Full Length Article

International Journal of HIGH PERFORMANCE COMPUTING APPLICATIONS

The International Journal of High Performance Computing Applications
1–16

DOI: 10.1177/1094342016677586
journals.sagepub.com/home/hpc
SAGE

# GPU-based computational modeling of magnetic resonance imaging of vascular structures

**Krzysztof Jurczuk[1], Marek Kretowski[1] and Johanne Bezy–Wendling[2,3]**

**Abstract**

Magnetic resonance imaging (MRI) is one of the most important diagnostic tools in modern medicine. Since it is a high-cost and highly-complex imaging modality, computational models are frequently built to enhance its understanding as well as to support further development. However, such models often have to be simplified to complete simulations in a reasonable time. Thus, the simulations with high spatial/temporal resolutions, with any motion consideration (like blood flow) and/or with 3D objects usually call for using parallel computing environments.

In this paper, we propose to use graphics processing units (GPUs) for fast simulations of MRI of vascular structures. We apply a CUDA environment which supports general purpose computation on GPU (GPGPU). The data decomposition strategy is applied and thus the parts of each virtual object are spread over the GPU cores. The GPU cores are responsible for calculating the influence of blood flow behavior and MRI events after successive time steps. In the proposed approach, different data layouts, memory access patterns, and other memory improvements are applied to efficiently exploit GPU resources. Computational performance is thoroughly validated for various vascular structures and different NVIDIA GPUs. Results show that MRI simulations can be accelerated significantly thanks to GPGPU. The proposed GPU-based approach may be easily adopted in the modeling of other flow related phenomena like perfusion, diffusion or transport of contrast agents.

**Keywords**

Computational modeling, graphics processing unit, magnetic resonance imaging, parallel computing, vascular structures

## 1 Introduction

The rapid development of computers and programming techniques that started in the second half of the 20th century led to tremendous changes in conducting research. Instead of building physical models, a lot of experiments are currently performed in a virtual world with the use of computers and computational models (Zeigler, 2000). Such computer simulations (called in-silico experiments) have a lot of advantages over *in vivo* and *in vitro* experiments. For example, they are usually cheaper and more effortless, actual systems (like the human body) are not exposed to damage, and a broader range of problems can be studied since several simulations can run simultaneously.

The paper concerns the computational modeling of magnetic resonance imaging (MRI) (Westbrook et al., 2011) which is one of the most important diagnostic tools in modern medicine. However, scientists and physicians still have restricted access to MRI equipment. This is partly due to the high cost connected with imaging equipment and also the necessity to employ skilled personnel to operate and maintain MRI scanners. MRI simulations can effectively eliminate the high cost related to the skilled personnel and physical devices (Benoit–Cattin et al., 2005). Moreover, simulations can be performed without the need for patients to participate and thus are not limited by the examination duration.

In this paper, we focus on MRI of vascular structures. Although MRI is known as a highly detailed three-dimensional (3D) imaging modality, there are still a lot of difficulties in vascular image formation and

[1]Faculty of Computer Science, Bialystok University of Technology, Poland
[2]INSERM, Rennes, France
[3]University of Rennes 1, Rennes, France

**Corresponding author:**
Krzysztof Jurczuk, Faculty of Computer Science, Bialystok University of Technology, Wiejska 45a, 15-351 Bialystok, Poland.
Email: k.jurczuk@pb.edu.pl

interpretation (Van Der Graaf et al., 2014; Rahimi et al., 2015). Imaging of blood flow areas is crucial since vascular diseases are the cause of large mortality rates (Aiyagari and Gorelick, 2011; Garin et al., 2013). On the one hand, pathology detection and characterization can be improved by using the intrinsic motion sensitivity of MRI. For example, blood flow-related signal diminution can be identified in zones of abnormal vessel shapes (like an aneurysm) that can have serious consequences and even cause death (Lasheras, 2007). On the other hand, a flow during MRI acquisition can give arise to various image artifacts. They introduce additional difficulties in image analysis which can lead to image misinterpretation and inappropriate patient treatment. Hence, the understanding of magnetic resonance (MR) flow image formation is of importance, due to clinical assessment of the disease as well as problems with artifacts.

Computational models have often been used to understand and/or explain MRI processes that are unclear, complex or difficult to observe. For instance, the models can help to study the relationships between vascular geometry changes and hemodynamic factors in silico (Dyverfeldt et al., 2009). The connection between fluid flow and image appearance can also be investigated (Lorthois et al., 2005). Turning on/off particular physical phenomena and the evaluation of various combinations of MRI equipment parameters are often time consuming and in some cases even impossible. On the other hand, in computational models it is far easier to switch on/off their components and to study the contribution of different factors (each factor separately or all factors together). Therefore, such modeling can certainly contribute in the understanding of pathological processes and improving MRI sequence design. Finally, controlled simulation experiments are also a valuable way of educating.

The in-silico modeling of MR flow imaging is not a trivial task. It requires the integration of many technological processes, phenomenon and factors linked to anatomy, physiology, hemodynamics, and imaging technology, in one computational model (Jurczuk et al., 2014). Besides addressing questions about the integrative model quality and the level of its detail, the challenge of such an approach lies also in the demand for high performance computing which is required to perform simulations within a reasonable period of time. Simulations of each physical phenomenon itself as well as interactions between them require many calculations. Large vascular simulations are vital to correctly investigate internal processes in human bodies (Grinberg et al., 2011) and the computational needs grow fast with the size of vascular structures.

There have been many proposed approaches in the modeling of MR flow imaging (Lorthois et al., 2005; Dyverfeldt et al., 2009; Marshall, 2010; Jurczuk et al., 2013), to name a few. The long simulation time was always one of the main factors limiting the extension of these models to a 3D version or to study complex vascular networks. Thus, it seems that further progress in computational modeling of MRI does not only depend on sophisticated equations but also on the development of parallel architectures and the algorithms. In our recent study (Jurczuk et al., 2014), we applied cluster computing in the modeling of MR flow imaging, which allowed us to investigate more complex vascular structures.

In this paper, we propose to use graphics processing units (GPUs) in the modeling of MR flow imaging. GPUs of modern graphics cards are equipped with hundreds or even thousands of small, energy efficient computing units (GPU cores) for handling multiple tasks in parallel and managing workloads efficiently (Wilt, 2013). Thus, general purpose computation on GPU (GPGPU) has gained in popularity (Yuen et al., 2013).

Our motivation is to exploit those GPU's computational resources and bring the possibility to perform fast MRI simulations on a single workstation. This way, complex vascular structure simulations can become independent of computer clusters that might be expensive, maintenance demanding and are not always accessible. In addition, modern GPUs often provide lower price/performance ratio than computer clusters. What is also important is that the parallel approach proposed in this paper may be easily applied in the modeling of other flow related imaging like perfusion, diffusion or contrast agents transport in MRI. As far as we know, the proposed computational model is the first GPU-based approach to the simulation of MRI of vascular structures. Our initial efforts to create the presented solution are described in the conference paper (Jurczuk et al., 2016).

The rest of the paper is organized as follows. Section 2 gives a brief introduction to MRI, then it presents the in-silico model of interest. The section ends with a related works part. In Section 3 we propose a GPU-based approach to in-silico modeling of MRI of vascular structures. Section 4 presents the performance evaluation of the proposed approach. In the last section conclusions and possible future works are sketched.

## 2 Computational model description

This section contains a brief introduction of MRI principals, followed by a description of the computational model and a related works part.

### 2.1 Magnetic resonance imaging

MRI is one of the most important tomography methods in medicine (Westbrook et al., 2011). Due to its

numerous advantages and applications it has revolutionized diagnostic imaging in medical science. MRI produces sharp high-resolution images. Moreover, it provides a unique contrast between soft tissues, which is generally superior to that of computed tomography (CT). So far, it seems to have no side effects related to radiation exposure, especially in comparison to CT or positron emission tomography (PET). Its clinical applications are still expanding rapidly as hardware and imaging technology overcome successive limitations. Some of them include functional MRI (fMRI), MR angiography (MRA), diffusion MRI (dMRI) or MR spectroscopy (MRS).

MRI relies on the intrinsic magnetic properties of body tissues in an external magnetic field (Kuperman, 2000). Hydrogen protons are usually used because of their high natural abundance in body tissues (in water and fat). When a patient is put in an MRI scanner equipped with a strong magnet, the body is temporarily magnetized. Then, an oscillating radio frequency (RF) pulse is additionally transmitted with an appropriate frequency to fit the frequency of the magnetized particles. As a result, the particles can absorb this additional energy. This process is known as excitation and it can cause the appearance of the resultant magnetization in the plane perpendicular to the main magnetic field.

After the RF pulse is turned off, the particles lose the absorbed energy and tend to realign with the main magnetic field. The process of particles returning to the equilibrium is called relaxation. Two independent relaxations take place. The amount of longitudinal magnetization gradually increases due to giving up the absorbed energy (longitudinal relaxation). The rate of longitudinal magnetization recovery is an exponential process with the time constant $T_1$: $1 - \exp(-t/T_1)$. At the same time, but independently, the value of the transverse magnetization decreases since the nuclei lose coherency due to dephasing (transverse relaxation). The decay of the transverse magnetization is an exponential process with the time constant $T_2$: $\exp(-t/T_2)$.

According to the Faraday's law of electromagnetic induction, if a receiver coil or any conductive loop is placed in the area of a changing magnetic field, a voltage is induced in this coil (Bernstein et al., 2004). Since the magnetic moments of the particles rotate/spin, they can induce a voltage in a receiver coil. Such induced current is the MR signal that is measured. It is then recorded and can be further processed. The spinning magnetic moments of the particles are often called spins.

The receiver coils are used during the relaxation, since different tissues return to equilibrium at different speeds. The number of the particles participating in MR, so-called proton density, also influences the received signal. The acquired signal is collected in a k-space matrix. The signal from a single excitation is the sum of all subsignals coming from all excited particles. Thus, the magnetic gradients are used to encode spatial directions.

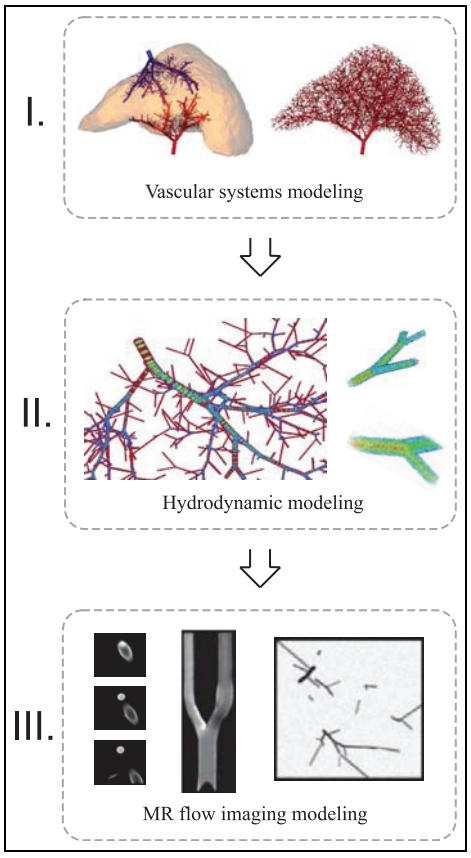


**Figure 1.** Three-component MRI model overview. The model allows us to generate vascular structures (I), then simulate flow behavior (II) and finally reproduce MRI processes (III).

The signal acquired after one excitation is insufficient to encode the nuclei in all directions. It is necessary to apply RF pulses multiple times with various gradients to spatially encode the signal of a 3D area. Then, signal series are collected and the image can be reconstructed based on them. The fast Fourier transform (FFT) (Aibinu, 2008) is used to transform the k-space matrix to the desired image.

MRI events (excitation, signal acquisition, spatial encoding, etc.) are spread over time. The temporal arrangement of the events (to form an image) is called a pulse sequence. There are two basic pulse sequences: spin echo (SE) and gradient echo (GE). Other more advanced pulse sequences are based on these two and additionally include some improvements e.g. in terms of time to obtain an image or better image contrast.

## 2.2 Computational model

In our previous research, we developed a three-component model of MR flow imaging (Jurczuk et al., 2014) (Figure 1). The first component is used for

generating the vascular structures based on physiological and hemodynamic parameters (Kretowski et al., 2003). The second one allows flow simulations to be performed in the generated vascular structures (Jurczuk et al., 2013). The last component makes use of the generated vascular structures and flow characteristics to simulate MRI of vascular structures. Since, in this paper, the algorithm of imaging is parallelized, only the last model component is described.

The 3D imaged area (object) is divided into cubic elements (Jurczuk et al., 2013). For each cubic element basic MR parameters (proton density, relaxation times), determined by the represented part of a tissue, are assigned. In addition, each cubic element contains hydrodynamic parameters (generated by the flow model) that are represented by the dimensionless lattice velocity $\mathbf{u} = u_x\hat{\mathbf{i}} + u_y\hat{\mathbf{j}} + u_z\hat{\mathbf{k}}$, where $\hat{\mathbf{i}}$, $\hat{\mathbf{j}}$, $\hat{\mathbf{k}}$ are unit vectors in $x$, $y$, $z$ directions. The flow velocity for stationary tissue structures (e.g. vessel walls, bones, parenchyma) equals zero.

Imaging simulation is divided into short time periods called time steps $\Delta t$. After each time step, local magnetizations of all cubic elements are modified taking into account both the flow influence ($\Delta\mathbf{M}_F$) and MRI events ($\mathbf{A}_{\mathrm{MRI}}$)

$$\begin{aligned}&\mathbf{M}(\mathbf{r}, t + \Delta t)\\ &= \mathbf{A}_{\mathrm{MRI}}(\mathbf{r}, \Delta t)[\mathbf{M}(\mathbf{r}, t) + \Delta\mathbf{M}_F(\mathbf{r}, \Delta t)]\end{aligned} \tag{1}$$

where $\mathbf{M}$ is the magnetization of the cubic element at spatial position $\mathbf{r}$.

First, the flow influence is computed (see top part of Figure 2). In each cubic element the magnetization fractions are propagated to the neighboring cubic elements (see black rectangles labeled by "a", "b" and "c"). This way, parts of magnetization can leave some cubic elements. At the same time, the magnetization fractions that leave some cubic elements enter neighboring cubic elements. The mean magnetization changes during a time step $\Delta t$ for a cubic element at position $\boldsymbol{r}$ are modeled as follows

$$\Delta\mathbf{M}_F(\mathbf{r}, \Delta t) = \Delta\mathbf{M}_{\mathrm{IN}}(\mathbf{r}, \Delta t) - \Delta\mathbf{M}_{\mathrm{OUT}}(\mathbf{r}, \Delta t) \tag{2}$$

where $\Delta\mathbf{M}_{\mathrm{IN}}$ denotes the inflow magnetization, while $\Delta\mathbf{M}_{\mathrm{OUT}}$ is the outflow magnetization.

The $\Delta\mathbf{M}_{\mathrm{OUT}}$ value is calculated based on the flow properties and the local magnetization of the considered cubic element

$$\begin{aligned}&\Delta\mathbf{M}_{\mathrm{OUT}}(\mathbf{r}, \Delta t) = \mathbf{M}(\mathbf{r}, t)[|u_x(\mathbf{r})||u_y(\mathbf{r})|\\ &+ |u_x(\mathbf{r})|(1 - |u_y(\mathbf{r})|) + (1 - |u_x(\mathbf{r})|)|u_y(\mathbf{r})|]\end{aligned} \tag{3}$$

Meanwhile, the fractions of magnetization entering into a cubic element are calculated with the use of the magnetizations of its neighboring elements as well as the flow properties of this cubic element, as follows

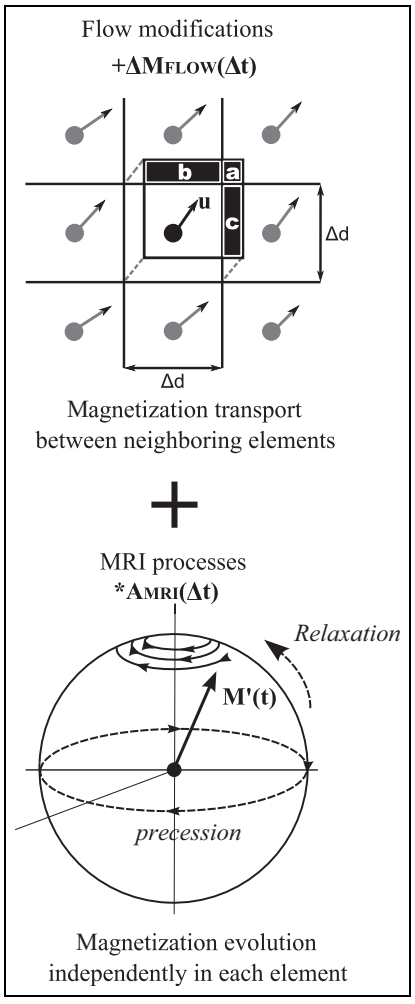


**Figure 2.** The connection of MRI and magnetization transport algorithms.

$$\begin{aligned}&\Delta\mathbf{M}_{\mathrm{IN}}(\mathbf{r}, \Delta t)\\ &= \mathbf{M}\left(\mathbf{r} - \Delta d\frac{u_x(\mathbf{r})}{|u_x(\mathbf{r})|}\hat{\mathbf{i}} - \Delta d\frac{u_y(\mathbf{r})}{|u_y(\mathbf{r})|}\hat{\mathbf{j}}, t\right)|u_x(\mathbf{r})||u_y(\mathbf{r})|\\ &+ \mathbf{M}\left(\mathbf{r} - \Delta d\frac{u_x(\mathbf{r})}{|u_x(\mathbf{r})|}\hat{\mathbf{i}}, t\right)|u_x(\mathbf{r})|(1 - |u_y(\mathbf{r})|)\\ &+ \mathbf{M}\left(\mathbf{r} - \Delta d\frac{u_y(\mathbf{r})}{|u_y(\mathbf{r})|}\hat{\mathbf{j}}, t\right)(1 - |u_x(\mathbf{r})|)|u_y(\mathbf{r})|\end{aligned} \tag{4}$$

In 3D modeling, each sum component in equations (3) and (4) is made up of two more cases in accordance with the $|u_z(\mathbf{r})|$ or $(1 - |u_z(\mathbf{r})|)$ term.

Later, the MRI influence is computed (see bottom part of Figure 2). It is modeled by the Bloch equation (Bloch et al., 1946). We apply its discrete time solution proposed by Bittoun et al. (1984). It uses the rotation matrices and exponential scaling to represent the response of spins' magnetization to magnetic field changes. Such an approach (used in many advanced

MRI simulators, e.g. SIMRI (Benoit–Cattin et al., 2005), ODIN (Jochimsen, 2006)) allows one to follow the variations of spins' magnetization during the whole MRI sequence without any integration. Based on it, in each cubic element, the following mathematical formulas are used to calculate the MRI influence

$$\begin{aligned}&\mathbf{A}_{\mathrm{MRI}}(\mathbf{r},\Delta t)\\&=\mathbf{E}_{\mathrm{RELAX}}(\mathbf{r},\Delta t)\mathbf{R}_z(\Theta_G)\mathbf{R}_z(\Theta_{IH})\mathbf{R}_{RF}(\mathbf{r},\Delta t)\end{aligned}\tag{5}$$

where $\mathbf{E}_{\mathrm{RELAX}}$ represents the relaxation phenomena

$$\begin{aligned}&\mathbf{E}_{\mathrm{RELAX}}(\mathbf{r},\Delta t)\\&=\mathrm{diag}\left[e^{\left(-\frac{\Delta t}{T2(\mathbf{r})}\right)},e^{\left(-\frac{\Delta t}{T2(\mathbf{r})}\right)},1-e^{\left(-\frac{\Delta t}{T1(\mathbf{r})}\right)}\right]\end{aligned}\tag{6}$$

$\mathbf{R}_z$ is the rotation matrix around the $z$-axis used to model the influence of the spatial encoding gradient $\mathbf{G}$ (rotation through angle $\Theta_G(\mathbf{r},\Delta t)=\mathbf{G}\cdot\mathbf{r}\gamma\Delta t$) and magnetic field inhomogeneities $\Delta B$ (rotation through angle $\Theta_{IH}(\mathbf{r},\Delta t)=\gamma\Delta B(\mathbf{r})\Delta t$)

$$\mathbf{R}_z(\theta)=\begin{bmatrix}\cos\theta & \sin\theta & 0\\ -\sin\theta & \cos\theta & 0\\ 0 & 0 & 1\end{bmatrix}\tag{7}$$

$\gamma$ is the gyromagnetic ratio and $\mathbf{R}_{RF}$ is the rotation matrix describing the influence of the RF pulse and the slice selection gradient (Jurczuk et al., 2013).

Based on Faraday's law of an electromagnetic induction, the MR signal coming from the imaged object at a time $t$ is expressed as a sum of the transverse magnetizations

$$\mathbf{S}(t)=\sum_{\mathbf{r}_o\in C}M_x(\mathbf{r}_o,t)\widehat{\mathbf{i}}+\sum_{\mathbf{r}_o\in C}M_y(\mathbf{r}_o,t)\widehat{\mathbf{j}}\tag{8}$$

where $C$ is the collection of all cubic elements of the imaged area. Each subsequent excitation is performed with a different phase encoding step and the acquired signals fill the successive matrix rows. The MR image is created by applying an FFT to the fully filled matrix.

In the model, the ideal spoiling of transverse magnetization is used. After each readout, the transverse magnetization of all cubic elements is set to zero. In addition, the so-called hard-pulse approximation is implemented to represent different RF pulse shapes (Bernstein et al., 2004). It allows the shaped RF pulses to be approximated by the sequence of short constant pulses of equal duration separated by periods of free precession, which perfectly suits the applied iterative approach of MRI and flow simulations.

By default, the time step for the modeling of MR flow imaging is equal to the time step from the flow modeling. If needed (for quick trial calculations or time optimization issues), this time step can be changed. Moreover, it may be different during various stages of imaging, e.g. shorter during a slice selection and longer after a signal acquisition up to the time of the next excitation when there is less change in magnetization, which can lead to significant computational savings in simulation studies. However, this time step cannot be longer than the shortest time needed by all the fluid to pass from one grid node to another. In the other case, it is possible that the magnetization fractions can go (jump) more than one grid node in a single time step.

As regards the grid resolution, a restriction arises because of the need to have an appropriate number of cubic elements in each image voxel. Due to the applied discrete-event solution of the Bloch equation, simulations are performed at discrete spatial locations and a continuous distribution of spins has to be imitated. Thus, the number of cubic elements (also called isochromats) in each image voxel has to be chosen carefully to generate a smooth image intensity.

## 2.3 Related works

There have been several proposed approaches in the modeling of MR flow imaging. They can be divided into two groups: Lagrangian-based (Jou et al., 1996; Marshall, 2010) or Eulerian-based (Jou and Saloner, 1998; Lorthois et al., 2005). In Lagrangian-based solutions, the flow pathlines are calculated first. Then, the temporal tracking of spin magnetizations along these pathlines is performed. While this approach is physically intuitive, it is known to be computationally expensive and not so efficient in complex geometries where particle tracking can fail (Marshall, 2010). Moreover, the algorithms for the compensation of regions of low particles density (e.g. close to vessel walls) might be needed since the densities of particles are not uniform over space, while in real fluids they are.

On the other hand, in Eulerian-based approaches a fixed grid of nodes is considered. Such a grid is used to represent the area under investigation. The magnetization values of the nearest spins are associated with each node. When the spins move, their magnetizations are transferred to neighboring nodes according to the fluid flow. Eulerian-based models inherently provide the uniform particle density and therefore, the aforementioned problems of low particles density are eliminated. The necessity to track magnetic particles along pathlines during imaging is also eliminated and consequently, simulations are often less time consuming. Nevertheless, the Eulerian-based approaches can hinder the modeling of the spatial-temporal interaction of flow with MRI events. Therefore, additional geometrical procedures had to be developed to take into account the displacement artifact (Nishimura et al., 1991), e.g. mesh transformation for mapping the physical plane into the MR image plane (Lorthois et al., 2005). As a result, other gradient arrangements, different RF pulses or 3D imaging may require additional mechanisms or some improvements in the existing algorithms.

The advantage of our approach (described in Section 2.2) stems from the coupling of the magnetization transport algorithm (2D/3D analytical operations) with the discrete-time solution of the Bloch equation. Hence, the Eulerian coordinate approach is retained and at the same time, flow-related artifacts are automatically taken into account. Such an approach allows us to closely follow the physical process of MRI, along with the automatic incorporation of flow-related artifacts. Thus, there is no need for geometrical procedures for mapping the physical plane to the image plane, such as a mesh transformation (Lorthois et al., 2005). Flow is automatically considered during most MRI events, e.g. during excitation, signal acquisition, and spatial encoding.

In our recent research, we extended this MR flow imaging model and investigated its MPI parallelization on a computer cluster (Jurczuk et al., 2014). The parallelization used the master-slave paradigm (Grama et al., 2003) together with the data decomposition strategy and managed to achieve up to 75 speedup with 128 CPU cores.

A GPU-based MRI simulator (Xanthis et al., 2014a) was also published recently. It allows only the stationary magnetization (without blood flow) to be investigated. The same research group extended their simulator to model various motions (Xanthis et al., 2014b). However, their extended solution still does not enable vascular structures to be taken into account as the magnetization transport algorithm is significantly simplified. Moreover, it uses the Lagrangian-based approach and it was tested only with a single straight tube.

To the best of our knowledge, there are no other studies in the literature about the simulation of MRI on vascular structures using the GPU-based approach yet. Our initial efforts to create the presented solution are described in the conference paper (Jurczuk et al., 2016). In comparison to this conference work, in this paper, we present an optimized version of the model and we provide new results and thorough performance analysis of the initial and the current implementations. The new contributions presented in this paper are as follows.

1. Optimized computational kernels (SoA-based memory organization (Strzodka, 2012), different padding mechanisms).
2. An in-depth analysis of the performance of our model concerning performance across various GPUs, both Kepler- and Maxwell-based ones, as well as concerning scalability, threads/blocks configurations, padding mechanisms, etc. Two additional GPUs are also added. It makes the model implementation more portable/optimized across different GPU devices.
3. A thorough analysis of the models performance in response to successive model improvements, from the initial and the current implementations (different memory layouts along with additional extensions like padding).
4. Profiling results of the CPU implementation as well as GPU-accelerated one are provided and analyzed.

## 3 GPU-accelerated approach

The most time consuming part of the MR flow imaging algorithm is the tracking of magnetizations $\mathbf{M}(\mathbf{r}, t)$ in time. A new magnetization value ($\mathbf{M}(\mathbf{r}, t + \Delta t)$) has to be calculated after each time step $\Delta t$. The time step has to be small enough (usually of the order of tens to hundreds of microseconds) to take into account the dynamic blood behavior appropriately.

Moreover, in each time step, both the blood flow and the influence of the MRI have to be considered in each cubic element at position $\mathbf{r}$. The size of cubic elements has to be small enough to provide the imitation of a continuous distribution of spins and, thus, the smooth image intensity changes. Its size is usually of the order of tens to hundreds of micrometers. The detailed performance analysis (using GNU gprof profiler (Von Hagen, 2006)) showed us that the calculation of $\mathbf{M}(\mathbf{r}, t + \Delta t)$ consumes, on average, more than 99% of the total CPU time that is required to obtain final MR images (see Table 1).

The general flowchart of our GPU-based approach is illustrated in Figure 3. One can observe that an MRI simulation is run in a sequential way on a CPU and the most time consuming operations concerning the evolution of magnetization values are delegated to a GPU. This way, the parallelization does not affect the behavior of the original sequential algorithm.

The algorithm starts with the initialization of the 3D object and the experiment parameters at a CPU (Figure 3). Then, they are sent to the GPU and saved in the allocated space in global memory. This CPU-GPU

**Table 1.** Fraction of total execution time devoted to particular actions of MRI modeling for a straight tube geometry filled completely by blood (the contribution of MRI/flow influence can different for other objects, depending on the number of cubic elements filled by fluid).

| Actions | Time fraction |
|---|---|
| 1. Object and experiment initialization | <0.01% |
| 2. Magnetization changes modeling: | 99.99% |
| - MRI influence | 42.20% |
| - Flow influence | 57.79% |
| 3. Image reconstruction (FFT) | <0.01% |
| 4. Others | <0.01% |

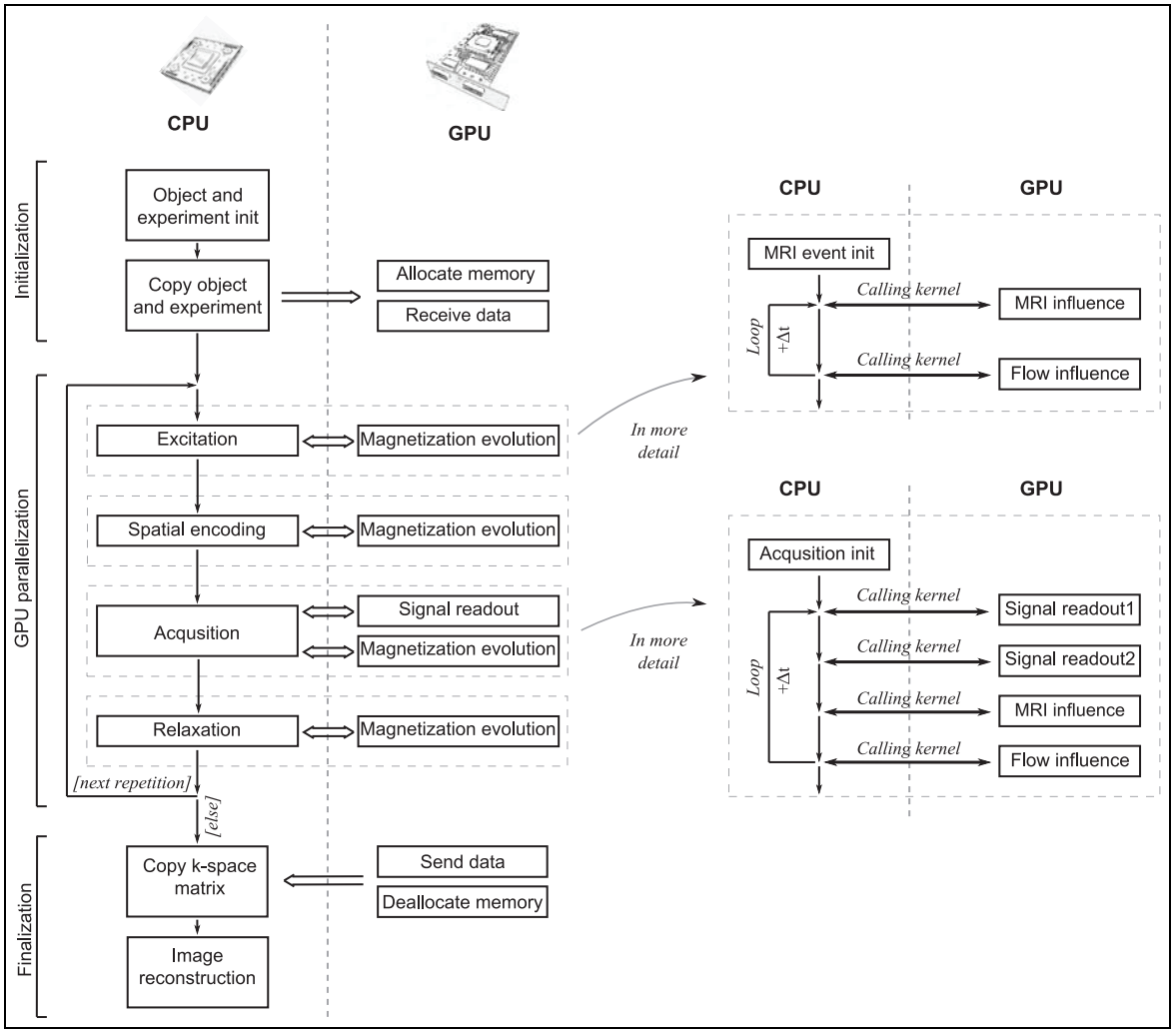


**Figure 3.** The general flowchart of the GPU-accelerated algorithm of MR flow imaging.

data transfer is performed only once before the MRI simulation and the data is kept on the GPU till the MRI experiment is finished. This way, data transfer is substantially reduced, especially for large objects, and each GPU thread has access to the data. Moreover, batching many small data transfers into a single larger one always performs more efficiently than executing each transfer separately (Cheng et al., 2014).

After the initialization, the MR image formation is started and here the GPU-based parallelization is applied. Each time when there is a need to compute new magnetization values, the CPU requests the GPU to perform calculations (see *calling kernel* in Figure 3). The data decomposition strategy is used. The 3D object is decomposed into multiple subdomains that are processed by GPU cores in parallel.

The following subsections describe in more detail the GPU-parallelized algorithm, along with the data decomposition strategy as well as additional optimizations that shorten the simulation time further.

## 3.1 GPU-parallelized magnetization evolution

The 3D object is decomposed into two levels (Figure 4). At first it is spread into parts (subsets of cubic elements) that are processed by different GPU blocks. We decided to use the 2D grid of blocks along the directions of the phase encoding gradients. Next, in each block, the cubic elements along the direction of frequency encoding gradient are spread further over the threads. By default, the number of blocks and threads are set to the number of cubic elements (object size) in the considered directions. If the object size exceeds the maximum number of threads/blocks, then the maximum value is used and a single thread/block processes more than one cubic element. The maximum numbers of threads/blocks are the algorithm parameters, however, they can also be hardware dependent.

Magnetization evolves in response to various MRI events (e.g. excitation, spatial encoding, relaxation) that are applied in appropriate order and with chosen parameters (see equation (5)). Each such an event is

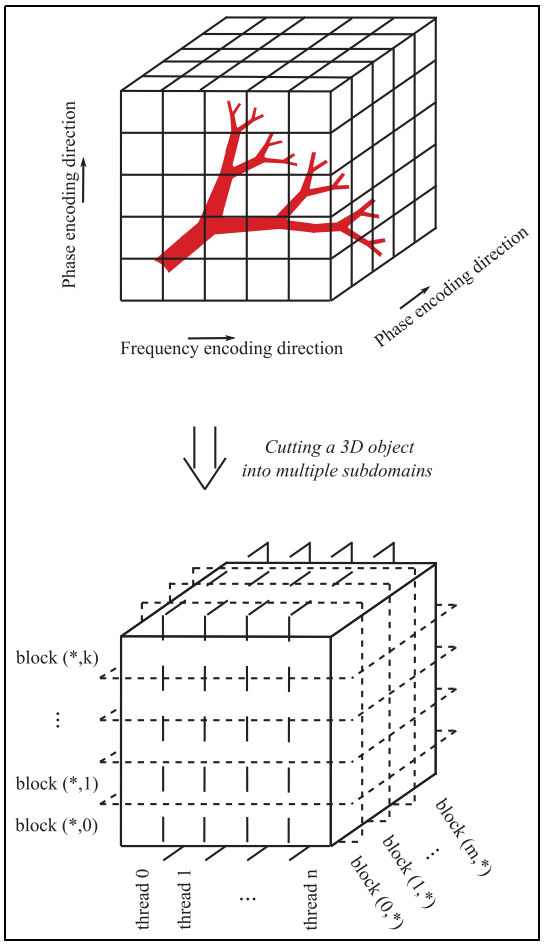


**Figure 4.** Data decomposition strategy. First, the 3D object is divided into parts along the phase encoding directions and these parts are spread over blocks. Second, each object part is divided further along the frequency encoding direction into smaller parts that are processed by various block threads. 2D grid of blocks and 1D blocks of threads are applied.

Listing 1: Pseudo code of the kernel function responsible for MRI influence.

```
__global__ procedure MRI_influence(gradients){
 int x, y, z, index;

 z=blockIdx.x;
 while z<zSize do
  y=blockIdx.y;
  while y<ySize do
   x=threadIdx.x;
   while x<xSize do
    id=index3Dto1D(z, y, x, ySize, xSize);
    pd=PD[id];
    t1=T1[id];
    t2=T2[id];
    magX=magnetX[id];
    magY=magnetY[id];
    magZ=magnetZ[id];

    //spatial encoding
    if gradients then
     df=calculateRotation(gradients);
    end if

    //help calculations
    sinRotation=sin(df);
    cosRotation=cos(df);
    E1=exp(-deltaT/t1);
    E2=exp(-deltaT/t2);

    //calculate new magnetization
    tMagX=magX*(E2*cosRotation)-magY*(E2*sinRotation);
    tMagY=magX*(E2*sinRotation)+magY*(E2*cosRotation);
    tMagZ=E1*magZ+pd*(1-E1);

    magnetX[id]=tMagX;
    magnetY[id]=tMagY;
    magnetZ[id]=tMagZ;

    x+=blockDim.x;
   end while
   y+=gridDim.y;
  end while
  z+=gridDim.x;
 end while
end
```

**Listing 1:** Pseudo code of the kernel function responsible for MRI influence.

simulated in consecutive time steps ($\Delta t$) because of blood flow that transports magnetizations between cubic elements. The magnetization of a cubic element in the next time step depends not only on its current magnetization value and the MRI influence (time dependency) but also on the magnetization of its neighboring cubic elements (spatial dependency), in accordance with equations (3) and (4). For this reason, each MRI event consists of a CPU loop that calls GPU kernels as it is illustrated in Figure 3.

As regards the excitation, relaxation and spatial encoding events, two kernel functions are called in each time step. The first one is responsible for the MRI influence, while the second one for the magnetization transport induced by the blood flow. Such an approach provides the synchronization for all threads (both inside and between blocks) after each time step. As a result, the time and spatial dependency between magnetizations of cubic elements is provided. During the MRI influence calculations, each thread accesses only the local magnetization values. However, during the magnetization transport calculations, threads have to also reach into neighboring cubic elements to read their magnetization values. These values are required to exchange magnetizations between neighboring cubic elements. Listings 1 and 2 show these two kernels functions in more detail.

The signal acquisition phase differs a little from other MRI events. Here, not only the MRI and flow influences are simulated, but also the signal from all cubic elements has to be read and saved in the k-space matrix. This operation is carried out by two additional kernel functions (*signal readout1* and *signal readout2*, see Figure 3). These two functions, in a single time step, provide the sum of all magnetizations (from all cubic elements). The first kernel performs the reduction of magnetization values for cubic elements inside blocks. The second kernel finishes the reduction with the use of one block where the number of treads equals to the number of blocks from the first kernel function.

Listing 2: Pseudo code of the kernel function responsible for magnetization transport.

```
__global__ procedure flow_influence(){
 int x, y, z, index;
 float xVel, yVel, zVel;

 z=blockIdx.x;
 while z<zSize do
  y=blockIdx.y;
  while y<ySize do
   x=threadIdx.x;
   while x<xSize do
    id=index3Dto1D(z, y, x, ySize, xSize);
    if cubicType[index]=='fluid' then
     xVel=cubicXVelocity[id];
     yVel=cubicYVelocity[id];
     //third direction is omitted in the further code
     //since of analogy to the x and y directions
     //zVel=cubicZVelocity[id];

     //count ΔM_OUT and ΔM_IN
     //propagation to right and top directions
     if xVel>=0 && yVel>=0 then
      //index of the left neighbor
      idL=id-1;
      //index of the bottom neighbor
      idB=index3Dto1D(z, y-1, x, ySize, xSize);
      //index of the bottom left neighbor
      idBL=index3Dto1D(z, y-1, x-1, ySize, xSize);

      partA=velX*velY;
      partB=(1-velX)*velY;
      partC=velX*(1-velY);

      magnetXIN=magnetX[idBL]*partA;
      magnetYIN=magnetY[idBL]*partA;
      magnetZIN=magnetZ[idBL]*partA;

      magnetXIN+=magnetX[idB]*partB;
      magnetYIN+=magnetY[idB]*partB;
      magnetZIN+=magnetZ[idB]*partB;

      magnetXIN+=magnetX[idL]*partC;
      magnetYIN+=magnetY[idL]*partC;
      magnetZIN+=magnetZ[idL]*partC;

      magnetXOUT=magnetX[id]*(partA+partB+partC);
      magnetYOUT=magnetY[id]*(partA+partB+partC);
      magnetZOUT=magnetZ[id]*(partA+partB+partC);
     end if

     //propagation to right and bottom directions
     if xVel>=0 && yVel<0 then
      //in analogy ...
     end if

     //propagation to left and top directions
     if xVel<0 && yVel>=0 then
      //in analogy ...
     end if

     //propagation to left and bottom directions
     if xVel<0 && yVel<0 then
      //in analogy ...
     end if
    end if
    deltaMagnetX[id] += magnetXIN - magnetXOUT;
    deltaMagnetY[id] += magnetYIN - magnetYOUT;
    deltaMagnetZ[id] += magnetZIN - magnetZOUT;
    x+=blockDim.x;
   end while
   y+=gridDim.y;
  end while
  z+=gridDim.x;
 end while

 //propagate magnetizations M = M + M_IN - M_OUT
 z=blockIdx.x;
 while z<zSize do
  y=blockIdx.y;
  while y<ySize do
   x=threadIdx.x;
   while x<xSize do
    id=index3Dto1D(z, y, x, ySize, xSize);
    if cubicType[index]=='fluid' then
     magnetX[id]+=deltaMagnetX[id];
     magnetY[id]+=deltaMagnetY[id];
     magnetZ[id]+=deltaMagnetZ[id];
    end if
    x+=blockDim.x;
   end while
  y+=gridDim.y;
  end while
  z+=gridDim.x;
 end while
end
```

**Listing 2:** Pseudo code of the kernel function responsible for magnetization transport.

Listing 3: AoS vs SoA memory layouts.

```
//M - number of cubic elements

//Array of Structs (AoS)
struct Magnetization {
      float x;
      float y;
      float z;
}
Magnetization objMagnetization[M];
objMagnetization[4].x = 1;

//Structure of Arrays (SoA)
struct Magnetization{
      float x[M];
      float y[M];
      float z[M];
}
Magnetization objMagnetization;
objMagnetization.x[4] = 1;
```

**Listing 3:** AoS vs SoA memory layouts.

After each repetition a successive line of the k-space matrix is filled by the MR signal. If the next repetition is needed, the algorithm starts again from the excitation. Otherwise, the k-space matrix is transferred from the GPU to the CPU. Finally, the MR image is created by the application of FFT to the received matrix at the CPU.

## 3.2 Memory access patterns

The GPU memory has a hierarchical structure (NVIDIA, 2015b). Several types of memories are provided with different scope, latency access, lifetime, and caching behavior. GPU memories can be grouped into two classes: small, fast on-chip memory (cache, resisters, etc.) and global memory (residing in device DRAM) with larger capacity but much higher latency access. In order to efficiently use these hardware resources, some algorithm improvements are added to the basic parallelization.

In GPU-parallelized applications the selection of an appropriate data layout for multi-valued data/containers (set of 3D points, 3D magnetization, etc.) is an important issue since it can drastically impact on computational efficiency (Mei and Tian, 2015). Generally, there are two major data layouts: Array-of-Structures (AoS) and Structure-of-Arrays (SoA) (see Listing 3), and the other more sophisticated are hybrid formats, like Array-of-Structures-of-Arrays (Strzodka, 2012). Although the same data is represented in both cases, each of these layouts implies a completely different memory access pattern. To improve overall performance, the memory access pattern should, primarily, minimize the number of memory transactions on the

off-chip global memory. Although SoA layout is often preferable from a GPU performance perspective, it is not always obvious which data layout gives better computational efficiency for a particular application (Govender et al., 2014; Giles et al., 2013). Thus, we decided to evaluate both layouts AoS and SoA.

In the algorithm, the most frequently read/written data is the information about the imaging object. At the same time, this information is also the largest data. In the global GPU memory the following parameters for each cubic element are stored:

- magnetization (three 32-bit floats, one for each direction);
- hydrodynamic properties (three 32-bit floats, one for each direction);
- MR characteristics: proton density, T1 and T2 relaxation times (three 32-bit floats).

In case of the SoA layout, all of these cubic parameters are organized in three structures (`struct Magnetization`, `struct Flow`, `struct MRIParam`). Each structure consists of three one-dimensional arrays of `M` floating point values (in accordance with Listing 3, lines 12–17). This provides coalesced memory access by accessing consecutive elements by threads within the same warp (or half-warp in older hardware) (Wilt, 2013). The coalesced global memory accessed by threads of a warp minimizes the number of memory transactions and as a result, minimizes DRAM bandwidth. In addition to the access to continuous memory locations, the memory alignment is also important. Thus, a padding mechanism (Cook, 2012) is applied to the SoA-based memory organization, in order to ensure data alignment in every row of an array. The mechanism pads extra cubic elements (filled by zero) to each row to meet the alignment requirements of a given device and thus, it may prove an additional speedup (Rojek et al., 2015).

In the case of the AoS layout, all of the cubic parameters are organized in three one-dimensional arrays (`Magnetization tabM[ M]`, `Flow tabF[ M]`, `MRIParam tabMRI[ M]`). Each array consists of `M` elements where each element is a three floating point valued container (in accordance with Listing 3, lines 3-8). The structure size of 12 bytes is suboptimal, since most types of memory are optimized for data access, where a chunk size is a power of two. CUDA specification says that if a global memory read/write operation does not fulfill the required size (1, 2, 4, 8, or 16 bytes) or the required alignment (its address is not a multiple of that size), the memory access can not be realized by a single memory transaction (NVIDIA, 2015a). In order to meet the 16 bytes alignment, for each of the structures a hidden 4 byte padding element is implicitly inserted by adding the specifier `__aling__(16)`. Such a structure (called Array-of-aligned-Structures, AoaS) occupies 16 bytes in memory (more than it is needed) but when it is used in an array, all array elements start at an address that is a multiple of 16 (which prevents from an interleaved memory access pattern). Moreover, to explore more of this data layout, we added to the solution another way to ensure the size requirement for the alignment using built-in data type `float4`.

At the beginning of each kernel function, data that is stored in the global memory and is frequently used (e.g. cubic element magnetization) is copied to local kernel variables explicitly. At the end of the kernel, the data is transferred back into the global memory containers. This way, each thread tries to accumulate temporary values (e.g. for magnetization) into registers (fast on-chip memory but of small capacity). If the data can not fit into register space, it is stored in a per-thread local memory which is nevertheless slower than registers. Local thread variables (for magnetization, flow and MR parameters) are stored in not aligned structures (or `float3` variables) since limited register space is a more important issue.

In the sequential algorithm implementation, the results of some repeated calculations are saved in auxiliary arrays before the MRI procedure and then used when needed (e.g. partial computation of the magnetization increase during relaxation after each time step $\Delta t$ individually for each cubic element: $\exp(-\Delta t/T_1(\mathbf{r}))$, $\exp(-\Delta t/T_2(\mathbf{r})$, etc.). In the GPU-parallelized algorithm, such global auxiliary arrays are not used. Results of frequently repeated calculations are saved locally in a kernel function when they are computed for the first time. Although it increases the number of arithmetic operations, it reduces redundant loads from the GPU global memory.

The GPU shared memory is used when the signal acquisition phase is simulated. The first kernel function performs the reduction of magnetization values for cubic elements inside blocks. It uses the shared memory inside thread blocks. The second kernel finishes the reduction with the use of one block and an array in global memory space. In addition, the possibility to finish the reduction at the CPU is added.

## 4 Performance analysis

This section shows the performance of the proposed GPU-based algorithm. Experiments were performed on different vascular structures using various NVIDIA graphics cards. In the paper, we focus on execution time and, in particular, on speedup relative to the sequential implementation. Moreover, the speedup obtained with the use of a few CPU cores by an OpenMP parallelization is also presented.

**Table 2.** Processing and memory resources of the NVIDIA graphics cards used in the experiments.

| NVIDIA graphics card | Engine | | Memory | | Compute |
|---|---|---|---|---|---|
| | No. CUDA cores | Clock rate (MHz) | Size (GB) | Band width (GB/s) | capability |
| Geforce GTX 760 | 1152 | 980 | 2 | 192.2 | 3.0 |
| Quadro K5000 | 1536 | 706 | 4 | 173.0 | 3.0 |
| Geforce GTX 780 | 2304 | 863 | 3 | 288.4 | 3.5 |
| Geforce GTX Titan Black | 2880 | 889 | 6 | 336.0 | 3.5 |
| Geforce GTX Titan X | 3072 | 1000 | 12 | 336.5 | 5.2 |

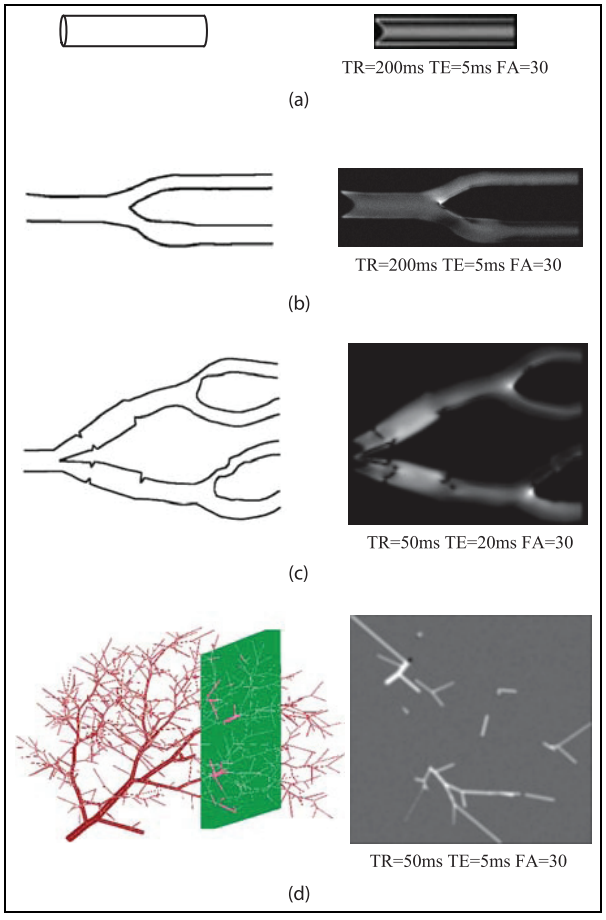


**Figure 5.** Geometry of tested objects/phantoms and examples of simulated MR images: (a) a single vessel, (b) a single bifurcation, (c) multiple bifurcations, (d) slice through the liver volume with many vascular structures.

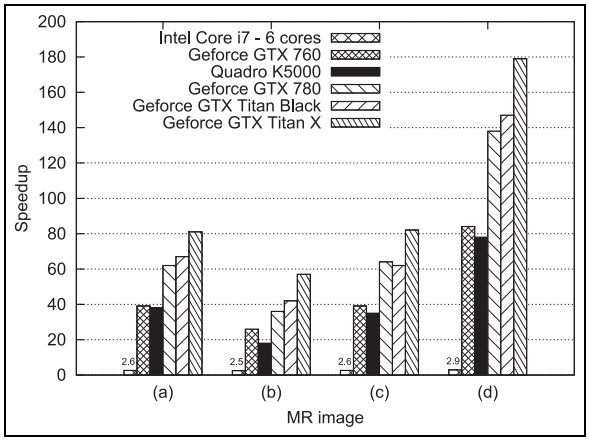


**Figure 6.** Speedup of the GPU-accelerated algorithm using the SoA + padding memory layout. Objects that are shown in Figure 5 and five different GPUs are tested. Moreover, the speedup of an OpenMP parallelization using six CPU cores is presented.

## 4.1 Setup

All experiments were performed on a workstation equipped with Intel Xeon CPU E5-2620 v3 (15 MB Cache, 2.40 GHz), 64 GB RAM and a single graphics card. We tested five different NVIDIA graphics cards described in Table 2 (four Kepler-based GPUs and one Maxwell-based: GTX Titan X). For each graphics card we gather basic specifications that cover the number of CUDA cores, a clock rate, available memory, bandwidth, and compute capability.

We used 64-bit Ubuntu Linux 14.04.02 LTS as an operating system. The sequential algorithm was implemented in C++ and compiled with the use of gcc version 4.8.2. The GPU-based parallelization was implemented in CUDA-C and compiled by nvcc CUDA 7.0 (NVIDIA, 2015a).

We present the results of four various vascular structures investigated in our previous papers (Jurczuk et al., 2013, 2014), where detailed flow and MRI settings were reported. Figure 5 shows the geometry of MRI phantoms (objects) and examples of simulated MR flow images with more and more complicated vascular structures. The GE imaging sequence (Bernstein et al., 2004) was used to obtain the images. Other basic imaging parameters are reported next to the MR images.

## 4.2 Results

Figure 6 shows the obtained speedup of the proposed GPU-accelerated algorithm in comparison to its sequential version. Speedups for various GPUs as well as for four different vascular structures (from a straight

**Table 3.** GPU profiling results for four objects shown in Figure 5 and three GPUs.

| NVIDIA GTX Titan X | | | | |
|---|---|---|---|---|
| Kernel | Time fraction (%) | | | |
| | (a) | (b) | (c) | (d) |
| MRI influence | 24.17 | 21.64 | 17.98 | 26.18 |
| Flow influence | 74.38 | 63.56 | 59.04 | 49.52 |
| Signal readout 1 | 1.21 | 7.27 | 11.86 | 14.82 |
| Signal readout 2 | 0.24 | 7.53 | 11.12 | 9.48 |
| **Geforce GTX Titan Black** | | | | |
| Kernel | Time fraction (%) | | | |
| | (a) | (b) | (c) | (d) |
| MRI influence | 25.86 | 21.96 | 17.56 | 25.56 |
| Flow influence | 73.19 | 67.91 | 56.76 | 47.97 |
| Signal readout 1 | 0.71 | 4.43 | 14.95 | 17.19 |
| Signal readout 2 | 0.24 | 5.69 | 10.73 | 9.28 |
| **Geforce GTX 760** | | | | |
| Kernel | Time fraction (%) | | | |
| | (a) | (b) | (c) | (d) |
| MRI influence | 24.28 | 21.3 | 25.9 | 25.91 |
| Flow influence | 74.78 | 70.84 | 53.39 | 53.39 |
| Signal readout 1 | 0.83 | 5.01 | 16.5 | 16.49 |
| Signal readout 2 | 0.11 | 2.85 | 4.21 | 4.21 |

vessel, through a few bifurcations and finally a liver volume with many vascular structures) are included separately. Moreover, the speedup obtained with the use of all six CPU cores by an OpenMP parallelization is also presented. It is clearly visible that the proposed GPU-based parallelization provides a significant decrease in computation time. All GPUs are able to obtain a speedup of at least one order higher than the OpenMP parallelization.

The results suggest that with the proposed approach even a regular PC with a medium-class graphics card is enough to accelerate the MRI simulations significantly. As it is expected, better graphics cards manage to achieve much better accelerations. The most expensive tested graphics cards in comparison to the cheapest ones are often 2 times faster and is able to simulate MRI of vascular structures about 180 times faster than the sequential simulations. With such accelerations, the speedup achieved by a six-core CPU using only the OpenMP parallelization is, to put the mildly - not very impressive.

The scale of the improvement is even more visible when the execution time between the sequential and parallel versions of the solution are compared. Currently, the time needed to simulate e.g. MR image in Figure 5(d) equals about 1 h with the use of Titan X GPU, instead of about 7-8 days by a single core CPU or 2–3 days using the OpenMP parallelization and six CPU cores. Moreover, the achieved speedup is comparable (images (a-c)) or even higher (image (d)) than the one obtained by a computer cluster of 16 nodes each equipped with 2 quad-core CPUs (Xeon 2.66 GHz) (128 CPU cores in total) and 16 GB RAM (Jurczuk et al., 2014).

There is also a difference in speedup between tested objects. The first reason could be the fraction of cubic elements that represent the object/image background where no MRI is simulated (black areas in images). The second one is the size of the object/image that does not always allow computational resources to be exploited efficiently. The object size and fraction of background elements were as follows: (a) $940 \times 190$, 0, (b) $167 \times 543$, 0.6, (c) $572 \times 740$, 0.7, (d) $1000 \times 1000 \times 70$, 0. The simulation box can differ from the object size due to additional background cubic elements added to objects in order to obtain the power of two image sizes, that allowed a FFT algorithm to be applied. After a reconstruction phase, the additional background black areas were cropped from the images (Jurczuk et al., 2013). The worst results are obtained for the single bifurcation object, where there are a lot of background elements and the horizontal object (167) as well as simulation box size (320) are too small to efficiently use GPU resources. The best speedup, in turn, is achieved for image (d) since there are no background elements and the object size is big enough.

The differences in speedup between tested objects can be further explained by the GPU profiling results included in Table 3. Calculating the flow influence demands much more memory transactions to/from global memory (read magnetizations from all neighboring cubic elements) than the MRI simulation itself, thus, the former operation is at the first place in all cases. For the straight tube geometry, in all cubic elements the flow has to be simulated, thus, the kernel responsible for this phenomena dominates over other actions. Also in geometry (d) all cubic elements take part in simulations, however, only a small fraction of them (where the vessels are present) need both MRI and flow influences calculations. It is a part of the reason why the speedup for (d) geometry is higher than for the straight tube geometry. Generally, the more time spent for flow influence calculations, the lower speedup we obtain. However, also the time spent for readout kernels and the ratio of the number of fluid cubic elements to the total number of cubic elements have to be taken into account. For the straight tube geometry, the readout kernels take a very small time fraction because calculations in other kernels dominate as this ratio equals 1 (the highest ratio from all the objects). The signal readout kernels have to fill all k-space cells by signals from all object cubic elements no matter what a cubic

element represents: fluid, stationary tissue or background. The results obtained using different GPUs for particular objects are coherent. Moreover, in many cases, they are very similar.

In further part of this section, the proposed GPU-accelerated algorithm is thoroughly verified, concerning added improvements, object size, blocks × threads configurations and the size of data alignment. Figure 7 shows the influence of the successively added improvements to the basic parallelization:

- AoS1 - basic parallelization using AoS data layout;
- AoS2 - AoS1 + the reduction is finished by GPU;
- AoS3 - AoS2 + data stored in global memory and frequently used is copied to local kernel variables explicitly;
- AoaS - AoS3 + structure alignment;
- SoA - all earlier improvements + SoA data layout;
- SoA + padding.

The results concern the straight tube geometry. It can be observed that the best results are obtained when the SoA data layout is applied. The smallest change in simulation time provides the AoS2 improvement. The biggest improvement is achieved when data alignment is added to the structure definition (AoaS data layout) as well as when SoA data layout is used. Also in the case of SoA-based memory organization, the memory alignment is important. We did not find any remarkable performance gains between using built-in data type `float4` and specifier `__aling__(16)`.

We also verified execution times with respect to the object size for the straight vessel geometry (Figure 8). The following lattice sizes are tested: 1000, 2000, 4000, 8000, 16000, which provide the straight tube geometries consisting of $237 \times 49$, $471 \times 96$ and $940 \times 190$, $1880 \times 380$, $3760 \times 760$ cubic elements (original/flow object sizes), respectively. We observe that the speedup increases when the lattice size grows. For the lattice size of 1000, the results are not so different between GPUs in comparison to higher spatial resolution images. This may suggest that when the object size is too small, there are not enough jobs in face of the increasing number of CUDA cores provided by more powerful GPUs. For the lattice size between 2000 and 16000, we clearly see that as the clock rates and the number of cores increase, simulations finish faster. The only exception is the GTX 760 GPU which achieves much lower speedup than other Kepler-based GPUs despite the fact that its clock rate is higher than other GPUs. This GPU is, however, equipped with at least two times less CUDA cores as well as its memory bandwidth is also lower than other faster GPUs.

We have also experimentally checked if different sizes of the data processed in each block/thread influences the algorithm speedup. Figure 9 presents the

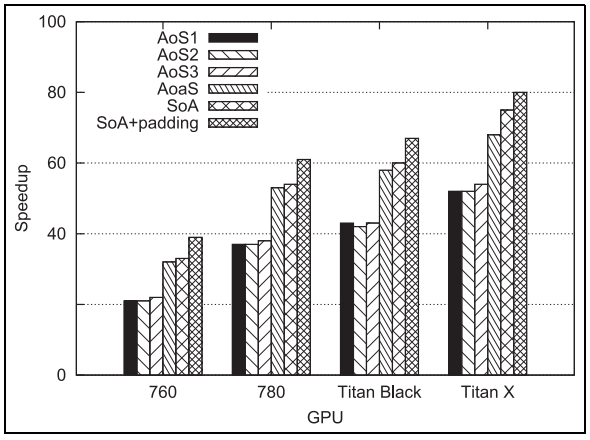


**Figure 7.** The influence of successive improvements.

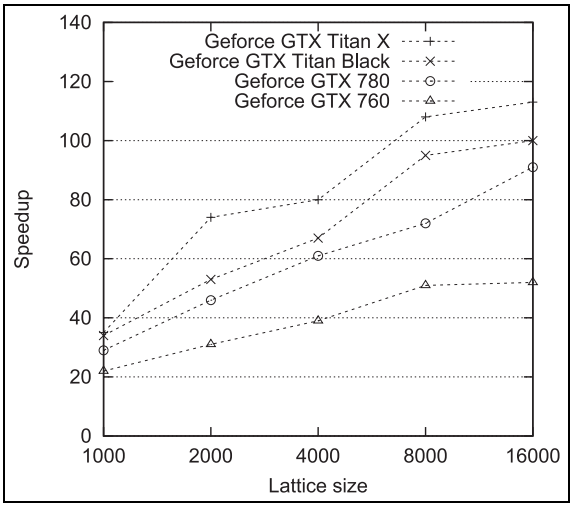


**Figure 8.** The influence of the object size.

speedup for the straight tube geometry of different spatial resolution for three graphics cards (Geforce GTX 760, Titan Black and Titan X). The tested Kepler-based GPUs achieve the best speedup when the number of threads is not too high (not more than 512). For Titan X GPU, it can be observed that for higher resolution (bigger object size) the configuration of blocks × threads equal to $1024 \times 512$ fits the best, whereas for smaller ones it is not so evident (and other configurations win). Since the memory bandwidths of GTX Titan Black and Titan X are almost the same, the reason can be found in the number of CUDA cores and the clock rates which are higher for the second GPU. However, the differences between verified configurations are always in range of $0 - 20$.

The influence of the size of memory alignment in SoA layout has also been verified. Figure 10 shows the speedup for various GPUs when different padding size is used. The size of the data alignment is expressed in bytes and is selected from the set [32; 64; 128; 256, 512, 1024]. For GTX 760 GPU the optimal memory

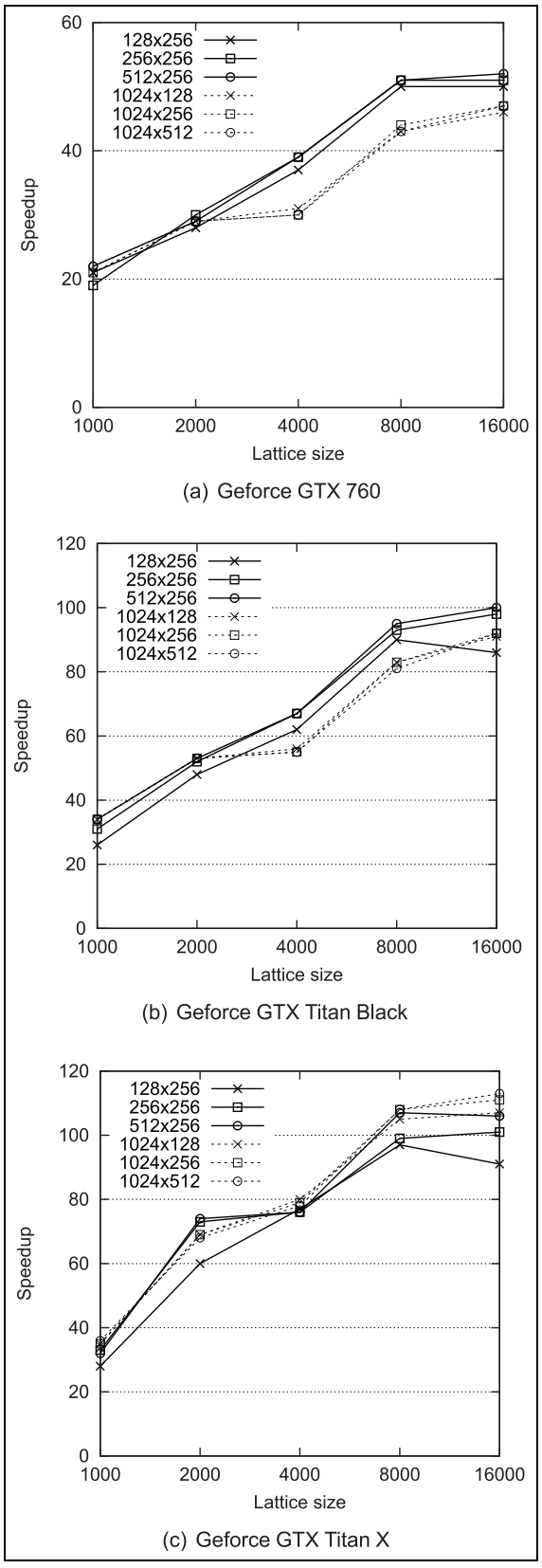


**Figure 9.** The influence of the number of blocks×threads. Different object lattice sizes and the straight tube geometry using various GPUs are considered.

alignment equals 128 B. The other GPUs provide the best performance when the size equals 256 B.

Having the performance results for different graphics cards, for various objects and with different configurations gives us an opportunity to estimate the mean speedup for particular GPUs (see Table 4). As expected, the best performance is achieved using the most powerful Maxwell-based GPU: Geforce GTX Titan X. We also observe that the GTX 780 and the GTX Titan Black GPUs give very similar performance. Concerning the GTX 760 and the Quadro K5000 GPUs, they achieve the worst speedup, but still high enough to be successfully used in fast MRI simulations.

## 5 Conclusion and future works

MRI simulators play an important role in improving this imaging modality and developing new imaging techniques. However, MRI models are often simplified to complete simulations in a reasonable time. In this paper, the authors propose a GPU-based parallelization to accelerate simulations of MRI of vascular structures. To efficiently exploit GPU resources, different memory layouts and further additional optimizations (like padding) are applied. A CUDA programming model is used.

The approach was tested with various GPUs (both Kepler- and Maxwell-based) and using different vascular structures. Moreover, its performance was analyzed in-depth in response to successive improvements and with various configurations, concerning scalability, thread/blocks settings and padding size. The results show that our solution is fast and scalable. It provides the speedup of at least one order higher than an OpenMP parallelization. The time to simulate MR image in Figure 5(d) decreases from several days to a few hours. Even a regular PC equipped with a medium-class graphics card is sufficient for our algorithm to reduce simulation time significantly. We believe that it opens a perspective to simulate more and more complex (that is more realistic) vascular structures. Moreover, the proposed GPU-based approach may be easily adapted in other algorithms concerning related phenomena like diffusion or perfusion.

We see many promising directions for the future research. We plan to deal with a multi-GPU parallelization to speedup the MRI simulations even further. Using OpenCL interface, instead of CUDA, could provide further improvements and the portability to GPUs from multiple vendors (McIntosh-Smith et al., 2015). Another improvement may refer to asynchronous (Farber, 2011) or collaborative (Mittal, 2015) CPU + GPU execution. We also plan to use the presented parallelization in the modeling of contrast agent

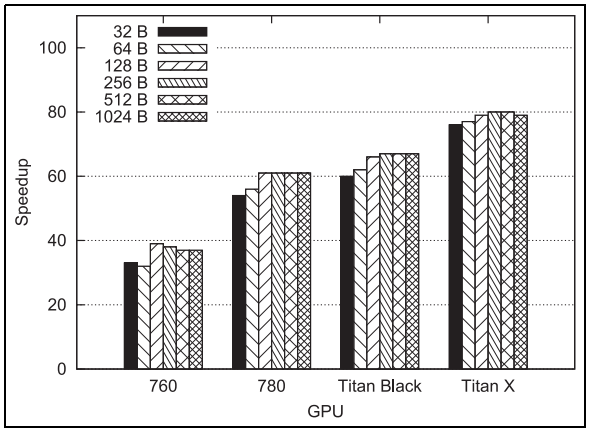


**Figure 10.** The influence of the size of memory alignment in SoA layout.

**Table 4.** Mean speedups over all tested images using optimal configurations.

| NVIDIA graphics card | Mean speedup |
|---|---|
| Geforce GTX 760 | 47 |
| Quadro K5000 | 42 |
| Geforce GTX 780 | 75 |
| Geforce GTX Titan Black | 80 |
| Geforce GTX Titan X | 100 |

propagation and dynamic contrast-enhanced MRI (Mescam et al., 2010).

## Acknowledgements

I would like to thank student Dariusz Murawski for his C++ implementation effort on the initial version of the algorithm, as well as Wojciech Kwedlo and Marcin Czajkowski for providing NVIDIA Geforce GTX 760 and NVIDIA Geforce GTX Titan Black graphics cards.

## Declaration of Conflicting Interests

The author(s) declared no potential conflicts of interest with respect to the research, authorship, and/or publication of this article.

## Funding

The author(s) disclosed receipt of the following financial support for the research, authorship, and/or publication of this article: This work was supported by the Grants W/WI/2/2014 and S/WI/2/2013 from Bialystok University of Technology.

## References

Aibinu AM, Salami MJE, Shafie AA, et al. (2008) MRI reconstruction using discrete Fourier transform: A tutorial. *World Academy of Science Engineering and Technology* 18: 179–185.

Aiyagari V and Gorelick PB (2011) *Hypertension and Stroke: Pathophysiology and Management*. Totowa, NJ: Humana Press.

Benoit–Cattin H, Collewet G, Belaroussi B, et al. (2005) The SIMRI project: A versatile and interactive MRI simulator. *Journal of Magnetic Resonance* 173(1): 97–115.

Bernstein MA, King KF and Zhou XJ (2004) *Handbook of MRI Pulse Sequences*. Burlington, MA: Elsevier Academic Press.

Bittoun J, Taquin J and Sauzade M (1984) A computer algorithm for the simulation of any nuclear magnetic resonance (NMR) imaging method. *Magnetic Resonance Imaging* 2(2): 113–120.

Bloch F, Hansen WW and Packard M (1946) Nuclear induction. *Physical Review* 69: 127.

Cheng J, Grossman M and McKercher TY (2014) *Professional CUDA C Programming*. Indianapolis, IN: Wrox.

Cook S (2012) *CUDA Programming: A Developer's Guide to Parallel Computing with GPUs*. Waltham, MA: Morgan Kaufmann.

Dyverfeldt P, Gardhagen R, Sigfridsson A, et al. (2009) On MRI turbulence quantification. *Magnetic Resonance Imaging* 27(7): 913–922

Farber R (2011) *CUDA Application Design and Development*. Waltham, MA: Morgan Kaufmann.

Garin E, Lenoir L, Edeline J, et al (2013) Boosted selective internai radiation therapy with 90 Y-loaded glass microspheres (B-SIRT) for hepatocellular carcinoma patients: A new personalized promising concept. *European Journal of Nuclear Medicine and Molecular Imaging* 40(7): 1057–1068.

Giles MB, Mudalige GR, Spencer B, et al. (2013) Designing OP2 for GPU architectures. *Journal of Parallel and Distributed Computing* 73(11): 1451–1460.

Govender N, Wilke DN, Kok S, et al. (2014) Development of a convex polyhedral discrete element simulation framework for NVIDIA kepler based GPUs. *Journal of Computational and Applied Mathematics* 270: 386–400.

Grama A, Karypis G, Kumar V, et al. (2003) *Introduction to Parallel Computing*. Boston, MA: Addison–Wesley.

Grinberg L, Cheever E, Anor T, et al. (2011) Modeling blood flow circulation in intracranial arterial networks: A comparative 3-D/1-D simulation study. *Annals of Biomedical Engineering* 39(1): 297–309.

Jochimsen TH, Schafer A, Bammer R, et al. (2006) Efficient simulation of magnetic resonance imaging with Bloch–Torrey equations using intra-voxel magnetization gradients. *Journal of Magnetic Resonance* 180(1): 29–38.

Jou LD and Saloner D (1998) A numerical study of magnetic resonance images of pulsatile flow in a two dimensional carotid bifurcation: A numerical study of MR images. *Medical Engineering & Physics* 20(9): 643–652.

Jou LD, van Tyen R, Berger SA, et al. Calculation of the magnetization distribution for fluid flow in curved vessels. *Magnetic Resonance in Medicine* 35(4): 577–584.

Jurczuk K, Kretowski M, Bellanger JJ, et al. (2013) Computational modeling of MR flow imaging by the lattice Boltzmann method and Bloch equation. *Magnetic Resonance Imaging* 31(7): 1163–1173.

Jurczuk K, Kretowski M, Eliat PA, et al. (2014) In silico modeling of magnetic resonance flow imaging in complex

vascular networks. *IEEE Transactions on Medical Imaging* 33(11): 2191–2209.

Jurczuk K, Murawski D, Kretowski M, et al. (2016) GPU accelerated simulations of magnetic resonance imaging of vascular structures. *Lecture Notes in Computer Science* 9573: 389–398.

Kretowski M, Rolland Y, Bezy–Wendling J, et al. (2003) Physiologically based modeling for medical image analysis: Application to 3D vascular networks and CT scan angiography. *IEEE Transactions on Medical Imaging* 22(2): 248–257.

Kuperman V (2000) *Magnetic resonance imaging physical principles and applications*, San Diego: Academic Press.

Lasheras JC (2007) The biomechanics of arterial aneurysms. *Annual Review of Fluid Mechanics* 39: 293–319.

Lorthois S, Stroud–Rossman J, Berger S, et al. (2005) Numerical simulation of magnetic resonance angiographies of an anatomically realistic stenotic carotid bifurcation. *Annals of Biomedical Engineering* 33(3): 270–283.

Marshall I (2010) Computational simulations and experimental studies of 3D phasecontrast imaging of fluid flow in carotid bifurcation geometries. *Journal of Magnetic Resonance Imaging* 31(4): 928–934.

McIntosh-Smith S, Price J, Sessions RB, et al. (2015) High performance in silico virtual drug screening on many-core processors. *International Journal of High Performance Applications* 29(2): 119–134.

Mei G and Tian H (2016) Impact of data layouts on the efficiency of GPU-accelerated IDW interpolation. *SpringerPlus* 5(104): 1–18.

Mescam M, Kretowski M and Bezy–Wendling J (2010) Multiscale model of liver DCE–MRI towards a better understanding of tumor complexity. *IEEE Transactions on Medical Imaging* 29(3): 699–707.

Mittal S and Vetter J (2015) A survey of CPU-GPU heterogeneous computing techniques. *ACM Computing Surveys* 47(4): 69:1–69:35.

Nishimura DG, Jackson JL and Pauly JM (1991) On the nature and reduction of the displacement artifact in flow images. *Magnetic Resonance in Medicine* 22(2): 481–492.

NVIDIA (2015a) CUDA C Programming Guide. Technical Report, NVIDIA. Available at: https://docs.nvidia.com/cuda/cuda-c-programming-guide/

NVIDIA (2015b) CUDA C Best Practices Guide in CUDA Toolkit. Technical Report, NVIDIA. Available at: https://docs.nvidia.com/cuda/cuda-c-best-practices-guide/.

Rahimi MS, Holmes JH, Wang K, et al. (2015) Flow-induced signal misallocation artifacts in two-point fatwater chemical shift MRI. *Magnetic Resonance in Medicine* 73(5): 1926–1931.

Rojek K, Ciznicki M, Rosa B, et al. (2015) Adaptation of fluid model EULAG to graphics processing unit architecture. *Concurrency Computation: Practice and Experience* 27(4): 937–957.

Strzodka R (2012) Abstraction for AoS and SoA layout, in *C++ Computing Gems: Jade Edition*. Waltham, MA: Morgan Kaufmann.

van der Graaf AWM, Bhagirath P, Ghoerbien S, et al. (2014) Cardiac magnetic resonance imaging: Artefacts for clinicians. *Netherlands Heart Journal* 22(12): 542–549.

von Hagen W (2006) *The Definitive Guide to GCC*. 2nd ed. Berkeley, CA: Apress.

Westbrook C, Roth CK and Talbot J (2011) *MRI in Practice*. 4th ed. Oxford: Wiley–Blackwell.

Wilt N (2013) *Cuda Handbook: A Comprehensive Guide to GPU Programming*. Upper Saddle River, NJ: Addison–Wesley.

Xanthis CG, Venetis IE and Aletras AH (2014) High performance MRI simulations of motion on multi-GPU systems. *Journal of Cardiovascular Magnetic Resonance* 16: 48.

Xanthis CG, Venetis IE, Chalkias AV, et al. (2014) MRISIMUL: A GPU-based parallel approach to MRI simulations. *IEEE Transactions on Medical Imaging* 33(3): 607–617.

Yuen D, Wang L, Chi X, et al. (2013) *GPU Solutions to Multi-scale Problems in Science and Engineering*. Berlin: Springer.

Zeigler BP, Praehofer H and Kim TG (2000) *Theory of Modeling and Simulation*. San Diego, CA: Academic Press.

## Author Biographies

*Krzysztof Jurczuk* completed his joined PhD between the University of Rennes 1, France, and the Faculty of Computer Science, at the Bialystok University of Technology, Poland, in 2013. He is currently working as an Assistant Professor in the Faculty of Computer Science at the Bialystok University of Technology, Poland. His research interests focus on biomedical informatics and parallel computing.

*Marek Kretowski* received a joined PhD degree in 2002 from the Faculty of Computer Science, at the Bialystok University of Technology, Poland, and from the University of Rennes 1, France. He is currently working as a Professor at the Faculty of Computer Science at the Bialystok University of Technology, Poland. His research interests focus on biomedical applications of computer science (modeling for image understanding, image analysis), bioinformatics and data mining.

*Johanne Bezy–Wendling* received a PhD degree from the University of Rennes 1, France, in 1997. She is Associate Professor at the Signal and Image Processing Laboratory, in the University of Rennes 1, INSERM U1099, France. She has been working on medical image analysis and modeling for 15 years in close collaboration with clinicians. Her work is devoted to image analysis and physiological modeling of tissues and vascular systems with main application to the liver.