EXTENDING THE SCIRUN PROBLEM SOLVING ENVIRONMENT TO LARGE-SCALE APPLICATIONS

Jovana Knežević, Ralf-Peter Mundani, Ernst Rank
Computation in Engineering, Technische Universität München
Arcisstraße 21, 80333 Munich, Germany

Ayla Khan, Chris R. Johnson
Scientific Computing and Imaging Institute, University of Utah
72 S Central Campus Drive, Salt Lake City, UT 84112

ABSTRACT

To make the most of current advanced computing technologies, experts in particular areas of science and engineering should be supported by sophisticated tools for carrying out computational experiments. The complexity of individual components of such tools should be hidden from them so they may concentrate on solving the specific problem within their field of expertise. One class of such tools are Problem Solving Environments (PSEs). The contribution of this paper refers to the idea of integration of an interactive computing framework applicable to different engineering applications into the *SCIRun* PSE in order to enable interactive real-time response of the computational model to user interaction even for large-scale problems. While the *SCIRun* PSE allows for real-time computational steering, we propose extending this functionality to a wider range of applications and larger scale problems. With only minor code modifications the proposed system allows each module scheduled for execution in a dataflow-based simulation to be automatically interrupted and re-scheduled. This rescheduling allows one to keep the relation between the user interaction and its immediate effect transparent independent of the problem size, thus, allowing for the intuitive and interactive exploration of simulation results.

KEYWORDS

Interactive Computing, Computational Steering Environment (CSE), Problem Solving Environment (PSE), *SCIRun*.

1. INTRODUCTION

With new hardware technologies, efficient algorithms, and parallelization strategies, the simulation of very complex physical phenomena, which used to be inconceivable, has become a realistic endeavor. It assumes the ability to model a particular physical problem domain, appropriate boundary conditions, and numerical approximations of the governing system of equations to be solved. The result is then validated and visualized for more intuitive interpretations.

Constructing a model refers to the geometrical definition of a physical domain, in which a continuous structure has been discretized. Typically, a new model must be generated for each new configuration, making this phase one of the most time-consuming. For numerical approximations of the governing partial differential equations and corresponding boundary conditions, one can use either common discretization methods (e. g. the finite element (FE) and finite difference (FD) method), hierarchical methods (e. g. multigrid and relatives), or a combination of the two. This yields a linear system, $A \cdot x = b$, where the time needed for calculating the vector x is correlated with the number of its elements. The final step—efficient graphical representation of the resulting, often large, data sets—is itself a considerable task.

Aforementioned cycles are traditionally carried out as a sequence of steps. However, the ever-increasing range of specialists in developing fields has necessitated a collaborative, interactive approach with the computational model. This requires real-time feedback from the running simulation, while experimenting with different simulation setups. Problem Solving Environments (PSEs) are popular tools that facilitate interactions with complex models, without requiring specialists to know their algorithmic, data, or

visualization structures. In short, these are user-friendly tools for guiding the numerically approximated problem solution.

One of the most prominent and widely used PSEs is *SCIRun* (SCI Institute, 2012). Compared to other state-of-the-art approaches such as *Cactus* (Allen et al. 2000), which enables runtime steering of parallel simulations, or *G-HLAM* (Rycerz et al. 2008), which addresses migration and monitoring mechanisms, *SCIRun* is a modular software package that follows the dataflow model and allows for GUI-based designs of the simulation flow, similar to the one in dataflow environments such as *AVS*[1], IBM *Open Visualization Data Explorer*, *OpenDX*[2], and *IrisExplorer*[3].

Due to its modularity, ease of extension, portability to different platforms, and convenience for interactive computational steering, it is used for many biomedical and other applications, within the SCI Institute and beyond. Immediate responses to the running simulation are achievable in real-time, up to the certain problem sizes. With the increase of the size of the problem—mesh resolution in the FE approximation, for example—the observable, causal relationship becomes less intuitive. Therefore, additional strategies need to be applied – in particular, saving cycles by skipping redundant work as early as possible.

To sum up, for building a PSE with real-time computational steering enabled for large-scale problems, one has to consider many aspects: from efficient (and parallelized) simulation consisting of the aforementioned phases, which can all be interrupted by the user at any point (to be re-executed with the updated settings), and fast transfer of the update, as well as the simulation results (Summa et al. 2011), to real-time visualization.

The contribution of this work is to allow for interactive feedback of the computational model, even for more time- and memory-consuming problems. This is achieved by interrupting the running simulation via software equivalents of hardware interrupts, i. e. signals, in order to skip the redundant computational cycles as soon as any changes are performed by the user. Due to the dataflow software architecture of the *SCIRun*, only necessary simulation modules are re-executed afterward.

2. SCIRUN

SCIRun is a PSE intended for interactive construction, debugging, and steering of large-scale, typically parallel, scientific computations (Shepherd, Johnson 2009). It is a modular, easily extendable software package based on dataflow programming, and it provides efficient and comfortable environments for interactive computational steering.

Every *SCIRun* simulation is designed as a network of computational components, i. e. modules, connected via input/output ports. It allows for new modules and easy modification of individual modules without affecting others. As a design pattern, object-oriented *SCIRun* code uses the Model-View-Controller paradigm. The Controller entity, i. e. an instance of the Scheduler class, is in charge of all the modules and their order of execution. After one module is triggered and stored in the queue for execution, dependent modules are in the same manner stored for execution.

A user interface is provided for every module in order to enable the modification of corresponding parameters. Regarding discretization parameters, one can choose a mesh resolution for all spatial directions. For solving the resulting linear system of equations, the selection is made among iterative solution methods (Conjugate Gradient (CG), Biconjugate Gradient (BCG), Jacobi, and Minimal Residual (MINRES)), as well as among different pre-conditioners; in addition, one may change tolerances, the maximal number of iterations, levels of accuracy, and other numerical parameters. Within the solution of a linear system of equations, users may also receive visual feedback on residual error or current iteration, after which they can interact with and re-direct the solution process. One may also decide to change other simulation-specific parameters, such as electrical conductivity in defibrillation-like simulations.

Upon initiating the simulation, the scientist views initial results, error per element of the finite element analysis, etc., then (s)he may decide whether to continue the computation using different discretization parameters or to restart the computation with different input conditions. Traditionally, results are exported to

[1] Advanced Visual Systems' (www.avs.com)
[2] www.research.ibm.com/dx
[3] www.nag.com

disk, and/or piped into a separate visualization software package once all computations are completed (Johnson and Parker 1995). Within *SCIRun*, visualization modules are integral part of the dataflow. It is possible to visualize and explore intermediate results after a pre-defined number of iterations, while the calculations continue to progress. Since some of those phases can be very time-consuming, it is very important to be able to interrupt them instantly with setting changes, and thus, to automatically start anew with modified parameters. This is where the interactive computing framework comes into play.

3. INTERACTIVE COMPUTING FRAMEWORK

In order to achieve an immediate response to changes made by the user, the regular course of the simulation is interrupted via signals in small, user-defined cyclic intervals, followed by a check for updates (Knezevic and Mundani 2010). If there has been any change on the user side, simulation-state variables are manipulated in order to skip the redundant computation and automatically calculate anew, starting from a particular point in the algorithm according to the updated settings (new geometry, boundary conditions, mesh resolution, etc.). It is then the responsibility of a user to instruct the simulation program how the received data should be assigned to the simulation.

With some intermediate (e. g. one iteration in case of an iterative solver) or the complete computation (e. g. in case of a direct solver) being finished without an interrupt, new results are handed on to the user process for visualization. The framework is intended to be integrated in various application scenarios; hence, it cannot predict how the results should be interpreted in each of them. It is, therefore, the user's responsibility to prescribe, on the front-end process, how to interpret the received data so that it can be appropriately visualized.

As elaborated in (Knezevic et al. 2011), to guarantee the correct execution of a program, one should use certain types of qualifiers for variables that are subject to sudden changes or interrupts. Deterministic behavior of the program has to be guaranteed by ensuring atomicity of certain arithmetic operations, e. g., on different architectures and accessing the correct values of variables in the main memory, instead of potentially outdated values in the cache.

An even more challenging task lies in applications that are amenable to concurrent executions; thus, they are programmed using either shared memory, message passing, or a combination in hybrid parallel algorithms. The design of the framework takes into consideration and supports all of these scenarios (Knezevic, Mundani and Rank 2011), although this results in an extra effort to ensure correct program execution and avoid synchronization problems when using threads, as is the case with *SCIRun* simulations.

4. INTEGRATION INTO *SCIRUN*

SCIRun provides an optimal software environment for integrating the aforementioned interactive computing framework. The dataflow model allows for triggering only the re-execution of the necessary modules during user updates. First, the updated module is stored in the processing queue. Then, it is supposed to trigger the scheduling of the modules whose input ports are connected with output ports of this module directly or indirectly. The modules that come earlier than the updated module in the execution pipeline are not being triggered.

In this already mature and sophisticated environment for computational steering, our goal is to have real-time feedback for even more time- and memory-consuming simulations, i. e. when a module execution requires more time than desirable within the computational steering loop. Hence, our intention is to interrupt the module currently being executed and skip its redundant cycles, as well as remove any module previously stored for execution from the actual schedule.

The concept is tested on several different simulations to evaluate the user response for different overall execution times, different orders of module execution, choice of parameter changes, etc. These scenarios are: (1) a simulation that facilitates early detection of acute heart ischemia, and (2) two defibrillation-like simulations, one on a homogeneous cube domain, and the other on an inhomogeneous human torso.

4.1 Tool for early detection of heart ischemia

Myocardial ischemia is a disease characterized by reduced blood supply of the heart muscle, usually due to coronary artery disease. Symptoms may include characteristic chest pain on exertion and decreased exercise tolerance (Wikipedia 2012a). It is the most frequent cause of death in most Western countries, and a major cause of hospital admissions (Podrid and Myerburg 2005). Since early detection may lead to the prevention of further complications, by measuring many anatomic details and electrical measurements, scientists hope to detail what occurs in the border zones between the healthy and ischemic tissue layers.

The aim of this application is the generation of a quasi-static volume conductor model of an ischemic heart, based on data from actual experiments (Stinstra and Swenson, 2012). Modeling pipeline requires the generation of experiment-specific models of the myocardium, based on the MR images/scans of a dog heart. The known values are extracellular cardiac potentials as measured by electrodes on an isolated heart or with needles inserted into the heart. The transmembrane potential (the potential difference between the intracellular and extracellular space) is not the same for ischemic and healthy cells. The latter effect causes so-called injury currents to flow within the intracellular and extracellular spaces. These can be observed at the surface of the heart as potential differences that translate to so-called ST shifts in the ECG (Stinstra and Swenson, 2012). A network of modules is constructed within *SCIRun* to simulate and then render a model of the transmembrane potential of a dog's myocardium in experiments (Figure 1, left).

In the network of modules created in this simulation, typically the most computationally expensive step is the SolveLinearSystem module. Thus, the first challenge is how to interrupt it as soon as any change is made by the user—in particular, the changes done via UI to this module. So far, it is provided by *SCIRun* that only necessary modules will be re-executed due to the change, however, the current module computation has to run till the end. Thus, to achieve *immediate* skipping of the outdated computation in the iterative-solver algorithm of the system of linear equations, the maximal number of iterations (which is normally user interface variable) is newly replaced by the globally visible variable registered in the framework. This value is then manipulated in the signal handler, i. e. set to some value outside of the domain of the iterator index. Hence, the iterative solver algorithm, which is the major part of *execute* function of the module, *instantly* recognizes that is should terminate. More precisely, this happens as soon as this value can be compared with the current iteration number. The *execute* function of this module also has to be re-scheduled afterward with the new user-applied settings. However, one has to take care that the previous interrupted execution of the same module is finished in a clean way.

Our contribution, thus, also assumes doing all the steps that would have been taken within the *execute* function without any interrupt. Moreover, the *execute* function has to be called anew (in order to trigger re-computation instantly). For this, we have to take care of several things, the most relevant of which is the Scheduler being explicitly informed about this new execution. The Scheduler also has to confirm storing the matching task for the execution, in terms of its identification number. All the input and output ports, which were opened by the previous *execution* call, now have to be both closed and re-opened in order for the SolveLinearSystem variables to re-initialize properly. It consists of re-initializing the maximum number of iterations, as well as some other user-interface variables, which would have been automatically re-initialized without the enforced interruption of the module execution.

One of those variables is, for instance, determining whether the partial solutions should be emitted. We choose for testing of the framework the most computationally demanding scenario—emitting the intermediate solution after *each* iteration. This involves scheduled executions of several visualization modules, each of which takes a few seconds, after each iteration. After an interrupted iteration, the preview of old results is cancelled. The execution of all the modules, which would happen after SolveLinearSystem, has to be aborted. This is achieved within this work by throwing an exception in the class GetFieldBoundary, since this exception gets automatically caught in all modules, such as ExtractIsosurfaceByFunction, ApplyMappingMatrix, ShowField, ShowFieldGlyphs, and ViewScene.

4.2 Defibrillation

Defibrillation is a common therapy for life-threatening cardiac and ventricular disorders. It consists of delivering a dose of electrical energy to the affected heart with a device that terminates the arrhythmia and allows normal sinus rhythm to be reestablished by the body's natural pacemaker. Implantable Cardioverter

Defibrillators (ICDs) are relatively common—implantable—devices that provide an electric shock to treat fatal arrhythmias in cardiac patients (Wikipedia 2012b). Children, due to their smaller size and often abnormal anatomy, require more specialized ICD configuration than adult patients (Burton et.al. 2011). Given a pattern of source activation, one of the bioelectric field problems cardiologists are interested in is determining the electric activity that would result through the rest of the domain. Such studies are used when investigating internal implantable defibrillator designs. Again, the goal is to simulate those governing equations using discrete numeric approximations. By building a computational model of a patient's body and mapping conductivity values over the entire domain, we can accurately compute how activity generated in one region would be remotely measured in another region (Weinstein, 2005).

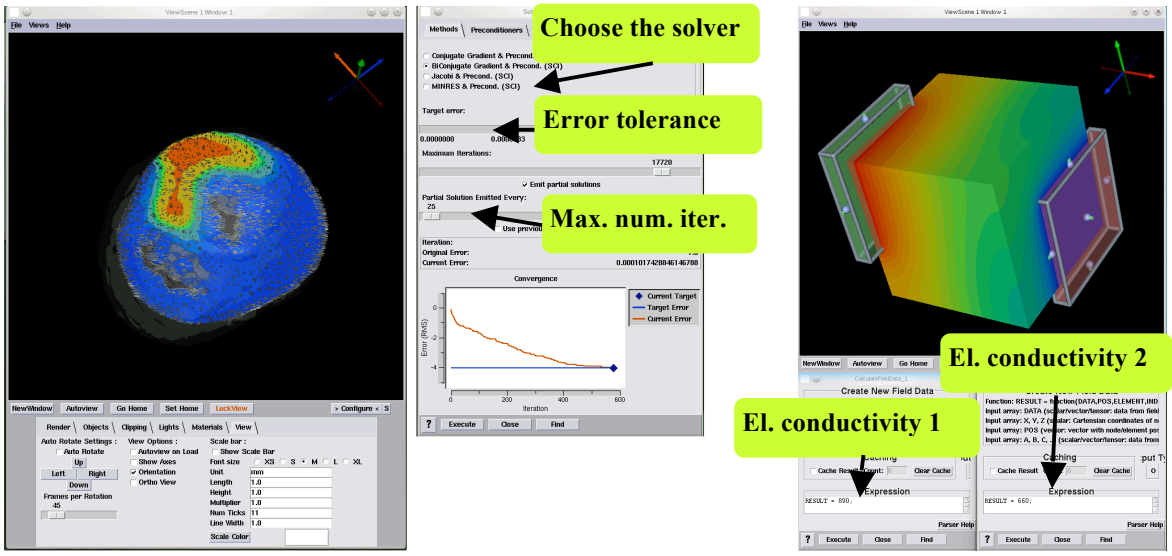
4.2.1 Simplified defibrillation simulation on a homogeneous cube

First, we consider a simpler defibrillation-like example—a simulation of the electrical conduction on a homogeneous cube domain with two interactively placed electrodes— in conjunction with our interactive computing framework. Each of the electrodes is assigned its own conductivity value. It is then explored with different values for both of the electrodes.

In this case, we want to place two electrodes with two input fields, so we use the CalculateFieldData2 module. The CalculateFieldData2 defines a whole range of mathematical operators that can be applied to various fields and a range of possible input streams—from the location of nodes to the data values located in the field. It operates by taking data from the two input fields as two input streams and applying the operation specified inside the UI of the module to each pair of data from the two fields. Hence, this module defines an operator that works on a stream of input data. In this case, the input function is specified as $RESULT = DATA_1 * DATA_2$. $DATA_1$ refers to the data from the first field and $DATA_2$ to the data from the second field, as represented in the Figure 1 (right).

As soon as the input field in any of these two modules has been modified, the Scheduler gets implicitly informed; thus, it is newly provided that it cancels the execution of all the scheduled modules that have not begun yet by making sure an exception is employed. In the case of the modules that are currently in the execution phase, the iterator indexes are manipulated within our framework functionality, as described before, in order to stop their execution instantly. Changing the input field of CalculateFieldData2 automatically triggers the re-execution of all the modules following it in the pipeline.

Figure 1: left: Heart ischemia with graphical user interface; right: defibrillation on a Cube with graphical user interface



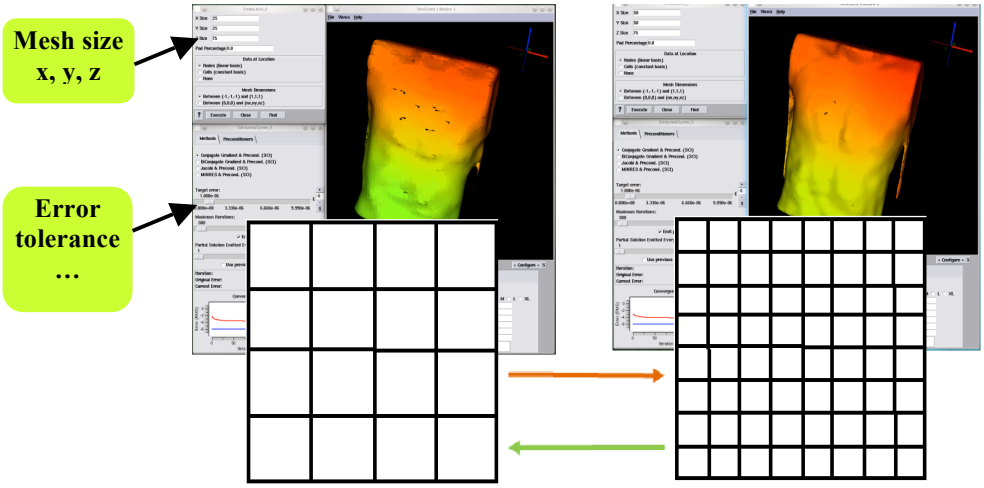
4.2.2 Defibrillation on a human torso

The following case study illustrates how to determine optimal energy discharge and placement of the ICD in the human torso (Figure 2). Segmentation of patient MRI or CT data provides the torso geometry into which ICD geometry is interactively placed. Local mesh refinement around the ICD reduces the overall number of

elements while maintaining crucial details. (Burton et.al. 2011) All tissues are modeled as passive with ICD geometry locations acting as sources and sinks. This allows the solution to be approximated via FE by simulating Poisson's equation.

In addition to manipulating solver-related parameters for the resulting system of the linear equations, and conductivity values, we can this time experiment with different mesh resolutions in FE approximation (to test our framework). This allows for previewing the solution on a coarser grid and switching to the finer one, once the user is satisfied with the current setting. In order that our framework has a desired effect, similar to the simulation on the homogeneous cube with different conductivities of the electrodes, it has to be ensured that the Scheduler now reacts on the change of the mesh resolution by allowing an exception for some of the modules following the updated CreateLatVol module within the network. Additionally, due to the framework, the iterator index used in other modules, such as SolveLinearSystem, is manipulated to skip any redundant computation.

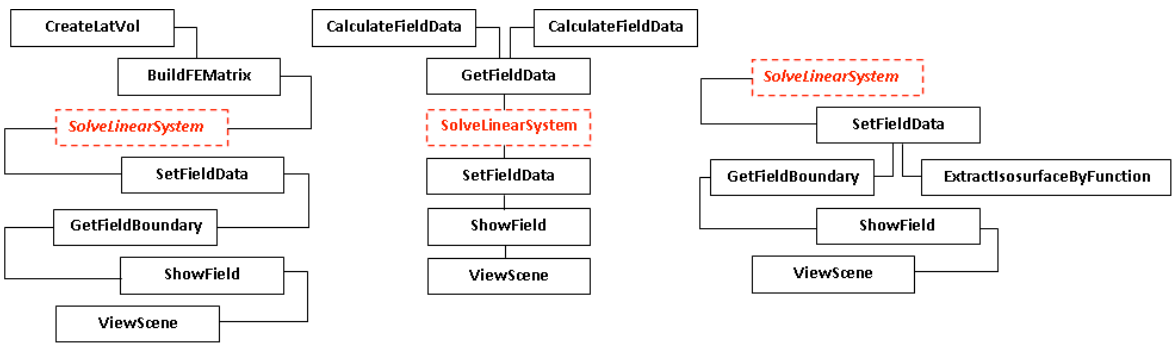
Figure 2: Changing mesh base parameters in the CreateLatVol module of a simplified defibrillation simulation on a torso



4.3 Interrupting the data flow

To summarize, in *SCIRun*, the challenges of getting an immediate feedback/response of the simulation depends on many factors. The difficulty depends not only on the size of the problem, but also on the choice of the modified parameters within the simulation, since this determines which parts of the dataflow have to be re-executed. The earlier in the execution pipeline the parameter appears, the more challenging it is to provide the real-time response to the user changes. Figure 3 represents the structure of each aforementioned simulation realized within the *SCIRun* environment, where it becomes clearer which modules need to be interrupted or cancelled in order to keep an intuitive connection between the user's change and its effect.

Figure 3: Sketches of the modules for: left - torso defibrillation-like simulation, middle - homogeneous cube defibrillation-like simulation, right – detection of heart ischemia; dashed-line boxes: most time-consuming module typically interrupted in the middle of execution by a user.



5. CONCLUSION

We have tested all three simulation scenarios in order to estimate the overhead of the framework. The tests have been made for different update intervals, namely, 5, 2, or 1 millisecond for different solvers of linear systems of equations (Figure 4).

In the case of the electrical conduction simulation on a human torso with inserted defibrillators, with the mesh resolution (50 × 50 × 75), we can see that in the case of the shortest interval (i. e. 1 millisecond), the overhead caused by the framework is around 15%. However, by making the interval longer (e. g. 2 or 5 milliseconds), the overhead is reduced to around 5%, and 2-3%, respectively. While increasing this interval, however, an end-user does not observe the difference in terms of the simulation response. Therefore, one may conclude that it is recommendable to experiment with different intervals for a specific simulation.

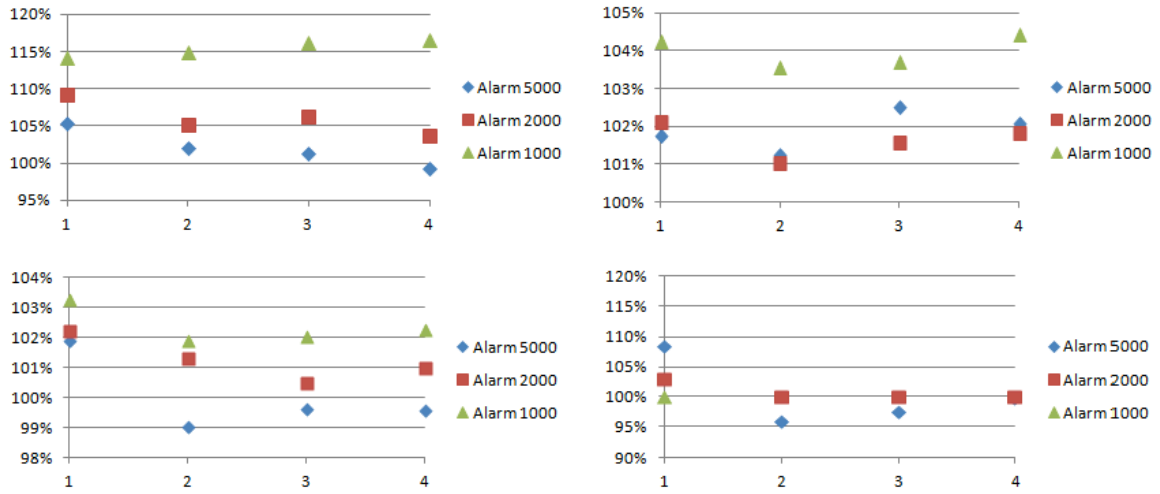
For the simulation based on the heart-ischemia model, one observes even less overhead—namely not more than 5% in all the cases (taking into consideration also that for such small overall execution times, 1-2% overhead can also be assigned to the cache coherence and measurement precision issues).

In the case of electrical conduction simulations on the homogeneous cube domain (with two inserted electrodes), the tests have been done for two problem sizes—first, mesh (32 × 32 × 32), second (64 × 32 × 32), and again not more than 5% overhead is observed, except for one of the solvers (CG) on the mesh 64 × 32 × 32, where it is close to 10% for alarm intervals of 5 milliseconds. This result shows once again that it is worth experimenting with different alarm intervals for a specific simulation, if one observes that the execution time has been significantly extended.

The steering process itself runs now intuitively and smoothly. For all the tested data sizes the immediate visual response to user changes is made possible, i. e. within a second. However, additional re-use of previous computation results for new computations when changing certain parameters should be considered (if applicable for some of the applications). It is also possible to test the integrated framework for any other simulation cases without additional code changes.

From the user effort point of view, the integration of the interactive computing framework turned out to be quick and straightforward, as was also the case for the other application scenarios (Knezevic et al. 2011, Knezevic, Mundani and Rank 2011, etc.). Significant advantages of the SCIRun software package over other test applications were its modularity and being based on a dataflow model. Due to its modularity, the best interfaces to the framework could be easily recognized. The dataflow model has contributed to an automatic re-execution of only necessary modules. The underlying Model-View-Controller design pattern, however, has introduced a few difficulties related to enforcing the Controller entity to cancel scheduled, but outdated, jobs.

Figure 4: top: left – torso, right – heart ischemia; bottom: cube: left – mesh 32 x 32 x 32, right 64 x 32 x 32; x-axis: options for the solver: 1- CG, 2- BCG, 3- Jacobi,4- MINRES; y-axis: new execution time in % of the execution time without the framework being integrated.



ACKNOWLEDGEMENT

This work was made possible in part by software from the NIH/NIGMS Center for Integrative Biomedical Computing, 2P41 RR0112553-12. It was accomplished in winter 2011/12 during a three-month research visit of Jovana Knežević to the Scientific Computing and Imaging Institute, University of Utah. She would like to express her very special appreciation and gratitude to this institute for hosting this stay, and also gratefully acknowledge the financial support of the Munich Centre of Advanced Computing (MAC) and the International Graduate School of Science and Engineering (IGSSE) at Technische Universität München.

REFERENCES

Allen, G., Benger, W. , Goodale, T. , Hege, H.-C., Lanfermann, G., Merzky, A., Radke, T., Seidel, E., Shalf, J., 2000. The Cactus Code: a problem solving environment for the grid, Proc of The Ninth International Symposium on High-Performance Distributed Computing, Pittsburgh, Pennsylvania, pp: 253 – 260.

Burton, B.M., Tate, J.D., Erem, B., Swenson, D.J., Wang, D.F., Steffen, M., Brooks, D.H., Van Dam, P.M., and Macleod, R.S., 2011. A toolkit for forward/inverse problems in electrocardiography within the SCIRun problem solving environment, Proc. of *Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, pp. 267—270.

Johnson, C.R. and Parker, S. G., 1995. Applications in Computational Medicine using SCIRun: A Computational Steering Environment, In Supercomputer '95. Springer Verlag, pp. 2—19.

Knežević, J., Frisch, J., Mundani, R.-P. and Rank, E., 2011. Interactive computing framework for engineering applications, *Journal of Computer Science*, Vol. 7, No. 5, pp. 591—599.

Knežević, J., Mundani, R.-P., and Rank, E., 2011. Interactive Computing−Virtual Planning of Hip Joint Surgeries with Real-Time Structure Simulations, International Journal of Modeling and Optimization, Vol. 1, No. 4, pp. 308-313.

Knežević, J., Mundani, R.-P., 2010. Interactive computing for engineering applications, *Proc. 22nd Forum Bauinformatik*, Berlin, Germany, pp. 137—144.

Podrid PJ, Myerburg RJ. Epidemiology and stratification of risk for sudden cardiac death. Clin Cardiol 2005;28:I3–I11.

Rycerz, K., Bubak, M., Sloot, P., Getov, V., Gorlatch, S., 2008. Problem Solving Environment for Distributed Interactive Applications, Achievements in European Research on Grid Systems, Springer US, pp. 55 – 66.

SCI Institute, 2012. SCIRun, SCIRun: A Scientific Computing Problem Solving Environment, Scientific Computing and Imaging Institute (SCI), download from: http://www.scirun.org.

Shepherd, J.F. and Johnson, C.R., 2009. Hexahedral Mesh Generation for Biomedical Models in SCIRun, In Engineering with Computers, Vol. 25, No. 1, pp. 97—114.

Steffen, M., Tate, J., Stinstra, J., Defibrillation Tutorial, VIewed March 2012,

< http://www.sci.utah.edu/devbuilds/scirun_docs/DefibrillationTutorial.pdf>

Stinstra J., Swenson, D., Ischemia Model Tutorial, 2012, Viewed March 2012,

<http://www.sci.utah.edu/devbuilds/scirun_docs/IschemiaModelTutorial.pdf>

Summa, B., Scorzelli, G., Jiang, M., Bremer, P.-T. and Pascucci, V., 2011. Interactive Editing of Massive Imagery Made Simple: Turning Atlanta into Atlantis, *In ACM Transactions on Graphics (TOG)*, Vol. 30, No. 2, Article 7.

Weinstein, D.M., Parker, S., Simpson, J., Zimmerman K., Jones, M., 2005. Visualization in the SCIRun Problem-Solving Environment, Visualization Handbook, Elsavier, pp. 615—632.

Wikipedia 2012a, Viewed 26 June 2012, <http://en.wikipedia.org/wiki/Ischaemic_heart_disease>.

Wikipedia 2012b, Viewed 26 June 2012, <http://en.wikipedia.org/wiki/Defibrillation>.