Vascular System Modeling in Parallel Environment - Distributed and Shared Memory Approaches

Krzysztof Jurczuk*, Marek Kretowski, and Johanne Bezy-Wendling

***Abstract*—The paper presents two approaches in parallel modeling of vascular system development in internal organs. In the first approach, new parts of tissue are distributed among processors and each processor is responsible for perfusing its assigned parts of tissue to all vascular trees. Communication between processors is accomplished by passing messages and therefore this algorithm is perfectly suited for distributed memory architectures. The second approach is designed for shared memory machines. It parallelizes the perfusion process during which individual processing units perform calculations concerning different vascular trees. The experimental results, performed on a computing cluster and multi-core machines, show that both algorithms provide a significant speedup.**

***Index Terms*—computational modeling, vascular system, parallel computing, distributed memory algorithms, shared memory algorithms.**

I. Introduction

In this paper, we deal with vascular system modeling which can promote understanding of complex vascular processes (e.g. angiogenesis) and their influence on pathology development [1]. In our case, we focus on the vasculature modeling towards an image generation. Many diseases are directly related to changes in vessels structures and a lot of these modifications can be visible in medical images, especially when a contrast agent is administrated.

In our previous studies, we proposed a two-level physiological model [3] which is able to reflect both morphology and functions of vascular networks in clinical images. The solution is a combination of a macroscopic model (vascular network growth and pathological anomalies) and a microvascular model (blood flow simulation in capillaries and contrast agent diffusion processes). Moreover, we coupled the model with imaging simulators (CT [2] and MR [4]). Hence, the whole solution constitutes a good way for a better understanding of medical images by linking image descriptors with physiological perturbations and markers of pathological processes.

The structure of simulated vascular network is obtained in the sequential algorithm of vascular development caused by a progressive increasing number of cells. Although we applied many code optimizations, the simulation is still time expensive process. Following the recent technological advances in parallel computers [5], we propose two approaches able to simulate the vascular development in a parallel environment. The first one is designed to distributed memory architectures, like computing clusters that are able to provide high computational performance but are also quite expensive. On the other hand, the second approach is strictly suited to shared memory environments, like multi-core/multi-processor machines that become affordable and consequently more attainable even for home users. Nevertheless, the main disadvantage of shared memory computers is the lack of scalability between memory and processing units.

This work was supported in part by the grant W/WI/3/11 from Bialystok University of Technology and in part by the Region Bretagne in France.

*K. Jurczuk is with the Faculty of Computer Science, Bialystok University of Technology, Wiejska 45a, 15-351 Bialystok, Poland and INSERM U642, Rennes F-35000, France; LTSI, Universite de Rennes 1, Rennes F-35000, France (e-mail: k.jurczuk@pb.edu.pl).

M. Kretowski is with the Faculty of Computer Science, Bialystok University of Technology, Wiejska 45a, 15-351 Bialystok, Poland (e-mail: m.kretowski@pb.edu.pl).

J. Bezy-Wendling is with INSERM U642, Rennes F-35000, France and LTSI, Universite de Rennes 1, Rennes F-35000, France (e-mail: johanne.bezy@univ-rennes1.fr).

The common aim of these two approaches is to accelerate the complex vascular growth simulation. As a result, we will be able to introduce more sophisticated details into the model, to test multiple sets of parameters corresponding to different behaviors of the system or to perform simulations with more complex configurations (e.q. more vessels or smaller vessels). Moreover, our intention is to bring the model closer to reality in which analogous processes of vascular development are performed inherently in a parallel way [6].

Many other vascular systems have been proposed, e.g. a fractal model [7], Constrained Constructive Optimization (CCO) model [8] or macroscopic [9] and microscopic models [10] developed by Szczerba et al. However, as far as we know, all the previous solutions, have been using only sequential approaches to develop vascular systems.

In Section II the main components of the vascular model are described. In Section III we present two parallel approaches of the vascular development. An experimental validation is performed in Section IV. Conclusions and future works are sketched in Section V.

II. Model Description

The macroscopic part of the discussed model consists of two main elements: the tissue and the vascular network. The tissue is represented by a set of Macroscopic Functional Units (MFU) that are distributed inside the specified shape [2]. The vascular network is composed of vessels that provide a blood supply for the tissue. The microvascular part of the model is hidden in the MFUs and is responsible for the propagation of an MRI contrast agents in the tissue. The most important and original part of the work presented in this paper concerns the parallel algorithm of vascular development on the macroscopic level. Therefore, the main features of this part of the model are detailed here. For a detailed survey of the complete model, we refer the reader to our previous papers, e.g. [2], [3], [4].

The model expresses the specificity of the liver, although most of its features are not linked with any specific organ. The liver is strongly vascularized and stands out from other vital organs by the unique organization of its vascular network that consists of three vascular trees [11]: hepatic arteries, portal veins and hepatic veins. In the model [3], each vascular tree is composed of vessels that can divide creating bifurcations. Each vessel segment (part of the vessel between two successive bifurcations) is represented by an ideal, rigid tube with a fixed radius, wall thickness, length and position. The model distinguishes vessels larger than capillaries, while the capillaries themselves are hidden in the MFUs. Based on a morphometrical investigation dealing with bigger vessels, e.g. conducted by Zamir [12], each vascular tree forms a binary tree. As a result, vessel intersections (anastomosis), which occur particularly among vessels with very small radii or in pathological situations, are not taken into account. Blood is transferred from hepatic arteries and portal veins to hepatic veins through MFUs.

The vessels' parameters (e.g. pressure, radius) are calculated according to two basic physical laws. Firstly, at each bifurcation the law of matter conservation must be observed, i.e. the quantities of blood entering and leaving a bifurcation have to be equal. Second constraint deals with the decreasing vessel radii in the vascular tree, creating a relationship between the radius of a vessel and radii of its two descendants. Moreover, blood is considered as a Newtonian fluid (with constant viscosity) [13] and its flow is modeled as a laminar flow induced by the pressure difference between the two extremities of a vessel, Poiseuille's law is in effect.

An MFU is a small, fixed size part of tissue and has been assigned a class that determines most of its structural and functional properties (e.g. size, probability of mitosis and necrosis) and also physiological features (e.g. blood pressures, blood flow rate) [2]. In order to

introduce a natural variability, certain parameters (like blood flow rate) are described by given statistical distributions. Several classes of MFUs can be defined. The MFU class can be changed over time, which makes it possible to simulate the evolution of a disease (e.g. from benign nodule to malignant tumor).

An adult organ is obtained in a vascular development process. The simulation starts with an organ whose size is a fraction of the adult one. In discrete time moments (called cycles), the organ enlarges its size (growth phase) until it reaches its full, mature form. The relative positions of MFUs remain unchanged but distances between them are increased, leading to the appearance of empty spaces. These spaces are filled with new MFUs in consecutive subcycles. In each subcycle, an MFU can divide and give birth to a new MFU of the same class (mitosis process) or die (necrosis process). During the growth phase (mitosis supremacy over necrosis [14]) the increasing needs of the growing tissue induce the development of a vascular network (new vessels appear) that is responsible for blood delivery. In each subcycle, the processes of mitosis and necrosis are repeated, while a new cycle starts only when the current organ shape is totally filled with MFUs.

New MFUs that appear during the mitosis are initially not perfused by the existing vascular system. For each new MFU a fixed number of the closest/candidate vessels (in each tree) is found. Then each candidate vessel creates a new bifurcation that temporarily perfuses the MFU. The spatial position of the bifurcation point is controlled by local minimization of additional blood volume needed for the MFU perfusion. Only one vessel from each tree can finally be designated to perfuse the new MFU. Therefore, the algorithm creates all possible combinations of candidate vessels (a single combination consists of one vessel from each tree). Firstly, the algorithm rejects those combinations that could introduce intersections between vessels (from the same tree or from two different trees, e.g. between arteries and veins). Finally, the combination with the lowest sum of volumes is chosen to permanently perfuse the MFU.

After the reproduction (i.e. mitosis and perfusion processes), comes a degeneration phase. At this step of the algorithm, some of the MFUs can die (necrosis process) and then all the vessels supplying these MFUs retract and disappear (retraction process). Next, the algorithm goes back to the reproduction phase again.

III. Parallel Vascular Modeling

In the sequential algorithm of vascular growth, all new MFUs are perfused one by one. For each new MFU several temporary bifurcations have to be created and tested. It requires many calculations to fulfill all constraints (i.e. physical and physiological laws). As a result, the perfusion process is the time dominant operation in the organ growth simulation. Based on profiling results (execution times of specific methods), we estimated that it generally consumes around 70-95% of the total CPU time needed to develop an adult organ (time estimation of other processes: mitosis phase around 2-13%, degeneration phase around 3-19%). Therefore, in both parallel approaches, presented below, we concentrate mainly on the perfusion phase.

A. Distributed Memory Approach

At the beginning, the general idea of the approach is described. Then we present in more detail a parallel perfusion algorithm and mechanisms used to ensure an optimal load balancing.

The solution is based on a message passing paradigm and, in consequence, interactions between processors are accomplished by sending and receiving messages [16]. Its general diagram is presented in Fig. 1. Each processor/node during the whole simulation has its

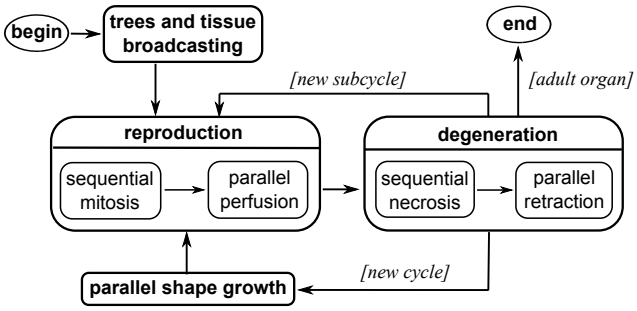


Fig. 1. The general diagram of parallel vascular growth algorithm that is based on a message passing paradigm (distributed memory approach).

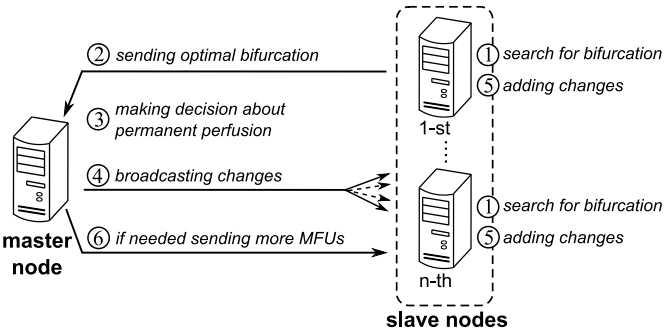


Fig. 2. The diagram of parallel perfusion process. Slave nodes attempt to find optimal bifurcations in a parallel way, while the master node is responsible for making decision about permanent perfusions and broadcasting changes.

own copy of vascular trees and tissue. Therefore, at the beginning, the master/managing processor broadcasts the whole initial organ to ensure that all the nodes possess the same starting information. After that, cycles and subcycles are iterated. Each subcycle starts with a sequential mitosis during which a list of new MFUs is created. Next, the parallel perfusion is carried out. In comparison to its sequential version, here the managing node does not make any attempt to find candidate vessels and bifurcation points but instead it spreads these tasks over slave/computing processors [15].

After the reproduction process, the degeneration phase follows. At the managing node, the sequential necrosis is carried out and next, the master processor broadcasts to all other processors information about the MFUs that have to be removed. The entire algorithm of retraction is performed at each node. The profiling results showed that the time needed for that part of the algorithm can be neglected, as it is very short in comparison to the perfusion time.

After the degeneration process, if a new subcycle is needed (current organ shape is not totally filled with MFUs), the process starts again with the mitosis. In the other case, the organ shape is enlarged. At each processor the same operations are carried out concurrently and only after they are all completed, the algorithm returns to the mitosis process. The algorithm ends when the organ reaches its adult size.

1) Parallel Perfusion Algorithm: After the sequential mitosis, the master node spreads new MFUs over slave nodes. When a computational node receives the message with several MFUs, it attempts to find the optimal bifurcation points to perfuse these new tissue elements (Fig. 2). Each time, when the search ends successfully, the computational node does not perfuse permanently the new MFU, but sends the parameters of the optimal bifurcation to the managing node. Next, if there are any queued message with permanent changes in vascular trees sent by the managing node, the calculating node applies these changes and continues to perform its remaining tasks.

The master node is responsible for managing the perfusion pro-

cess. When it receives a message with an optimal bifurcation of one of the new MFUs, it makes a decision about the permanent perfusion. Because of communication latency and independent work of slave processors, vascular networks at individuals nodes (both at computational ones and managing one) can be slightly different (trees' nonuniformity). Therefore, the master processor searches, in its vasculature, for the vessels that are proposed by the computational node to perfuse the considered MFU. If the processor cannot find at least one of these vessels, then the MFU is rejected. Such a situation simply means that such a vessel (or vessels) was earlier used to perfuse another tissue element and was replaced by new vascular structures (local modification due to the creation of a new bifurcation). The rejected MFU leaves an empty space in the tissue for other new MFUs that can appear during the next subcycles. But in the other case, the MFU is permanently connected to the vascular system. Subsequently, the managing node broadcasts all organ changes (related with the new MFU) to the slave processors and waits for next messages.

2) Load Balancing: The amount of work required to find candidate vessels and optimal bifurcation points can vary for different MFUs. Moreover, it is impossible to approximate the time needed to perform each task before the work distribution. As a result, due to the spreading all new MFUs, immediately after the mitosis process, some of the slave processors can be idle after performing all their jobs, while others have tasks queuing for execution. Therefore, an algorithm that aims to assure the optimal load balancing [17] (fair distribution of computations) between calculating nodes is introduced. In this algorithm, all jobs are not necessarily divided immediately after the mitosis process. Instead, the master node may keep a fixed quantity of new MFUs that are assigned to slave nodes only on demand (operation 6 in Fig. 2). When a slave node finishes its assigned tasks, it sends to the master node a message with a request to get more job. Such a solution is able to detect and handle load imbalances dynamically.

Almost all messages are passed in a non-blocking fashion and therefore processors do not lose time waiting. Nevertheless, such a solution requires more effort from the managing node to decide about permanent perfusions. First of all, the managing node has to collect information about the changes in vascular trees that are sent to individual slave nodes. When a slave node introduces and then confirms the changes in vascular trees, this information has to be refreshed (the managing node marks these changes as delivered and applied).

In order to provide a still more efficient solution, a load balancing of the master processor was also taken into account. When this node is idle, it can also perform calculations related to finding parameters of optimal bifurcations (i.e. the same as slave nodes).

B. Shared Memory Approach

The second approach we propose is based on the assumption that all processors operate independently but share the same memory resources. As a result, there is no need to take care of data communication between processors, in contrast to the aforementioned solution.

In this solution, we spread calculations over processors during the perfusion process. The remaining algorithm phases (i.e. mitosis, degeneration and shape growth) are carried out sequentially by one processor. Fig. 3 presents the general scheme of parallelization. There are three algorithm blocks (i.e. search for optimal bifurcations, connection and disconnection in the case of crossing vessels) during which individual processing units are responsible for calculations concerning one of the three vascular trees (i.e. hepatic arteries, portal veins or hepatic veins). There is also a possibility to assign more than one vascular tree to one processing unit, for example when the number of processing units is smaller than the number of vascular trees.

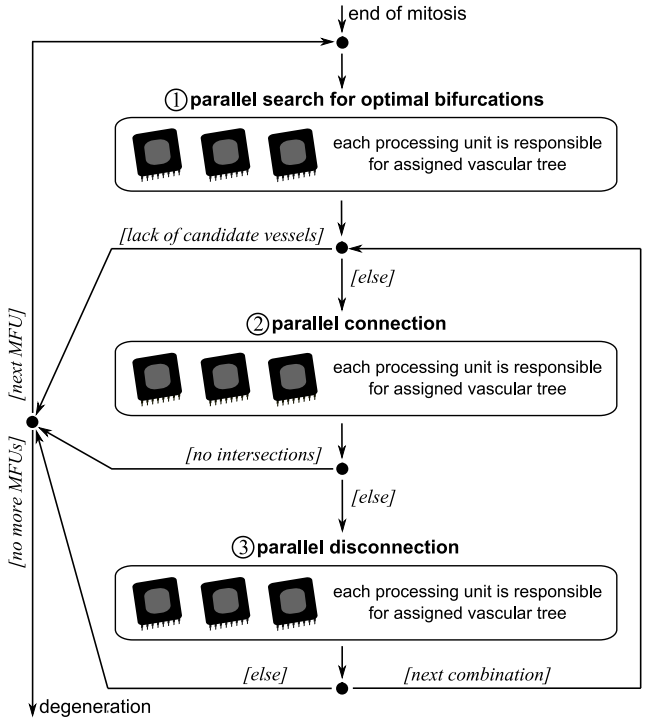
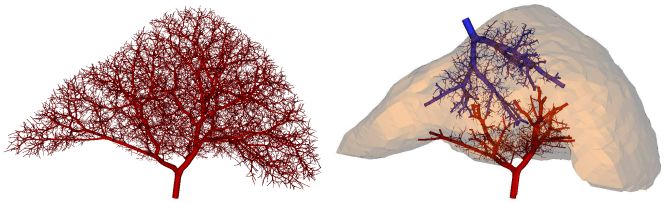


Fig. 3. Flowchart of the parallel perfusion algorithm of vascular growth designed for shared memory environments. Computations are performed in a parallel way in three algorithm blocks: search for optimal bifurcations, connection and disconnection.

After the sequential mitosis, the vascular system tries to perfuse each new MFU iteratively. At the beginning, the parallel search for optimal bifurcations is carried out (operation 1 in Fig. 3). A single processor is responsible successively for: searching for the candidate vessels in the assigned vascular tree, creating the optimal temporary bifurcations with each found candidate vessel and recalculating vessels' characteristics. If at least one of the processing units does not find any candidate vessel to perfuse the new tissue element (because of crossing vessels in its own tree), the other processors are informed about this and they abandon their tasks. It means that in at least one vascular tree there is no possibility to connect the new MFU. Afterwards, if there are more new MFUs to perfuse, the next one is considered.

On the contrary (if in each vascular tree at least one candidate vessel is found), the algorithm creates all possible combinations of candidate vessels (a single combination consists of one vessel from each tree). Next, the combinations are sorted according to increasing volume and processed. The MFU is connected in a parallel way to all vascular trees (operation 2 in Fig. 3) based on the information from the first tested combination. Then, possible intersections between the perfusing vessels are searched. If none of them is detected, the MFU is marked as permanently perfused. However, if any intersection is found, the MFU is disconnected concurrently from each vascular tree (operation 3 in Fig. 3) and the following combination (with the volume just superior) is taken into account. When all MFUs are processed, the degeneration phase begins.



(a) (b)

Fig. 4. Visualization of the simulation results - adult liver with about 50000 MFUs and 300000 vessel segments: (a) portal veins, (b) main hepatic arteries, portal veins and hepatic veins within liver shape.

IV. Experimental Validation

We checked the behavior of the algorithms starting from small size configurations (about 1000 MFUs) and ending with large configurations (about 50000 MFUs and consequently about 300000 vessel segments). Typical physiological parameters of the hepatic vascular network were used [3]. The performance of parallel implementations was mainly evaluated using the speedup (ratio of the time necessary to solve a problem sequentially to the time required by its parallel version [17]). Fig. 4 shows a visualization of one of the simulated hepatic vascular networks.

A. Distributed Memory Approach

The solution was implemented in the MVAPICH C++ version 0.9.5 which is the MPI [16] implementation over Infiniband network. For the performance analysis we used the Multi-Processing Environment (MPE) library and the graphical visualization tool Jumpshot-4 [16]. The experiments were performed on a cluster of sixteen SMP servers (two 64-bit Xeon 3.2GHz CPUs, 2MB L2 cache, 2GB of RAM) running Linux 2.6 and connected by an 10Gb/s Infiniband network. The GCC 3.4.6 compiler was used.

Fig. 5(a) presents the obtained mean speedup. It is clearly visible that the parallel algorithm significantly decreases the computation time. In practice, it means that the typical time needed to simulate an organ with 50000 MFUs and consequently 300000 vessel segments can be reduced from 21 hours (with a single processor machine) to 2 hours (with 16 processors).

Fig. 5(b) shows how the percentage of kept MFUs (that are sent on demand) influences the simulation time. If not all MFUs are spread immediately after the mitosis process, the algorithm can dynamically increase the load of idle processors by sending the remaining tasks. However, we can see that it is difficult to choose one common value (percentage of MFUs) that gives the best gain in time for a different number of processors. On the other hand, it is visible that values higher than 90% can increase the simulation time. Finally, we suggest that any value within the range from 30% to 70% is acceptable.

Moreover, based on the results presented in Fig. 5(c), it is clearly visible that in the case of small number of processors (i.e. smaller than 4) if the master node, besides managing, performs the same calculations as slave processors the simulation time can be reduced. On the other hand, i.e. the number of processors is bigger than 3, we should not arrange any additional job to the master node.

Furthermore, based on a fundamental formula called Amdahl's law [18], the maximum attainable speedup was thoroughly examined. Considering that the parallelized code consumes around 95% of the total CPU time during its serial execution (experimentally measured), the upper bound speedup is equal to approximately 9 with 16 processors. In practice, the obtained value was slightly lower because of an overhead due to communication and synchronization. Anyway, in the presented method, we are able to obtain even better speedup (i.e. about 10, see Fig. 5(a)) due to the introduction of an algorithm of periodical memory reallocations. It assumes that additionally, at the beginning of each subcycle, slave nodes rebuild their vascular structures in order to assure that these structures are represented in the operating memory by continuous memory blocks. Obviously these operations take some time but the performance analysis showed that exactly in the same time the mitosis process is being carried out at the master node. This mechanism accelerates the code execution related to the optimal bifurcation search algorithm.

B. Shared Memory Approach

The solution was implemented in C++ and the OpenMP Application Program Interface [19] that supports multi-platform shared-memory parallel programming. The experiments were carried out on two available multi-core computers. The hardware specifications and results are summarized in Table I. It can be noticed that very good speedups are obtained. However, we see that the second machine provides a slightly lower acceleration despite the fact that both computers consist of very similar hardware and work on the same operating systems. We speculate that this difference can be caused by using two different compilers. It is often emphasized that Intel compilers are especially tuned for its own hardware (e.g. Xeon processors), i.e. that they include advanced optimization features and provide highly optimized performance libraries for creating multi-threaded applications [20].

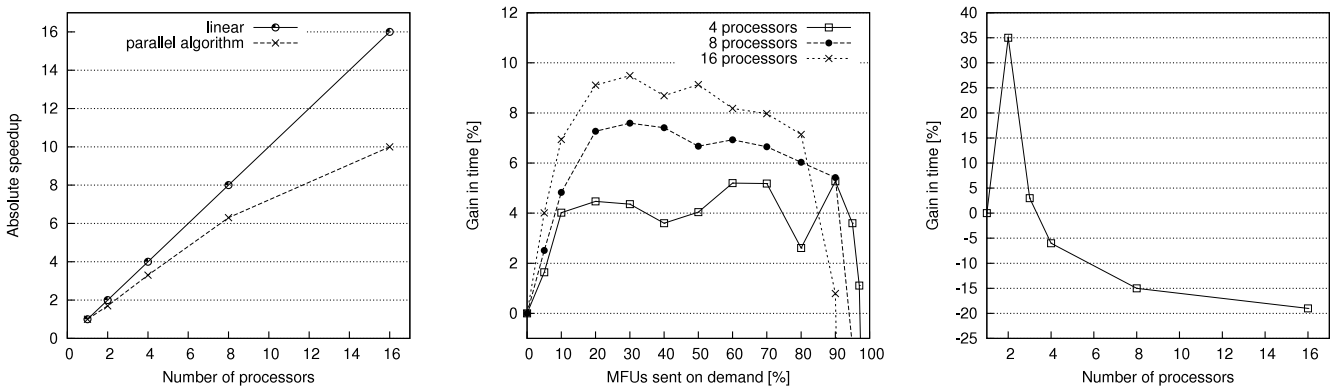
TABLE I
Mean, minimum and maximum values of speedup of shared memory algorithm on two multi-core computers

	Intel Xeon Quad-Core X5355, 2.66 GHz 8GB RAM Intel C++ Compiler 10.1			Intel Xeon Quad-Core X3220, 2.40 GHz 4GB RAM GCC 4.3.2		
number of used processor cores	speedup					
	mean	min	max	mean	min	max
2	1.47	1.38	1.50	1.35	1.32	1.40
3	2.57	2.26	3.12	2.30	2.20	2.43

Moreover, it is worth noting that, a speedup greater than the number of processors (called superlinear speedup [17]) can sometimes be observed (see maximum speedup value in Table I). It is due to the fact that the amount of work performed by serial and parallel algorithms can differ. In the parallel solution, when at least one of the processors is not able to find any candidate vessel to perfuse a new MFU in its own vascular tree, the other processors stop searching in their own vascular structures. On the other hand, in the serial algorithm, all vascular trees are checked step by step and, in the worst case, the tree with no candidate vessel for the new MFU can be the last to be verified.

V. Conclusion and Future Works

We have developed two algorithms of vascular modeling in a parallel environment: distributed and shared memory approaches. It is experimentally shown that both algorithms are able to significantly accelerate simulations. Although the shared memory solution allows us to gain a better efficiency (ratio between speedup and corresponding number of processors [17], e.g. in the case of a superlinear speedup), the algorithm implemented on a computing cluster is able to achieve a greater speedup, since it is not limited by the number of vascular trees. Considering the two strategies, we have access to a modeling framework able to increase the performance of the vascular development simulation on high-performance computing clusters as well as on low-cost multi-core/multi-processor commodity hardware.



(a) (b) (c)

Fig. 5. Performance of the distributed memory approach algorithm: (a) mean speedup, (b) influence of load balancing related to a fixed quantity of MFUs that are kept by the master node and sent only on demand, (c) influence of load balancing related to additional tasks that are assigned to the master processor.

In addition, one can treat the presented solutions as the first step to build the organ model in which physiological processes can be simulated both sequentially and concurrently. We suggest that such an approach can reflect the reality more accurately and consequently provide more information about real complex system behaviors.

The future work will deal with an hybrid solution able to take advantage of parallel machines that employ both shared and distributed memory architectures (mixed MPI-OpenMP implementation).

ACKNOWLEDGMENTS

We greatly appreciate Wojciech Kwedlo for the excellent hardware support with the computing cluster at Bialystok University of Technology.

REFERENCES

[1] J. Demongeot et al., "Multiscale modeling and imaging: The challenges of biocomplexity," *Proc. IEEE*, vol. 91, pp. 1723-1737, Oct. 2003.

[2] M. Kretowski, Y. Rolland, J. Bezy-Wendling, and J.-L. Coatrieux, "Physiologically based modeling for medical image analysis: application to 3D vascular networks and CT scan angiography," *IEEE Trans. on Medical Imaging*, vol. 22, no. 2, pp. 248-257, Feb. 2003.

[3] M. Kretowski, J. Bezy-Wendling, and P. Coupe, "Simulation of biphasic CT findings in hepatic cellular carcinoma by a two-level physiological model," *IEEE Trans. on Biomedical Engineering*, vol. 54, no. 3, pp. 538-542, Mar. 2007.

[4] M. Mescam, M. Kretowski, and J. Bezy-Wendling, "Multiscale model of liver DCE-MRI towards a better understanding of tumor complexity," *IEEE Trans. on Medical Imaging*, vol. 29, no. 3, pp. 699-707, Mar. 2010.

[5] T. Rauber and G. Runger, *Parallel Programming: For Multicore and Cluster Systems.* New York: Springer-Verlag, 2010, ch. 2.

[6] D. T. Shima and Ch. Ruhrberg, "Angiogenesis" in *The Molecular Biology of Cancer,* S. Pelengaris and M. Khan, Eds. Oxford: Blackwell, 2006, pp. 411–423.

[7] M. Zamir, "Arterial branching within the confines of fractal L-system formalism," *J. Gen. Physiol.*, vol. 118, pp. 267-275, Sep. 2001.

[8] W. Schreiner et al., "Optimized arterial trees supplying hollow organs," *Medical Engineering & Physics*, vol. 28, no. 5, pp. 416-429, June 2006.

[9] D. Szczerba and G. Szekely, "Macroscopic modeling of vascular systems," in *Lecture Notes in Computer Science vol. 2489*, Heidelberg: Springer-Verlag, Jan. 2002, pp. 284-292.

[10] D. Szczerba and G. Szekely, "Computational model of flow-tissue interactions in intussusceptive angiogenesis," *Journal of Theoretical Biology*, vol. 234, no. 1, pp. 87-97, May 2005.

[11] S. Sherlock and J. Dooley, *Diseases of the Liver and Biliary System*. Malden, MA: Blackwell Science, 2002.

[12] M. Zamir and H. Chee, "Branching characteristics of human coronary arteries," *Can. J. Physiol. Pharmacol.*, vol. 64, pp. 661-668, June 1986.

[13] Y. C. Fung, *Biomechanics: Motion, Flow, Stress and Growth,* New York: Springer, 1990.

[14] D. O. Morgan, *Cell Cycle: Principles of Control*. London: New Science Press, 2007.

[15] K. Jurczuk, M. Kretowski, and J. Bezy-Wendling, "Vascular network modeling - improved parallel implementation on computing cluster," in *Lecture Notes in Computer Science* vol. 6067, Heidelberg: Springer-Verlag, July 2010, pp. 289-298.

[16] P. Pacheco, *Parallel Programming with MPI.* San Francisco, CA: Morgan Kaufmann Publishers, 1997.

[17] A. Grama, G. Karypis, V. Kumar, and A. Gupta, *Introduction to Parallel Computing.* Boston, MA: Addison-Wesley, 2003.

[18] G. M. Amdahl, "Validity of the single processor approach to achieving large scale computing capabilities", in *Proc. AFIPS Conf.*, Atlantic City, Apr. 1967, vol. 30, pp. 483-485.

[19] B. Chapman, G. Jost, R. van der Pas, and D. J. Kuck, *Using OpenMP: Portable Shared Memory Parallel Programming.* Cambridge: MIT Press, 2007.

[20] Intel C++ Compiler 10.1 for Linux (Product In-Depth). [Online]. Available: http://software.intel.com/en-us/intel-compilers/