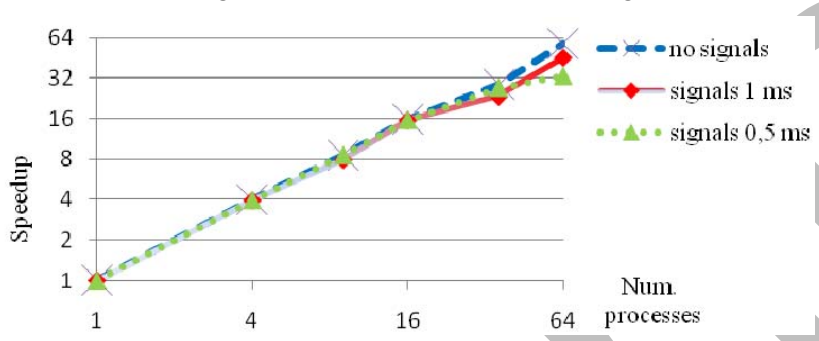

Fig. 6: Excellent speedup of a distributed simulation, for up to 64 processes, with/without integration of our framework for different intervals in which signals occur (1 and 0,5 milliseconds). Overhead caused by the framework is negligible.

6. Acknowledgements

This research 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.

7. 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. 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: Computer Methods in Applied Mechanics and Engineering 197. 2009, pp. 3768 — 3782.

[4] C. Dick, R. Georgii, R. Burgkart, R. Westermann: Computational Steering for Patient-Specific Implant Planning in Orthopedics. In: Visual Computing for Biomedicine. 2008, pp. 83—92.

[5] C. Dick, R. Georgii, R. Burgkart, R. Westermann: Stress Tensor Field Visualization 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: ECCM. 2010.

[7] J. Knežević, R.-P. Mundani: Interactive Computing for Engineering Applications. In: 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. M. R. et al. ed. Springer. 2009.

[9] J. Vetter, K. Schwan: High Performance Computational Steering of Physical Simulations, 11th International Parallel Processing Symposium, 1997.

[10] R. Nicolas, A. Esnard, O. Coulaud: Toward a Computational Steering Environment for Legacy Coupled Simulations, Sixth International Symposium on Parallel and Distributed Computing. 2007, pp. 43.