Blucher Mechanical Engineering Proceedings
May 2014, vol. 1 , num. 1
www.proceedings.blucher.com.br/evento/10wccm

GPU accelerated reconstruction of Electrical Impedance Tomography Images through Simulated Annealing

T. C. Martins[1], J. M. Kian[2], D. K. Yabuki[2], M. S. G. Tsuzki[2]

[1] Department of Mechatronics, Polytechnic School at the University of São Paulo (thiago@usp.br)

[2] Department of Mechatronics, Polytechnic School at the University of São Paulo

Abstract.

EIT image reconstruction may be performed by a Simulated Annealing algorithm that minimizes the differences between the superficial impedance behavior of a virtual body simulated using the Finite Element Method and real data acquired on a physical body. The evaluation of objective functions - that involve solving FEM linear systems - is responsible for the majority of the process computational cost. This work presents a strategy for implementing the Preconditioned Conjugate Gradient algorithm on a GPU in order to benefit from its massive parallel computing capacities. This strategy takes in account the specificities of the EIT reconstruction through SA. It involves heavy preprocessing to identify the computations that may be performed in parallel. Initial results show that this strategy greatly improves not only on sequential approaches, but also on other generic GPU approaches.

.

Keywords: *Electrical Impedance Tomography, Preconditioned Conjugated Gradients, Parallel Processing, GPU.*

1. INTRODUCTION

This work deals with the acceleration of the reconstruction of Electrical Impedance Tomography (EIT) images using Graphic Processing Units. Section 2 will present a brief introduction to Electrical Impedance Tomography, its applications and reconstruction techniques. Section 3 will deal with the so-called "forward problem", that is, the problem of determining superficial impedance behaviour when the interior conductivity is known, and how it can be numerically solved through the Finite Element Method (FEM). Section 4 will present briefly Simulated Annealing, the probabilistic optimization metaheuristic used here to reconstruct EIT images by solving forward problems. Section 5 will present the preconditioned conjugated gradients method, used here to solve the linear systems posed by the Finite Element Method, and the main object of paralization in this work. Section 6.1 will present the CUDA architecture. Section

2. Electrical Impedance Tomography

Electrical impedance tomography (EIT) is a noninvasive imaging technique that estimates the electrical conductivity distribution within a body when low amplitude current patterns are applied to its surface and the corresponding electric potentials at determined points of that surface are measured [19]. EIT has a wide range of medical applications, detection of acute cerebral stroke [2], breast cancer [10], monitor cardiac activity [4] and monitor lung aeration imposed by mechanical ventilation in critically ill patients [9, 19].

The reconstruction of the electrical conductivity distribution within the body is a fundamentally dificult problem since, in a general case, the electric current cannot be forced to flow linearly (or even along a known path) in an inhomogeneous conductor. As such, impedance between two known points on the surface of a body is a composition of the whole conductivity distribution on the interior, often in an unpredictable way.

2.1. EIT as an optimization problem

A popular method for solving the reconstruction problem is to look at it as an optimization problem[14], where one tries to maximize the superfitial impedance behaviour of a virtual body and the actual measured impedance data. In this problem, the the optimization variables are a parametrization of the conductivity inside the domain (for instance, the domain is divided in continuous segments and the conductivity inside each segment — presumed to be constant — becomes a parameter) and the optimization function is some measure of how a computer simulation of the domain — using the same current patterns applied to the physical body — measured data. One possible objective function E is the Euclidean distance between the measured electric potentials ϕ_m^i and the calculated potentials ϕ_c^i obtained as results of simulations on the virtual body for all the applied current patterns

$$E = \sqrt{\Sigma |\phi_m^i - \phi_c^i|^2}. \tag{1}$$

An example of such approach is in [14], where the objective function 1 is minimized by Sequential Linear Programming yielding estimations of the conductivity distribution. First and second order optimization methods cannot be easily applied to this problem though, since the underlying objective function, calculated from the results of a computer simulation, is subject to numerical errors that are amplified on its derivatives. For that reason, some authors are turning towards zero order methods that require no gradient, and in particular, metaheuristics such as Simulated Annealing [8, 13]. The downside of this approach is that it requires many more objective function evaluations, and, as it will be shown in section 3, those can be quite costly.

3. The forward problem

The typical forward problem in EIT is of, for a given domain Ω with conductivity distribution σ and currents J injected through the domain boundary, finding the potential distribution ϕ within Ω and in particular the resulting potentials at the measurement electrodes ϕ_m. The frequencies used in EIT are low enough so that the quasi-static approximation holds, and thus capacitive and inductive effects can be ignored. Under such quasi-static conditions, the solution of the forward problem is rather simple as it only requires solving the equation

$$\nabla \left(\sigma \nabla \phi \right) = 0. \tag{2}$$

That is an elliptic equation in the divergence form [3, ch. 6]. The boundary conditions are given by currents J at the domain frontier

$$\sigma \frac{\partial \phi}{\partial \hat{n}} = J \tag{3}$$

where $\hat{n}$ is the external normal. The convention adopted here is to have positive J values for currents *exiting* the domain. First-derivative boundary conditions such as the formulated in (3) are called *Neumann boundary conditions*. The forward problem with Newmann boundary conditions is particular to scenarios in which the boundary currents are known. Another possible scenario is that of known boundary potentials, leading to *Dirichlet boundary conditions*. Concerning boundary conditions, it is important to notice that integrating (2) over the domain Ω and applying Gauss' theorem,

$$\oint_{\partial\Omega} \sigma \nabla \phi \cdot \hat{n} \, \mathrm{d}s = 0. \tag{4}$$

Replacing (3) in (4),

$$\oint_{\partial\Omega} J \, \mathrm{d}s = 0$$

that is a necessary condition for $(2, 3)$ to have a solution. Physically, it is equivalent to requiring that the sum of currents entering and leaving the domain is zero.

It is easy to notice that if a given ϕ^* is a solution of $(2, 3)$, then so is $\phi^* + k$, where k is a constant. This means the problem is undefined minus a constant, which is hardly surprising, as electric potential is a relative measure. The indetermination is often circumvented in EIT problems by adopting a *ground point* x_0 such that

$$\phi\left(x_0\right) = 0. \tag{5}$$

The point x_0 is often taken at the domain boundary.

3.1. Finite Element Method applied to the Forward problem

For an irregular domain and isotropic media, analytical solution to the equation (2) with boundary condition (3) are unknown; thus, the partial differential equations were approximated by the Finite Element Method (FEM). The procedure for obtaining a FEM formulation for the EIT forward problem is the roughly the same as for a regular Poisson equation [1]. A variational formulation of equations (2, 3) is produced. The potential function ϕ is expressed as a linear combination of a base of elementary functions with compact support, transforming the variational problem on a quadratic optimization problem. Analytical solutions for (2, 3) for arbitrary domains Ω are not known. An approximated solution of $\tilde{\phi}$ may be obtained by a Finite Element (FEM) formulation. The procedure for obtaining a FEM formulation for the EIT forward problem is the roughly the same as for a regular Poisson equation [1]. A variational formulation of (2, 3) is produced. Then the approximated solution $\tilde{\phi}$ is chosen from a linear subspace generated by a set $\{\psi_1, \psi_2, \ldots, \psi_n\}$ of base functions. By writing

$$\tilde{\phi} = \mathbf{\Phi}^T \cdot \mathbf{\Psi} \tag{6}$$

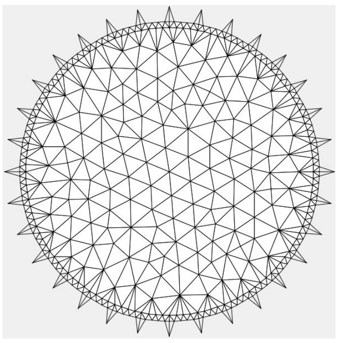

Figure 1. Mesh used in the image reconstruction.

$$\boldsymbol{\Psi} = [\psi_1, \psi_2, \ldots, \psi_n]^T$$

replacing it on the variational formulation and finding its minimum on Φ the equation

$$\boldsymbol{K}(\sigma) \cdot \boldsymbol{\Phi} - \boldsymbol{C} = 0 \tag{7}$$

is obtained, where

$$K(\sigma)_{ij} = \int_\Omega \sigma \left(\nabla\psi_i \cdot \nabla\psi_j\right) \mathrm{d}x \tag{8}$$

is the *stiffness matrix*, a symmetric positive definite matrix and $\boldsymbol{C} = \oint_{\partial\Omega} \boldsymbol{\Psi} \cdot \boldsymbol{J} \, \mathrm{d}s$ is the *load vector*. By solving the linear system (7) for $\boldsymbol{\Phi}$, one obtains the approximate solution $\tilde{\boldsymbol{\phi}}$.

3.1.1 Base functions

The base functions $\boldsymbol{\Psi}$ are chosen so that $\tilde{\boldsymbol{\phi}}$ is continuous, with support in the whole domain and the integrals in 8 are easily calculated. In this particular project, the domain (a cylindrical container, modeled here as a $2d$ problem) is divided in triangular segments, as seen in figure 1. To each node is assigned a ϕ_i base function, so that its value at that node is 1 and zero at all the others. Inside the triangles, the base functions are a 1st degree interpolation of the values at the vertexes. Under those base functions it can be seen that $K(\sigma)$ is a very sparse matrix. Indeed, if i and j are two vertexes that do not share a triangle, $K(\sigma)_{ij} = 0$. By equation (8), one can see that even if i and j are edges of the same triangle, it is possible to have $K(\sigma)_{ij} = 0$, but this is a particular case, uncommon on irregular meshes and can be ignored here without loss of generality. As such, if one would look at the mesh of figure 1 as a graph, the matrix K would have the same zero/nonzero structure than its adjacency matrix.

4. Simulated Annealing

Simulated Annealing (SA) [12] is a hill-climbing local exploration optimization heuristic, which means it can skip local minima by allowing the exploration of the space in directions that lead to a local increase on the cost function. It sequentially applies random modifications on the evaluation point of the cost function. If a modification yields a point of smaller cost, it is automatically kept. Otherwise, the modification also can be kept with a probability obtained from the Boltzman distribution

$$P(\Delta E) = e^{-\frac{\Delta E}{kT}} \tag{9}$$

where $P(\Delta E)$ is the probability of the optimization process to keep a modification that incurs an increase ΔE of the cost function. k is a parameter of the process (analogous to the Stefan–Boltzman constant) and T is the instantaneous "temperature" of the process. This temperature is defined by a cooling schedule, and it is the main control parameter of the process. The probability of a given state decreases with its energy, but as the temperature rises, this decrease (the slope of the curve $P(\Delta E)$) diminishes.

4.1. Applying Simulated Annealing to EIT

As seen in section 2.1, the EIT inverse problem can be formulated as an optimization problem, and as such, can be approached with SA. Indeed, in [8] Herrera et al. applied SA to the minimization of objective function (1) and by doing so managed to reconstruct very accurate conductivity distributions of the body, but at a very high computational cost. This is unsurprising, as each step of the SA involves the solution of a full FEM problem in order to evaluate the objective function.

The application of SA in this work is derived from the work of Martins et al. [13]. In that work, the conductivity is parametrized using the same base functions described in 3.1.1, that also are used for the electric potential, This greatly simplifies the calculation of the integrals in (8) since, inside a triangle, the conductivity may be replaced by a constant equal to the average of the conductivity at each node. At each iteration, SA modifies the conductivity of at most *two* nodes. As seen from (8), if the conductivity in node k is modified, a coefficient K_{ij} will be modified only if i and j share an edge with k.

5. The Conjugated Gradients Algorithm

The Conjugated Gradients Algorithm is a methodo for obtaining the numerical solution of linear systems like

$$Ax - b = 0 \tag{10}$$

where A is a symmetric positive definite matrix. That makes it particularly suitable for solving the linear systems (7) posed by the FEM method described particularmente apropriado in section 3.1.

It is an iterative process that starts from an initial solution x_0 and at each iteration i produces a new solution x_i successively closer to the exact solution. In exact arithmetic, it can be shown that the algorithm recovers the exact solution in at most n iterations, where n is the rank of A [18]. In general, it is computationally inviable to work with exact arithmetic. Even so, the method is numerically stable and able to find acceptable numerical solutions with finite precision arithmetic [15, cap.8].

The algorithm iterations are as follows: (adapted from [18]):

$$d_0 = r_0 = b - Ax_0$$

$$\alpha_i = \frac{{r_i}^T r_i}{{d_i}^T A d_i} \tag{11}$$

$$x_{i+1} = x_i + \alpha_i d_i \tag{12}$$

$$r_{i+1} = r_i - \alpha_i A d_i \tag{13}$$

$$\beta_{i+1} = \frac{{r_{i+1}}^T r_{i+1}}{{r_i}^T r_i} \tag{14}$$

As one can see, the most expensive operation in a iteration is the matrix $\times$ vector product in (11) (Notice it may be reused in (13)).

5.1. Convergence and preconditioning

Let ϵ_k the error at the k-th iteration, and $\|\epsilon_k\|_A = {\epsilon_k}^T A \epsilon_k$ its A-norm.

Than it can be shown that ([18]):

$$\|\epsilon_k\|_A \leq 2 \left(\frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1} \right) \|\epsilon_0\|_A \tag{15}$$

Where $\kappa = \lambda_{max}/\lambda_{min}$ is the *condition number* of A, defined as the ratio between its largest and smallest eigenvalues.

Let now M be a symmetric positive definite matrix that approaches A. One expects that the eigenvalues of $M^{-1}A$ are closer than those of A and its condition number is smaller. As such, the convergence of GC applied to the system

$$M^{-1}Ax - M^{-1}b = 0 \tag{16}$$

will be faster. Notice that the solution of system (16) is the same as that of system (10).

The issue is that $M^{-1}A$ may not be symmetric nor positive definite, so it may not be possible to apply the Conjugated Gradient Method to system (16).

Now suppose it is possible to find a decomposition of matrix $M = EE^T$. In this case, the eigenvalues of $M^{-1}A$ and $E^{-1}AE^{-T}$ are the same, but this last matrix is symmetric, enabling the application of CG to the system.

$$E^{-1}A\hat{x}E^{-T} - E^{-1}b = 0 \tag{17}$$

and

$$\hat{x} = E^T x$$

The Preconditioned Conjugated Gradients Algorithm is (adapted from [18]):

$$r_0 = b - Ax_0$$

$$Mz_0 = r_0$$

$$d_0 = z_0$$

$$\alpha_i = \frac{{r_i}^T z_i}{{d_i}^T A d_i} \tag{18}$$

$$x_{i+1} = x_i + \alpha_i d_i \tag{19}$$

$$r_{i+1} = r_i - \alpha_i A d_i \tag{20}$$

$$Mz_{i+1} = r_{i+1} \tag{21}$$

$$\beta_{i+1} = \frac{{z_{i+1}}^T r_{i+1}}{{z_i}^T r_i} \tag{22}$$

$$d_{i+1} = z_{i+1} + \beta_{i+1} d_i \tag{23}$$

The calculation of z_{i+1} in step (21) requires the solution of a whole linear system! It is thus fundamental that $M^{-1}r_{i+1}$ may be calculated quickly, since this operation will be executed once per iteration of the Conjugated Gradients Algorithm (clearly a generic matrix M is unacceptable, since the cost of a single iteration of the method would be equivalent of solving a whole system).

There are several canditates for a suitable M. In [14] and [13] the incomplete Choleksy Decomposition was used with Preconditioned Conjugated Gradients for reconstruction of EIT images.

The Incomplete Cholesky Decomposition is an approximated LL^T decomposition for sparse symmetric positive definite matrices. It produces for a given matrix A a lower triangular matrix E such that $A \approx EE^T$ and $A_{i,j} = 0 \Rightarrow E_{i,j} = 0$, that is, E keeps the sparse structure of A. Incomplete Choleksy Decomposition may be performed by algorithm 1.

Algorithm 1 Incomplete Cholesky Decomposition (Reproduced from [6]).

```
 1: function INCOMPLETECHOLESKY($A$)
Input: $A$ is an $n \times n$ matrix
Output: A lower triangular $E$ that is the Incomplete Cholesky Decomposition of $A$
 2:     Initialize $E$ with the lower triangular part of $A$
 3:     for $k = 1$ to $n$ do
 4:         $E_{k,k} \leftarrow \sqrt{E_{k,k}}$
 5:         for $i = k + 1$ to $n$ do
 6:             if $E_{i,k} \neq 0$ then
 7:                 $E_{i,k} \leftarrow E_{i,k} / E_{k,k}$
 8:             end if
 9:         end for
10:         for $j = k + 1$ to $n$ do
11:             for $i = j$ to $n$ do
12:                 if $E_{i,j} \neq 0$ then
13:                     $E_{i,j} \leftarrow E_{i,j} - E_{i,k}E_{j,k}$
14:                 end if
15:             end for
16:         end for
17:     end for
18:     return $E$
19: end function
```

When using $M = EE^T$ from the Incomplete Cholesky Decomposition, the calculation of z_{i+1} in step (21) is equivalent of solving two sparse triangular systems.

6. Implementation in GPU

6.1. The CUDA architecture

GPUs are nowadays specialized processors with huge computing capability. Their role in computer graphics is mainly to stock big and complex sets of triangles in an ideal three dimensional environment, to assemble several of them as the CPU commands. In the last years GPUs grew capable of doing the most amazing graphic effects, like blur for foggy environment, transparencies for glasses, windows and liquids, reflections as well as partial reflections, for water effects. As the possibilities increased, instead of producing different hardware for each effect, the graphic units manufacturers started conceiving programmable graphic units, which could run arbitrary user defined programs in order to manipulate images in some of the key steps of the graphic rendering [16].

Recent graphic processing units are able of generic massively parallelized computations. The CUDA architecture [11] (see figure 2), created by NVIDIA, implements what is called the SIMT (Single Instruction Multiple Thread) architecture, that is, all processing nodes are expected to execute the same instruction at the same time. This is a similar to the classic SIMD (Single Instruction Multiple Data) architecture where the same operation is applied to a data set. The differences between SIMT and SIMD is that SIMT is able to accommodate unstructured data (although performance increases with regular data accesses) and SIMT can handle *thread divergence*, that is, threads running simultaneously that execute different code. CUDA architecture handles thread divergence by transparently allocating different fetch cycles for processing nodes that are executing different instructions. Such divergence can, however, incur severe performance reductions.

A typical CUDA computation involves the execution of a single function, called a *kernel* across many processing nodes such that each node receives slightly different parameters, its *grid coordinates*. Since every node is executing exactly the same kernel, it relies on its grid coordinates for differentiation.

6.2. Accelerating the objective function evaluation with CUDA

As stated in section 4.1, the evaluation of the objective function is responsible for almost all the computational cost of the EIT image reconstruction using Simulated Annealing. Inspecting this evaluation, it is clear that the computational power is spent on CG iterations and Preconditioner calculations. Within CG, the most expensive operations are the product Sparse Matrix $\times$ Vector and particularly, the two triangular solve operations. It then our priority to accelerate using the GPU the following operations:

- Sparse Matrix $\times$ vector product
- Triangular Solver
- Incomplete Choleksy Decomposition

while keeping in mind that the structure of the stiffness matrix $K(\sigma)$ *do not change* between SA iterations, that is, its zeroed coefficients remain so for every σ produced by SA.

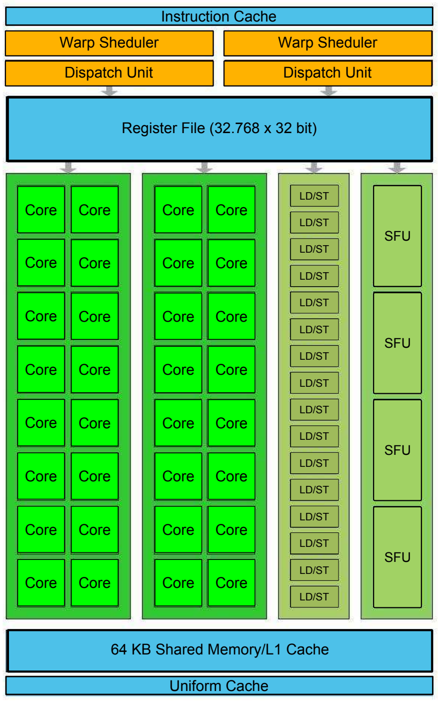

Figure 2. Architecture of the GPU streaming multiprocessor.

6.3. Sparse Matrices

As seen in section 3.1.1, the stiffness matrices $K(\sigma)$ produced by the Finite Element Method are very sparse, that is, they have most of its coefficients zeroed.

There are several schemes for efficiently storing sparse matrices. The scheme adopted here is CSR (Compressed Sparse Row). It stores a sparse matrix with three vectors, here called *data*, *column* and *ptr*. The vector *data* stores the nonzero elements of the matrix. The vector *column* stores at the corresponding position the number of the column for the element. The {ptr vector has as many entries as M has rows, and stores at its i-th entry the position in *data* of the first nonzero element at the i-th row of M (see figure 3 for an example).

Since, as mentioned in section 6.2, the structure of matrix $K(\sigma)$ is *constant*, its *column* and *ptr* vectors may be stored in the *constant memory* of the CUDA GPU. The constant memory

$$\begin{bmatrix} 1 & 0 & 2 & 0 & 0 \\ 0 & 7 & 1 & 0 & 0 \\ 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 4 & 8 \\ 9 & 0 & 0 & 0 & 7 \end{bmatrix}$$

$$ptr = \begin{bmatrix} 0 & 2 & 4 & 5 & 7 \end{bmatrix}$$

$$data = \begin{bmatrix} 1 & 2 & 7 & 1 & 3 & 4 & 8 & 9 & 7 \end{bmatrix}$$

$$column = \begin{bmatrix} 0 & 2 & 1 & 2 & 2 & 3 & 4 & 0 & 4 \end{bmatrix}$$

Figure 3. Example of CSR matrix storage

is a special memory location on the GPU used to store data that may not be changed by a kernel. It may in exchange be aggressively cached by the CUDA multiprocessors, greatly increasing its data throughput.

6.4. Sparse Matrix $\times$ Vector product

While the Sparse Matrix $\times$ Vector product is computationally cheaper than the Triangular Solver, it still caries a considerable cost. Fortunately, the its implementation is very simple under a CSR representation. Algorithm 2 describes its implementation on a serial (non-parallel) processor.

Algorithm 2 CSR matrix $\times$ vector multiplication.

```
Input: Vectors $data$, $column$ and $ptr$ of the sparse matrix $A_{m\times n}$, vector $x$
Output: a vector $y$ such that $y = Ax$
1: for $i = 0$ to $m - 1$ do
2:     $y[i] \leftarrow 0$
3:     for $l = ptr[i]$ to $ptr[i + 1] - 1$ do
4:         $y[i] \leftarrow y[i] + data[l] \cdot x[column[l]]$
5:     end for
6: end for
```

The algorithm 2 can be trivially parallelized. Indeed, it suffice to send each line of matrix A to a kernel. The grid coordinates are set up to match the range of i variable in the outer loop. The kernel is just algorithm 2 stripped of its outer vector.

Algorithm 3 CSR matrix $\times$ vector Kernel.

```
Input: Vectors $data$, $column$ and $ptr$ of the sparse matrix $A_{m\times n}$, vector $x$ and the grid
    coordinate $i \in \{0..m - 1\}$
1: $y[i] \leftarrow 0$
2: for $l = ptr[i]$ to $ptr[i + 1] - 1$ do
3:     $y[i] \leftarrow y[i] + data[l] \cdot x[column[l]]$
4: end for
```

At the end of the grid execution of the kernel depicted in 3, vector y will contain the product Ax. While the sum operations in line 3 could also be parallelized using a round-based reduction [11, sec. 6.1], the small number of non-zero elements at each line does not justify it (see figure 1). Thread divergence isn't a serious issue either, since there's not a huge difference in the number of nonzero elements at each row.

Indeed, this almost trivial kernel is able to outperform in our particular case the one provided by NVIDIA for CSR matrix $\times$ vector operations in CUBLAS [17] (CUBLAS outperforms it for generic sparse matrices).

6.5. Triangular Solver

The two triangular linear system solutions, performed to obtain the value of z_{i+1} in step (21) (remembering that $M = EE^T$ from the Incomplete Cholesky Decomposition), are the most costly operations performed on a iteration of the Preconditioned Conjugated Gradients Algorithm.

Lower triangular systems like $Lx = y$ can be solved using forward substitutions [6, Ch. 3](see Algorithm 4). An analogous algorithm for upper triangular systems $Ux = y$ is called back-substitution.

Algorithm 4 Serial Forward substitution.

```
Input: A lower triangular matrix $L$ and a vector $y$
Output: a vector $x$ such that $Lx = y$
1: $x[0] = b[0]/L[0,0]$
2: for $k = 1$ to $n - 1$ do
3:     $\sigma = 0$
4:     for $j = 0$ to $k - 1$ do
5:         $\sigma = \sigma + L[k,j]x[j]$
6:     end for
7:     $x[k] = (b[k] - \sigma)/L[k,k]$
8: end for
```

A trivial parallelization of the outer loop in algorithm 4 is no longer viable, since the operations at line 5 require the previously computed values of $x[j]$. At least, the (costly) floating point division operations in line 7 can be eliminated by taking a D as a diagonal matrix containing the diagonal of L and observing that the solution of $D^{-1}Lx = D^{-1}y$ is the same of $Lx = y$. By replacing L by $D^{-1}L$ and y by $D^{-1}y$ in algorithm 4, there's no longer need for any floating point division (notice that the diagonal of $D^{-1}L$ has only ones). The calculations of $D^{-1}L$ and $D^{-1}y$ can be trivially parallelized using the GPU. Even better, D^{-1} can be precomputed only once for all CG iterations for all simulated current patterns at a SA iteration, leaving only the much faster floating point multiplications to be performed by the GPU.

On the parallelized execution of the substitution step, the matrices L in this case are the result of an Incomplete Cholesky Decomposition, and as consequence have the same sparse structure of the system stiffness matrix K. As such, the operations in line 7 of algorithm 4 do not depend on all previously calculated $x[j]$, but on only those for which $L[k,j]$ is nonzero. One can create a *substitution schedule* for a parallel algorithm in order to . A *substitution schedule* is a sequence of sets $S = \{s_1, s_2, \ldots, s_n\}$ where s_i is a set of indexes k for which the values of $x[j]$ in line 5 are already calculated. A greedy algorithm can be written to cerate a substitution schedule

Algorithm 5 Serial Forward substitution.

```
Input: A lower triangular matrix $L_{m\times m}$
Output: a substitution schedule $S = \{s_1, s_2, \ldots, s_n\}$
1: $A \leftarrow \{0, 1, \ldots, m-1\}$
2: $B \leftarrow \emptyset$
3: $i \leftarrow 1$
4: while $A \neq \emptyset$ do
5:     $d_i \leftarrow \{k \in A \mid \forall j < k, L[k,j] \neq 0 \Rightarrow j \in B\}$
6:     $B \leftarrow B \cup s_i$
7:     $A \leftarrow A \setminus s_i$
8:     $i \leftarrow i + 1$
9: end while
```

The algorithm 5 greedily computes a substitution schedule S by taking at each iteration the set of rows of L that have nonzero entries only at positions that correspond to already taken rows. The algorithm finishes when there's no more rows to take.

Algorithm 6 Serial Forward substitution Kernel.

```
Input: A lower triangular matrix $L$, a vector $y$, a substitution schedule $S$ and a grid coordinate $i$
Output: a vector $x$ such that $Lx = y$
1: for $r = 1$ to $n$ do
2:     if $i + 1 \leq |s_r|$ then
3:         $k \leftarrow i$ -th element of $s_r$
4:         for $j \in \{j \in j < k \mid L[k,j] \neq 0\}$ do
5:             $\sigma = \sigma + L[k,j]x[j]$
6:         end for
7:         $x[k] = b[k] - \sigma$
8:     end if
9: end for
```

The algorithm 6 describes the kernel that performs parallel forward substitution using a substitution schedule. The grid must be set up so that there's as much threads as the cardinality of the largest s_i. Notice that thread divergence occurs in line 2 (when there are more threads running than the cardinality of s_r, meaning that some threads have nothing to work on) and in line 4 where the number of nonzero entries may vary from row to row. The sets in line 4 may be precomputed along with the substitution schedule. Also all index references may be transformed to positions in the *data* vector of the sparse representation of L. Since the structure of the matrix L do not change (as it follows the immutable structure of K), this preprocessing need to be performed only *once* for all CG iterations of all simulated current patterns of all SA iterations, and this data may be stored in the *constant memory* of the GPU.

Unfortunately, initial tests with the approach above reveal a disappointing performance. Indeed, the performance of the GPU triangular solvers is so weak that it almost nullifies the benefits of having a preconditioner in the first place. A closer inspection reveals that the original structure of L allows for very little parallelization.

6.5.1 Node Numbering

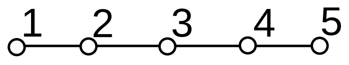

Figure 4. FEM model with 5 nodes numbered sequentially.

The structure of L, ill-suited for parallel forward substitution, is a result of a naive numbering scheme of the nodes of the FEM mesh. Indeed, consider the simple unidimensional FEM model with 5 nodes in figure 4. Its nodes are numbered sequentially, and the corresponding lower triangular portion of its stiffness matrix has the following structure:

$$\begin{pmatrix} * & & & & \\ * & * & & & \\ & * & * & & \\ & & * & * & \\ & & & * & * \end{pmatrix}$$

The substitution schedule produced by algorithm 5, when applied to such a matrix, would produce the following result:

$$S = \left\{ \begin{array}{c} \{1\} \\ \{2\} \\ \{3\} \\ \{4\} \\ \{5\} \end{array} \right\}$$

This substitution schedule allows for absolutely no parallelization at all. A parallel kernel performing under this schedule would run on a single thread, performing exactly the same steps of the serial algorithm 6.

Now consider a coloring of the graph in figure 4 so that to each node is assigned a color that is different from those of its neighbors. It is easy to show that the nodes of an unidimensional model can be colored using only two colors.

If the nodes of this graph would be numbered *color-wise*, that is, each color has its nodes numbered in sequence, the resulting graph would be like in figure 5.

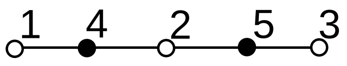

Figure 5. FEM model with colored nodes and numbered color-wise.

The corresponding lower triangular structure of the stiffness matrix would be

$$\begin{pmatrix} * & & & & \\ & * & & & \\ & & * & & \\ * & * & & * & \\ & * & * & & * \end{pmatrix}$$

and now the substitution schedule would be.

$$S = \left\{ \begin{array}{c} \{1,2,3\} \\ \{4,5\} \end{array} \right\}$$

This substitution schedule allow for the processing of the triangular solver in just two passes of algorithm 6, exactly the number of colors used in the coloring of the mesh graph.

It can be shown that a planar graph like the mesh used on our model can be colored using at most 4 colors. In this work, an algorithm from [5] was used. This particular algorithm guarantees to use at most 5 colors, but often is capable of better. Indeed, in figure 7 there's the structure of the preconditioner after the mesh used here was colored with that algorithm. Only four colors were used, and the triangular solver requires only four passes (the original matrix structure is in figure 6).

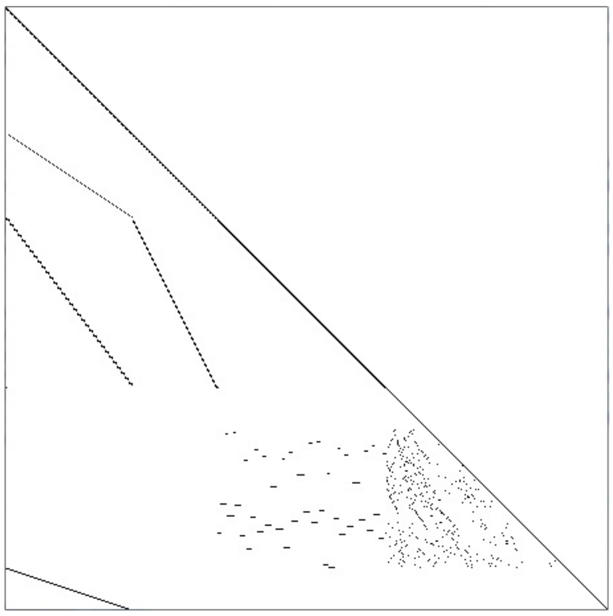

Figure 6. Preconditioner structure for the original numbering scheme

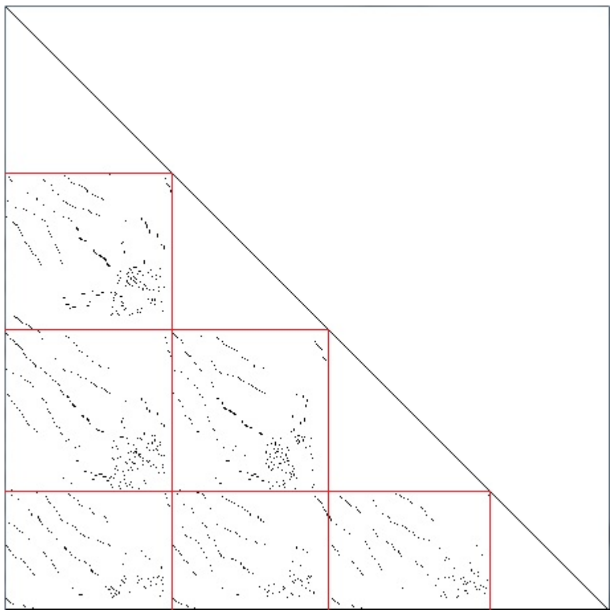

Figure 7. Preconditioner structure for the collored graph

The performance impact is huge. Indeed, benchmarks show that the triangular solver performs more than twenty times faster with the new numbering scheme. The graph coloring step can be performed in a preprocessing step and its cost is negligible.

6.6. Preconditioner

While the preconditioner calculation is performed only once per SA iteration, it is still interesting to perform it inside the GPU for two reasons:

- It is a quite computationally expensive step, more costly than a whole CG iteration
- Bandwith between host computer and GPU is limited, so the upload of a preconditioner produced by the CPU to the GPU at each SA iteration would impact negatively the reconstruction speed.

As seen in algorithm 1, the Incomplete Cholesky Decomposition can be performed column-wise by the repeated iteration of three steps:

1. Calculate the square root of the first element of the current column
2. Divide the whole column by the square root of its first element
3. Propagate the result to next column

the propagation is done on all column for which there's a corresponding non-zero element on the current row. This means that in the calculation of the Incomplete Cholesky Decomposition for matrix K, the j-th column will be affected by the results of the processing of the i-th column only if $K_{i,j} \neq 0$.

The implications of this fact are that the dependency pattern on operations on rows for the Incomplete Choleksy Decomposition are *exactly the same* of those for the rows on the lower triangular solver. As such, the same strategy adopted of section 6.5 can be adopted here. Create a scheduler that greedly constructs a schedule for the three steps above in parallel and create a Kernel that follow those steps. The schedule format, scheduler and kernel (Actually three kernels were implemented, one for each step above) are more complex than those of section 6.5, but roughly the same ideas were followed. As expected, the performance benefits also from the node numbering scheme of section 6.5.1. Even better, since the coloring scheme effectively creates a "propagation hierarchy" (a color can only propagate preconditioner values to the next colors), it is possible to quickly "update" the preconditioner from one SA iteration to another by creating "propagation schedules" for each possible σ modification (this mode was not implemented on this work)

7. Results

The evaluation of the objective function on the reconstruction process performed by Martins et al. in [13] was reproduced here using the same datasets. Since the method in [13] uses an l_2 error based stopping criteria on the evaluation of the objective funtion, it is interesting to compare the evolution of the l_2 error for The Preconditioned Conjugated Gradient Algorithm for our code and the host-only code.

The host code conjugated gradients algorithm uses the Eigen2 library [7], was was compiled with GCC and executed on a intel i7 processor at 3.3 Ghz. The GPU code was

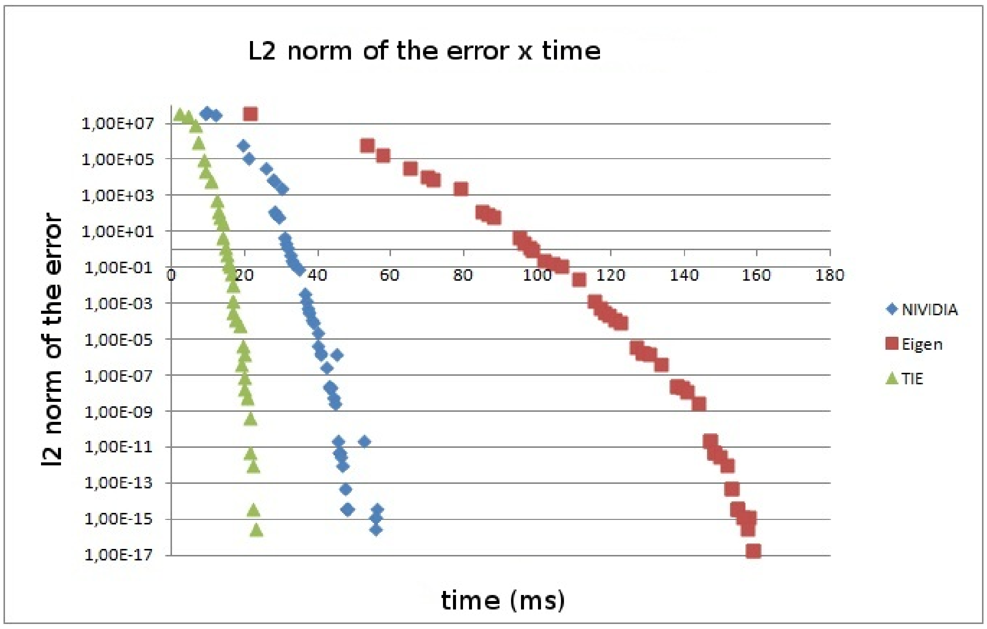

Figure 8. l_2 norm of the error $\times$ time on the objective function evaluation for host-only code (*Eigen*), NVIDA CUSPARSE libraries (*NVIDIA*) and the presented approach (*TIE*).

executed on a NVIDIA GTX 480 GPU. Both used double precision floating point. The comparison also included NVIDIA's CUSPASE library, that has primitives for both Matrix $\times$ vector operations and triangular solver using sparse matrix.

The results are presented in figure 8. Our approach greatly outperforms the host code, presenting an almost tenfold increase in performance. This result is even more impressive if one cosiders the GTX 480 GPU *does not* have the *fast double precision* feature. Our approach also outperforms NVIDIA's CUSPARSE library, showing the merits of optimizing the routines for this particular application.

The Incomplete Cholesky Decomposition also presents a similar tenfold increase in speed when compared to the host-only code.

8. Conclusions and Future work

We proposed here an approach to the parallelization of the evaluation of the objective function for the reconstruction of EIT images using Simulated Annealing. The approach involves identifying at a preprocessing step operations that may be performed in parallel and producing calculation scripts that may be followed by the GPU. The FEM mesh structure used to solve the EIT forward problem greatly affects the throughput of the proposed approach. A node numbering scheme based on graph coloring can be adopted to obtain a more favorable structure.

Future works involve the exploitation of the symmetry of the stiffness matrix K on the matrix $\times$ vector operation and replacement of the full preconditioner calculation by updates that take in account the modifications performed by SA on the matrix K

9. ACKNOWLEDGEMENTS

Thiago C. Martins was supported by FAPESP (Grant 2010/19380–0). Jacqueline M. Kian was supported by FAPESP (Grant 2011/01194–8). Diego K. Yabuki was supported by CNPq. Marcos de Sales Guerra Tsuzuki was partially supported by the CNPq (Grant 309570/2010–7). This research was supported by FAPESP (Grant 2010/19380–0).

10. REFERENCES

[1] Klaus-Jürgen Bathe. *Finite Element Procedures (Part 1-2)*. Prentice Hall, 1995. ISBN 0133014584.

[2] M. T. Clay and T. C. Frree. Weighted regularization in electrical impedance tomography with applications to accute cerebral stroke. *IEEE Transactions on Medical Imaging*, 21: 629–637, 2002.

[3] Lawrence C. Evans. *Partial Differential Equations: Second Edition (Graduate Studies in Mathematics)*, volume 19 of *Graduate Series in Mathematics*. American Mathematical Society, Province, Rhode Island, 2nd edition, 2010. ISBN 0821849743.

[4] B. M. Eyüboglu, B. H. Brown, and D. C. Barber. In vivo imaging of cardiac related impedance changes. *IEEE Engineering Med. Biol. Mag.*, 8:39–45, 1989.

[5] Greg N. Frederickson and Communicated David Gries. On linear-time algorithms for five-coloring planar graphs. *Inform. Process. Lett*, pages 219–224, 1984.

[6] Gene H. Golub and Charles F. Van Loan. *Matrix Computations*. The Johns Hopkins University Press, Baltimore and London, third edition, 199.

[7] Gaël Guennebaud, Benoît Jacob, et al. Eigen v2. http://eigen.tuxfamily.org, 2010.

[8] C. N. L. Herrera, M. F. M. Vallejo, F. S. Moura, J. C. C. Aya, and R. G. Lima. Electrical impedance tomography algorithm using simulated annealing search method. In *Proceedings of International Congress Of Mechanical Engineering*, Brasília, 2007. ABCM.

[9] P. Hua, E. J. Woo, J. G. Webster, and W. J. Tompkins. Finite element modeling of electrode-sking contact impedance in electrical impedance tomography. *IEEE Transactions on Biomedical Engineering*, 40:335–343, 1993.

[10] T. J. Kao, D. Isaacson, J. C. Newell, and G. J. Saulnier. A 3d reconstruction algorithm for eit using a handheld probe for breast cancer detection. *Physiological Measurements*, 27: S1–S11, 2006.

[11] David B. Kirk and Wen-mei W. Hwu. *Programming Massively Parallel Processors: A Hands-on Approach*. Morgan Kaufmann, 2010. ISBN 0123814723.

[12] S. Kirkpatrick, C. D. Gellat, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, 1983.

[13] T.C. Martins, E.D.L.B. Camargo, R. G. Lima, M. B. P. Amato, and M. S. G. Tsuzuki. Electrical impedance tomography reconstruction through simulated annealing with incomplete evaluation of the objective function. In *Proceedings of the 33rd Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBS 11)*, Boston, 2011. IEEE EMBS.

[14] Luís Augusto Motta Mello, Cícero Ribeiro de Lima, Marcelo Britto Passos Amato, Raul Gonzalez Lima, and Emílio Carlos Nelli Silva. Three-dimensional electrical impedance tomography: a topology optimization approach. *IEEE transactions on biomedical engineering*, 55(2 Pt 1):531–40, 2008.

[15] Gérard Meurant. *The Lanczos and Conjugate Gradient Algorithms: From Theory to Finite Precision Computations*. Society for Industrial and Applied Mathematics, 2006.

[16] Rajib Nath, Stanimire Tomov, Tingxing ”Tim” Dong, and Jack Dongarra. Optimizing symmetric dense matrix-vector multiplication on GPUs. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, SC ’11, pages 6:1–6:10, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0771-0.

[17] *CUDA CUBLAS Library*. nVidia Corporation, August 2010.

[18] Jonathan R Shewchuk. An Introduction to the Conjugate Gradient Method Without the Agonizing Pain. Technical report, Carnegie Mellon University, Pittsburgh, PA, USA, March 1994. URL `http://www.cs.cmu.edu/~quake-papers/painless-conjugate-gradient.pdf`.

[19] F. C. Trigo, R. Gonzales-Lima, and M. B. P. Amato. Electrical impedance tomography using the extended kalman filter. *IEEE Transactions on Biomedical Engineering*, 51: 72–81, 2004.