Simulations of the Electrical Activity in the Heart with Graphic Processing Units

B. M. Rocha F. O. Campos G. Plank
R. W. dos Santos M. Liebmann G. Haase

SFB-Report No. 2009–016 May 2009

Supported by the
Austrian Science Fund (FWF)

SFB sponsors:

- **Austrian Science Fund** (FWF)
- **University of Graz**
- **Graz University of Technology**
- **Medical University of Graz**
- **Government of Styria**
- **City of Graz**

Simulations of the Electrical Activity in the Heart with Graphic Processing Units

B. M. Rocha[1], F. O. Campos[1], G. Plank[1 2], R. W. dos Santos[3], M. Liebmann[4] and G. Haase[4]

[1] Medical University of Graz, Institute of Biophysics, Harrachgasse 21, 8010 Graz, Austria
[2] Medical University of Graz, Institute of Physiology, Harrachgasse 21, 8010 Graz, Austria
[3] Federal University of Juiz de Fora, Department of Computer Science and Master Program in Computational Modeling, Campus Universitário de Martelos, 36036-330 Juiz de Fora, Brazil
[4] Karl-Franzens-University Graz, Institute for Mathematics and Scientific Computing, Heinrichstrasse 36, 8010 Graz, Austria

Abstract. The modeling of the electrical activity of the heart is of great medical and scientific interest, because it provides a way to get a better understanding of the related biophysical phenomena, allows the development of new techniques for diagnoses and serves as a platform for drug tests. The cardiac electrophysiology may be simulated by solving a partial differential equation (PDE) coupled to a system of ordinary differential equations (ODEs) describing the electrical behavior of the cell membrane. The numerical solution is, however, computationally demanding because of the fine temporal and spatial sampling required. The demand for real time high definition 3D graphics made the new graphic processing units (GPUs) a highly parallel, multithreaded, many-core processor with tremendous computational horsepower. It makes the use of GPUs a promising alternative to simulate the electrical activity in the heart. The aim of this work is to study the performance of the use of GPUs to solve the equations underlying the electrical activity in a simple cardiac tissue.

1 Introduction

The phenomenon of electrical propagation in the heart comprises a set of complex nonlinear biophysical processes. Its multi-scale nature spans from nanometer processes such as ionic movements and protein dynamic conformation, to centimeter phenomena such as whole heart contraction [1]. Computer models have become valuable tools for the study and comprehension of such complex phenomena, as they allow different information acquired from different physical scales and experiments to be combined in order to generate a better picture of the whole system functionality.

There are two components that contribute to the modeling of cardiac electrical propagation [2]. The first is a model of cellular membrane dynamics, describing the flow of ions across the cell membrane. Such models are usually formulated as a system of nonlinear ODEs describing processes occurring on a wide range of time scales. These models are being continually developed to give an increasingly detailed and accurate description of cellular physiology. However, this development also tends to increase the complexity of the models making them substantially more costly to solve numerically [3]. The second is an electrical model of the tissue that describes how currents from one region of a cell membrane interact with other regions and with neighboring cells. When these two components are put together, the ODEs are coupled to a set of PDEs giving rise to the bidomain model [4]. Despite being currently the most complete description of cardiac electrical activity, the numerical solution of the bidomain equations are very computationally demanding [5–8]. A simpler approximation can be obtained from the model formulation in [4] by considering the extracellular potential to be constant; or the tissue conductivity to be isotropic; or the intracellular and extracellular conductivities to have equal anisotropy ratios [9]. If one of these assumptions is made, the equations are reduced to the monodomain model [10], which has been recently demonstrated to yield essentially to equivalent results as the bidomain model [11].

Nevertheless, the numerical solution of these tissue models involves the discretization in space and time of PDEs as well as the integration of nonlinear systems of ODEs. Using an appropriate decomposition for the time discretization, the nonlinearity of the system may be isolated, i.e. for each node of the spacial mesh a system of ODEs, that comes from the cellular model, may be solved independently. To perform realistic simulations of cardiac tissue, a large number of ODE systems must be solved at each time step, which contributes substantially to the total computational work. Therefore, it is necessary to pursuit new efficient ways of solving the large linear algebraic system that arises from the discretization of the PDEs as well as the systems of ODEs associated with the cell models.

The advent of multi-core CPUs and many-core GPUs means that mainstream processor chips are now parallel systems. Many efforts have been made to solve efficiently the equations modeling the cardiac electrical activity on a single CPU as well as in cluster of computers [3, 5, 6, 8, 12]. However, the newer programmable GPU has become a highly parallel, multithreaded, many-core processor with tremendous computational horsepower and very high memory bandwidth. The challenge is to develop application software that transparently scales its parallelism to leverage the increasing number of processor cores. CUDA is a parallel programming model and software environment designed to overcome this challenge while maintaining a low learning curve for programmers familiar with standard programming languages such as C.

The aim of this work is to evaluate the performance of CPU and GPU solvers for the monodomain model developed in standard C and extended to the NVIDIA CUDA parallel environment [13].

2 Mathematical Formulation

The monodomain model may be written as:

$$\beta C_m \frac{\partial V_m}{\partial t} = \nabla \cdot (\boldsymbol{\sigma} \nabla V_m) - \beta I_{ion}(V_m, \eta_i) \tag{1}$$

$$\frac{\partial \eta_i}{\partial t} = f(V_m, \eta_i) \tag{2}$$

where β is the surface-to-volume ratio of the cardiac cells, C_m is the membrane capacitance, V_m is the transmembrane voltage, $\boldsymbol{\sigma}$ is the conductivity tensor and I_{ion} is the ionic current across the membrane, which depends on V_m as well as on many other state variables represented by η_i.

In this work, the Luo-Rudy model (LR-I model) [14], a mathematical description of the action potential (AP) in ventricular cells, was considered to simulate the kinetics of I_{ion}. In this mammalian ventricular model, I_{ion} is defined as:

$$I_{ion} = I_{Na} + I_{si} + I_K + I_{K1} + I_{Kp} + I_b \tag{3}$$

where I_{Na} is the fast sodium current, I_{si} is the slow inward current, I_K is the time-dependent potassium current, I_{K1} is the time-independent potassium current, I_{Kp} is the plateau potassium current and I_b is time-independent background current.

Four of the 6 ionic currents in Eq. 3 are controlled by the action of ionic gates described by ODEs of the form:

$$\frac{dn}{dt} = \frac{n_\infty(V_m) - n}{\tau_n(V_m)} \tag{4}$$

where

$$n_\infty(V_m) = \frac{\alpha(V_m)}{\alpha(V_m) + \beta(V_m)} \tag{5}$$

and

$$\tau_n(V_m) = \frac{1}{\alpha(V_m) + \beta(V_m)} \tag{6}$$

where n represents any gating variable, $n_\infty(V_m)$ is the steady-state value of n, and $\tau_n(V_m)$ is its time constant.

In addition to the ODEs for the gating parameters, the model includes an ODE for the intracellular calcium concentration $[Ca^{2+}]_i$:

$$\frac{d[Ca^{2+}]_i}{dt} = -10^{-4} I_{si} + 0.07 \left(10^{-4} - [Ca^{2+}]_i\right) \tag{7}$$

In summary, the model is based on a set of 8 ODEs describing ionic currents and intracellular calcium concentration. A more complete description can be found in [14].

3 Numerical Schemes

In order to obtain a numerical solution for Eqs. (1)-(2), the Godunov operator splitting [15] was employed. The coupled system is then reduced to a two numerical step scheme that involves the solutions of a parabolic PDE and a nonlinear system of ODEs at each time step. This splitting leads to the following two steps algorithm:

1. Solve the systems of ODEs

$$\frac{\partial V_m}{\partial t} = -\beta I_{ion}(V_m, \eta_i) \tag{8}$$

$$\frac{\partial \eta_i}{\partial t} = f(V_m, \eta_i) \tag{9}$$

2. Solve the parabolic problem

$$\beta \frac{\partial V_m}{\partial t} = \nabla \cdot (\boldsymbol{\sigma} \nabla V_m) \tag{10}$$

The Finite Element Method (FEM) was used for the spatial discretization of Eq. 10 and the implicit Euler method to advance the solution in time. Applying the FEM for the spatial discretization, a system of linear equations must be solved at each time step:

$$(M + CK)\upsilon^{k+1} = M\upsilon^k \tag{11}$$

where υ discretizes V_m at time $k\Delta t$; M and K are the mass and stiffness matrices from the FEM discretization, respectively; and the constant C is $\frac{\beta C_m}{\Delta t}$. The non-preconditioned conjugate gradient (CG) method was used to solve this system.

Due to the discretization nature of the FEM, K and M are essentially sparse and thus specific data structures were used to store them in an efficient manner. In the present work we used the well known *compressed sparse row* (CSR) format [16] for the matrix representation. This format stores a sparse matrix A in three arrays Ax, Aj and Ap that hold, respectively, the nonzero entries, the columns of these entries and the pointers to the beginning of each row in the arrays Ax and Aj, as illustrated in Figure 1.

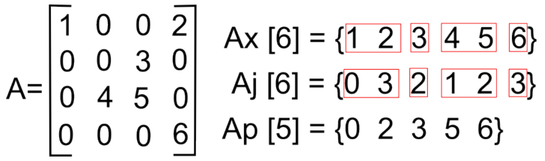

Fig. 1. Example of storage of a sparse matrix using the CSR format.

The system of ODEs in Eqs. (8)-(9) was integrated using the explicit Euler method. Although is well known that explicit numerical methods have strong limitations because stability restrictions [3], they are widely used due to their simplicity of implementation [5, 8].

4 Methods

We have developed three different *in-silico* tissue preparations for our investigations. Each tissue is a two dimensional model, which was discretized through the FEM with bilinear elements with zero flux boundary conditions imposed. Spatial discretization was set to $\Delta x = 12.5\,\mu m$. Numerical tests performed using the forward Euler method to solve the system of ODEs revealed that a $\Delta t = 0.01\,ms$ is the largest allowable time step for stability constraints. Table 1 presents the dimensions of the simulated tissues as well as the properties of the FEM meshes. The absolute tolerance of the CG method was set to 10^{-6}. The parameters of the monodomain model were taken from the literature [1, 17]: $C_m = 1\,uF/cm^2$; $\beta = 0.14\,\mu m^{-1}$; and tissue conductivity was considered homogeneous and isotropic with $\sigma = 0.0001\,mS/\mu m$.

Table 1. Dimensions of the *in-silico* tissue preparation and the properties of their associated FEM meshes.

Tissue	**Nodes**	**Elements**
$2 \times 2\,mm$	25921	25600
$4 \times 4\,mm$	103041	102400
$8 \times 8\,mm$	410881	409600

We simulated $100\,ms$ of electrical propagation in these cardiac tissues after applying a stimulus in a predefined area. The stimulus was made by setting V_m in the left edge of the meshes to a value above the threshold required to generate an AP. Figure 2 (a) illustrates the electrical propagation in a simulated tissue of dimensions 4x4 mm, and (b) the time course of the AP of the cell highlighted in (a).

In order to obtain high performance using GPU programming two requisites should be kept in mind. The first one is the challenge inherent to the development of applications for parallel systems, which should balance the workload and scaled it over a number of processors. The second is that different than in the CPU, in the GPU the global memory space is not cached, so it is also very important to follow the right access pattern to get maximum memory bandwidth [13]. We implemented the numerical solution of Eqs. (1)-(2), in which I_{ion} is given by Eq. 3, using standard C (CPU) and then, extended it to the NVIDIA CUDA parallel environment (GPU). Specific CUDA kernels were developed to solve the system of ODEs as well as the parabolic problem. All kernels were launched with 256 threads per block and the number of blocks per grid was set accordingly to the size of the problem, i.e. according to the number of nodes in the FEM mesh.

The ODE Systems We have designed two kernels to solve the system of ODEs related to Eq. 2. The first one refers to the setting of initial conditions to the

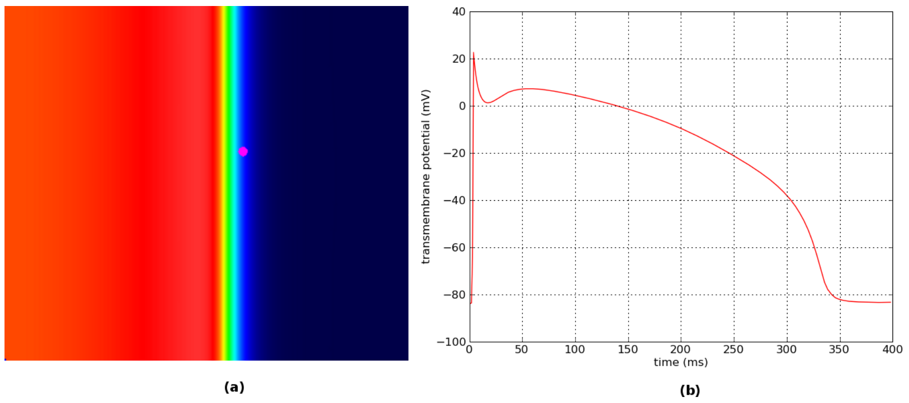

Fig. 2. Simulation results. (a) Spatial distribution of V_m in a 4x4 mm tissue simulation 3 ms after the stimulus onset. (b) the AP related to the cell highlighted in (a).

cells, i.e. to the systems of ODEs. The state variables of m cells were stored in a unidimensional vector of size $m * n$, where n is the number of differential equations of the ionic model ($n = 8$ according to the Luo and Rudy (1991) model [14]). This approach was chosen in order to simplify the structure of the CUDA code as well as the memory access by the threads. Since each cell model has the same number of state variables n, $m * n$ threads are required to set all initial conditions to all cells. The strategy for setting up initial conditions is illustrated in Figure 3 (a). Note that the threads access the memory in a coalesced way, which (according to [13]) should improve the performance of the code. The second kernel integrates the ODEs at each time step and differently from the first one, every thread is used to solve one ODE system. Therefore, each thread has to load and operate over all state variables of one cell model. Figure 3 (b) illustrates this strategy.

The Parabolic Problem The solution of the parabolic problem essentially consists of basic linear algebra operations such as vector scalar product and matrix vector multiplication, which in our case is sparse. The implementation of basic linear algebra kernels in CUDA such as vector addition, vector scale and the SAXPY operation is a straightforward task since each thread is assigned to do the computation of one position of the array. On the other hand, the implementation of the scalar product is not straightforward and requires a reduction operation. In this work this kernel was implemented doing a partial sum of the vectors on each multiprocessor and then a reduction on shared memory.

The CSR sparse matrix vector multiplication (SPMV) can be easily parallelized since the product of one row of the matrix and the vector is independent of other rows. We implemented it assigning one thread to compute one dot product of the matrix row and the vector. Although, there are more efficient variants

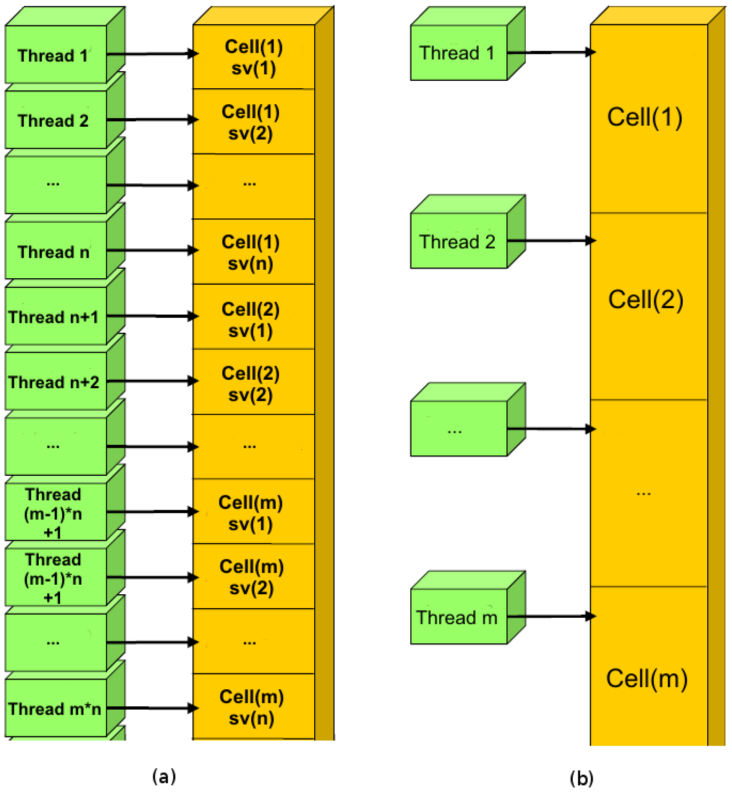

Fig. 3. Strategy used to set up the initial conditions (a) and to solve the ODE systems (b) in a cardiac tissue containing m cells modeled by n ODEs.

of this algorithm such as the Block Compressed Row Storage (BCRS) [18], we have only tested the CSR kernel as described before.

In order to check for possible deviations in the numerical solutions due to the use of single precision in the arithmetic operations, the results related to V_m were stored and used for comparisons. The numerical solutions obtained using single precision were compared against one computed using double precision through the relative root-mean-square norm (RRMS):

$$error = 100 \frac{\sqrt{\sum_t \sum_{i,j} (\upsilon_{i,j}^t - \nu_{i,j}^t)^2}}{\sqrt{\sum_t \sum_{i,j} (\upsilon_{i,j}^t)^2}} \tag{12}$$

where υ is the solution computed using double precision, ν is the solution computed using single precision and t and i, j are indexes in time and space, respectively.

5 Results

In this section we present the numerical results obtained for the monodomain simulations using a ventricular cell model on graphics processing units through NVIDIA CUDA environment. Simulations were performed on a single processor of a quad-core Intel Xeon E5420 2.50GHz and 16GB of shared memory equipped with a NVIDIA GeForce GTX 295, a dual-GPU graphic card with two GT200s emulating a pair of GTX 260, with a total of 480 processor cores and 1.8 GB GDDR3 of memory.

We compared the CPU and GPU implementations by means of time required to integrate the ODE systems (Table 2) as well as to solve the parabolic system (Table 3). According to Table 2, the GPU achieved much better performance than the CPU in all tissue simulations. Speedup factors of about 48-fold, 50-fold and 51-fold were achieved in the 2x2 mm, 4x4 mm and 8x8 mm tissue preparations, respectively. Regarding the parabolic system, the GPU was 2 times faster than the CPU to simulate the smaller tissue, then it was 3.6 faster and finally 4.5 times faster to simulate the largest preparation.

Table 2. Comparison between CPU and GPU solvers for the ODE systems. Execution times are given in seconds.

Mesh	$\mathbf{CPU_{Time}}$	$\mathbf{GPU_{Time}}$	**Speedup**
$2 \times 2\,mm$	379.67	7.78	48.80
$4 \times 4\,mm$	1509.59	29.70	50.82
$8 \times 8\,mm$	6005.06	117.57	51.07

Table 3. Comparison between CPU and GPU solvers for the parabolic problem. Execution times are given in seconds.

Mesh	$\mathbf{CPU_{Time}}$	$\mathbf{GPU_{Time}}$	**Speedup**
$2 \times 2\,mm$	323.08	158.31	2.04
$4 \times 4\,mm$	1820.72	493.74	3.68
$8 \times 8\,mm$	8442.98	1888.38	4.68

Table 4 gives the RRMS norm related to a solution calculated using double precision. It can be noticed that the precision used does not influenced the numerical solutions.

6 Discussion

We have evaluated the performance of CPU and GPU solvers for the monodomain model developed in standard C and extended to the NVIDIA CUDA

Table 4. Error values for simulations using single point precision on the CPU and GPU solvers.

Mesh	$\mathbf{CPU_{Float}}$	$\mathbf{GPU_{Float}}$
$2 \times 2\,mm$	0.00607%	0.00458%
$4 \times 4\,mm$	0.02813%	0.02204%
$8 \times 8\,mm$	0.00485%	0.00394%

parallel environment. Three different *in-silico* tissue preparations were used in this work for the performance tests. In all cases, the GPU performed much better than the CPU during the integration of the ODE systems, where a speed increase of about 50 times was obtained. On the other hand, a more modest speedup of about 3.5 was achieved during the solutions of the parabolic problem. The performance of the parabolic solver, which consists essentially of a sparse matrix vector multiplication and subsequent solution of a linear systems, is strongly dependent on the storage format of the sparse matrix associated to linear system. We have employed the well known CSR format, which is used to store general sparse matrices. However, from the point of view of parallelization using CUDA, the CSR format has a significant drawback. In this format, the threads do not access the CSR Aj and Ax arrays contiguously. A better implementation could make use of the fact that the matrices are diagonal structured and thus leverage the memory access by the threads. In this work, all results were obtained using single point precision operations. To make sure that there were no deviations in the numerical solutions, we compared the results against one computed using double precision. Table 4 reveals that the precision of the arithmetic operations does not affect the numerical solutions.

Many efforts have been made to solve efficiently the equations modeling the cardiac electrical activity on a single CPU as well as in cluster of computers [3, 5, 6, 8, 12]. However, the CUDA environment represents a new way to develop parallel applications. The results presented in this work showed that the GPUs are a promising alternative to simulate the electrical activity in the heart. The GPU allows the increasing of the complexity of the problem that can be modeled, and also represent a parallel system with high performance computing for a relative low cost when compared to a cluster of CPUs.

Some important limitations of this work are: 1- the fact that only one cell model was used to draw the conclusions, whereas many other models with different numerical characteristics are available in the literature; 2- there are more suitable solvers for the system of ODEs.

Acknowledgments

This work was supported by the grant P19993-N15 and SFB grant F3210-N18 from the Austrian Science Fund (FWF). Dr. Weber dos Santos thanks the support provided by FAPEMIG, CNPq and CAPES.

References

1. G. Plank, L. Zhou, J. L. Greenstein, S. Cortassa, R. L. Winslow, B. O'Rourke, N. A. Trayanova: From mitochondrial ion channels to arrhythmias in the heart: computational techniques to bridge the spatio-temporal scales. Philos Transact A Math Phys Eng Sci **366** (2008) 3381–409
2. R. Plonsey, R. C. Barr: Mathematical modeling of electrical activity of the heart. Electrocardiol **20**(3) (1987) 219–26
3. M. C. Maclachlan, J. Sundnes, R. J. Spiteri: A comparison of non-standard solvers for odes describing cellular reactions in the heart. Comput Methods Biomech Biomed Engin **10** (2007) 317–326
4. D. B. Geselowitz, W. T. Miller: A bidomain model for anisotropic cardiac muscle. Ann Biomed Eng **11** (1983) 191–206
5. E. J. Vigmond, F. Aguel, N. Trayanova: Computational techniques for solving the bidomain equations in three dimensions. IEEE Trans. Biomed. Eng. **49**(11) (2002) 1260–1269
6. R. W. Santos, G. Plank, S. Bauer, E. J. Vigmond: Parallel multigrid preconditioner for the cardiac bidomain model. IEEE Trans Biomed Eng **51**(11) (2004) 1960–1968
7. E. J. Vigmond, R. W. Santos, A. J. Prassl, M. Deo, G. Plank: Solvers for the cardiac bidomain equations. Prog Biophys Mol Biol. **96**(1-3) (2008) 3–18
8. G. Plank, M. Liebmann, R. W. Santos, E. J. Vigmond, G. Haase: Algebraic multigrid preconditioner for the cardiac bidomain model. IEEE Trans. Biomed. Eng. **54**(4) (2007) 585–596
9. J. Sundnes, B. F. Nielsen, K. A. Mardal, X. Cai, G. T. Lines, A. Tveito: On the computational complexity of the bidomain and the monodomain models of electrophysiology. Ann. of Bio. Eng. **34**(7) (2006) 1088–1097
10. J. Clark, R. Plonsey: A mathematical evaluation of the core conductor model. Biophys J **6**(1) (1966) 95–112
11. M. Potse, B. Dubé, A. Vinet, R. Cardinal: A comparison of monodomain and bidomain propagation models for the human heart. IEEE Trans. Biomed. Eng. **53** (2006) 2425–2435
12. J. Sundnes, G. Lines, A. Tveito: Efficient solution of ordinary differential equations modeling electrical activity in cardiac cells. Mathematical Biosciences **172** (2001) 55–72
13. NVIDIA Corporation: NVIDIA CUDA Programming Guide (2009)
14. C. Luo, Yoram Rudy: A Dynamic Model of the Cardiac Ventricular Action Potential. Circulation Research **74**(6) (1994) 1071–1096
15. J. Sundnes, G. T. Lines, A. Tveito: An operator splitting method for solving the bidomain equations coupled to a volume conductor model for the torso. Mathematical Biosciences **194** (2005) 233–248
16. Y. Saad: Iterative Methods for Sparse Linear Systems. PWS Publishing Company (1996)
17. B. J. Roth: Electrical conductivity values used with the bidomain model of cardiac tissue. IEEE Trans. Biomed. Eng. **44**(4) (1997) 326–328
18. L. Buatois, G. Caumon, B. Lévy: Concurrent number cruncher: an efficient sparse linear solver on the gpu. LNCC **4782** (2008) 358–371