O PROBLEMA DIRETO DA TIE PARALELIZÁVEL EM GPU UTILIZANDO O FORMATO MATRICIAL PJDS COLORIDO PARA MATRIZES ESPARSAS

Renato Seiji Tavares*, Thiago de Castro Martins*, Edson Kenji Ueda*,
André Kubagawa Sato*, Rogério Yugo Takimoto*, Raul Gonzalez Lima*,
Marcos de Sales Guerra Tsuzuki*

*Laboratório de Geometria Computacional
Departamento de Engenharia Mecatrônica e de Sistemas Mecânicos
Escola Politécnica da Universidade de São Paulo
Avenida Prof. Mello Morais, 2231, CEP 05508-030 - São Paulo, SP, Brasil

Emails: reseta@uol.com.br, thiago@usp.br, ueda.edson@gmail.com,
andre.kubagawa@gmail.com, takimotoyugo@gmail.com, lima.raul@gmail.com, mtsuzuki@usp.br

Abstract— Electrical Impedance Tomography (EIT) is an imaging technique that attempts to reconstruct the conductivity distribution inside an object from electrical currents and potentials applied and measured at its surface. This work uses the preconditioned conjugated gradients and it has two main modules: sparse matrix-vector multiplication and triangular solver. The main problem is the solution of a linear system with symmetric positive definite sparse matrix. Several approaches use the compressed sparse row (CSR) format. However, the triangular solver with the CSR format has thread divergence. In this work, it is proposed a new representation for sparse matrix which eliminates the thread divergence. The new representation is colored jagged diagonals storage. The proposed GPU accelerated forward solver reduces the expected computational time.

Keywords— GPGPU, CUDA, Paralell Processing, Graph Coloring, Electrical Impedance Tomography.

Resumo— A Tomografia por Impedância Elétrica (TIE) é uma técnica de imageamento que reconstroi a distribuição da condutividade interna a um objeto a partir da injeção de corrente e medição de potencial elétrico na superficie do objeto. Este trabalho utiliza o gradiente conjugado precondicionado para solucionar o problema direto, e possui dois principais módulos: multiplicação matriz esparsa por vetor e solucionador triangular. O problema principal é a solução de sistema linear com matriz esparsa definida positiva simétrica. Várias abordagens utilizam a formato CSR que apresenta divergência de threads quando o solucionador triangular é executado. Neste trabalho, é proposta uma nova representação para matrizes esparsas que elimina a divergência de threads e realiza acesso à memoria exclusivamente sequencial e alinhados. A nova representação foi denominada como colored jagged diagonals storage. Os resultados demonstram o elevado desempenho da nova representação.

Palavras-chave— GPGPU, CUDA, Processamento Paralelo, Coloração de Grafos, Tomografia por Impedância Elétrica.

1 Introdução

A tomografia por impedância elétrica (TIE) é uma modalidade de imageamento não invasiva que estima a distribuição da condutividade dentro do corpo quando um padrão de corrente de baixa amplitude é aplicado sobre a superfície, e o potencial é medido utilizando eletrodos (Trigo et al., 2004; Martins, Camargo, Lima, Amato and Tsuzuki, 2012). O problema direto da TIE supõe conhecida a distribuição da condutividade interna ao corpo e o padrão de corrente injetado. Uma abordagem muito utilizada para resolver o problema direto é o método dos elementos finitos (MEF). O MEF utiliza uma malha para discretizar o domínio. Tavares et al. (2014) mostraram que malhas com níveis de refinamento adequados na região interior e exterior pode efetivamente reduzir o número de nós necessários para adquirir o nível de qualidade desejável, e ao reduzir o número total de nós o custo computacional também diminui.

Este trabalho possui como objetivo acelerar o solucionador de sistemas lineares, que neste caso é o gradientes conjugados (GC). O recente aparecimento de arquitetura com múltiplos núcleos, como GPUs, motivaram sua utilização em aplicações cientificas. Martins, Kian, Sato and Tsuzuki (2012) implementaram em GPU o GC precondicionado com formato CSR (Compressed Sparse Row) para armazenar matrizes esparsas. Sabe-se que implementações que utilizam o CSR possuem divergência de threads (algumas threads não executam processamento) quando o solucionador triangular (ST) é executado. Este trabalho propõe uma nova representação para matrizes esparsas que permite acelerar ainda mais o ST executado em GPU, e, consequentemente, o GC.

Este trabalho é estruturado da seguinte forma. A seção 2 explica o problema direto. A seção 3 explica a arquitetura CUDA e o armazenamento em matriz esparsa proposto. A seção 4 mostra os resultados, e as conclusões estão na seção 5.

2 O Problema Direto

O problema direto da TIE é, dada a distribuição de condutividades σ e a corrente J injetada pelos eletrodos do contorno, é necessário encontrar

a distribuição de potencial ϕ dentro de Ω e, em particular, o potencial resultante medidos nos eletrodos ϕ_m. As frequências utilizadas no TIE são baixas o suficiente, desta forma os efeitos capacitivos e indutivos podem ser desprezados. Assim, é necessário resolver a equação de Laplace

$$\nabla \cdot (\sigma \nabla \phi) = 0. \tag{1}$$

Nos contornos, correntes são injetadas utilizando dois eletrodos; então, a densidade de corrente J na superfície dos eletrodo é dada por

$$\sigma \frac{\partial \phi}{\partial \widehat{n}} = J \tag{2}$$

onde $\widehat{n}$ é o versor normal, e a densidade de corrente é nula nos demais eletrodos.

2.1 MEF Aplicado ao Problema Direto

A solução analítica para equações para um domínio irregular e meios isotrópicos com condição de contorno são desconhecidas, então, as equações diferenciais parciais são aproximadas pelo MEF. Uma formulação variacional das equações anteriores é realizada e a função ϕ é expressa como uma combinação linear de uma base de funções elementares, transformando o problema variacional em um problema de otimização quadrática. Uma solução aproximada de $\tilde{\phi}$ pode ser obtido pelo MEF. O domínio é discretizado com elementos triangulares lineares com condutividade constante, e, ambos os problemas, direto e inverso, são resolvidos numericamente. O princípio potencial virtual associado com a equação de Laplace fornece os elementos locais da matriz. Quando os elementos locais da matriz são iniciados em termos de coordenadas globais da malha, a matriz de condutividade global (Trigo et al., 2004; Martins and Tsuzuki, 2011), que incluem efeitos de impedância do eletrodo de contato, é obtido; então é válido

$$K \cdot \Phi = C \tag{3}$$

onde $K(\sigma) \in \mathbb{R}^{s \times s}$ é a matriz de condutividade calculada em um distribuição particular σ_p, Φ é a matriz que contém potencial nos nós, e C representa a corrente.

3 CUDA GPU

GPUs são unidades de processamento especializado com enorme capacidade computacional, e altamente paralelizável. CPUs atuais possuem dezenas de núcleos disponíveis, contra milhares de núcleos em GPUs. A NVIDIA introduziu uma API CUDA que implementa a SIMT (Single Instruction Multiple Thread) onde as instruções são executadas ao mesmo momento em todos os processadores. O SIMT permite acomodar dados não estruturados e é possível lidar com divergência de threads (threads sendo executada simultaneamente com códigos diferentes). Em ambos os casos ocorrem uma drástica queda de rendimento.

A unidade básica de trabalho em uma GPU é a thread, que é executada em um ambiente de memória compartilhada. O número máximo de threads que pode ser executado concorrentemente é determinado pelo hardware, e o escalonador de threads decide quando um grupo de threads pode ser executado. Um kernel, é uma sub-rotina que pode ser chamada pelo host para ser executada no dispositivo CUDA, deve utilizar muitas threads para realizar a tarefa designada. As threads são divididas em blocos, e cada bloco possui um conjunto interno de memória local utilizado para compartilhar dados entre as threads. Dados podem ser compartilhados entre blocos pela memória global, que, por ser externa, é limitada pela banda e está sujeita a regras de coalescência. A latência em acessar a memória global pode ser alta, até 600 vezes mais devagar que acessar uma variável de registro.

O dispositivo aglutina o carregamento e armazenamento da memória global que é requerida pelas threads de um warp em poucas transações, para minimizar a banda da DRAM. Blocos que carregam endereços contínuos de memória sequencialmente são muito mais rápidos que aqueles que realizam acessos dispersos. Uma warp possui 32 threads para todos os dispositivos CUDA com capacidade de computação 2.0 ou maior.

3.1 Armazenamento de Matriz

As matrizes de rigidez produzidas pelo MEF na TIE são esparsas. Existem vários métodos para armazenar efetivamente matrizes esparsas: CSR, BCSR (Block CSR), JDS (Jagged Diagonals Storage) e ELLPACK. O CSR utiliza três vetores para armazenar todos os elementos não nulos de uma matriz, um para dado, um para o índice de coluna correspondente ao valor, e o último para cada local de início de linha nos dois primeiros arrays. Considere a seguinte matriz

$$A = \begin{pmatrix} 3 & 0 & 0 & 0 & 4 \\ 0 & 8 & 7 & 8 & 0 \\ 0 & 7 & 1 & 0 & 8 \\ 0 & 8 & 0 & 2 & 0 \\ 4 & 0 & 8 & 0 & 4 \end{pmatrix} \tag{4}$$

sua representação CSR é

$$\begin{aligned} data &= [\,3\ 4\ 8\ 7\ 8\ 7\ 1\ 8\ 8\ 2\ 4\ 8\ 4\,] \\ idx &= [\,0\ 4\ 1\ 2\ 3\ 1\ 2\ 4\ 1\ 3\ 0\ 2\ 4\,] \\ ptr &= [\,0\ 2\ 5\ 8\ 10\ 13\,]. \end{aligned}$$

Este formato armazena o mínimo de dados necessário, mas não é otimizado para acessos aglutinados e, isto pode criar desvios condicionais (Vázquez et al., 2011). O CSR já foi

implementado em GPU por alguns pesquisadores (Martins, Kian and Yabuki, 2012).

ELLPACK-R, um variante do ELLPACK original, é o padrão utilizado para implementar a multiplicação entre matriz esparsa e vetor (MMExV) em GPUs (Vázquez et al., 2011). A idéia é comprimir as linhas ao compactar os elementos não nulos à esquerda, resultando na matriz $N \times N_{\max(nz)}$, onde N é o número de linhas e $N_{\max(nz)}$ é o número máximo de elementos não nulos em uma linha. Em contraste com o formato CSR, o ELLPACK-R armazena entradas nulas. A matriz mostrada em (4) é representada no formato ELLPACK-R como

$$data = \begin{bmatrix} 3 & 4 & 0 \\ 8 & 7 & 8 \\ 7 & 1 & 8 \\ 8 & 2 & 0 \\ 4 & 8 & 4 \end{bmatrix} \quad idx = \begin{bmatrix} 0 & 4 & 0 \\ 1 & 2 & 3 \\ 1 & 2 & 4 \\ 1 & 3 & 0 \\ 0 & 2 & 4 \end{bmatrix} \quad rl = \begin{bmatrix} 2 \\ 3 \\ 3 \\ 2 \\ 3 \end{bmatrix},$$

onde *data* contém a matriz comprimida, *idx* contém a coluna de cada elemento em *data* e *rl* contém o comprimento. Segundo Vázquez et al. (2011), a utilização de *rl* reduz o número de desvios condicionais dentro do código em GPU.

O formato pJDS (padded Jagged Diagonals Storage) foi proposto como um aprimoramento ao ELLPACK-R, com o objetivo de reduzir o armazenamento de entradas desnecessárias de zero e aprimorar a utilização dos recursos computacionais (Kreutzer et al., 2012). As linhas no formato ELLPACK-R são classificadas de acordo com o seu comprimento. Então, as linhas são divididas em blocos de tamanho do warp e todas as linhas em um bloco são adicionados à linha mais longa no bloco. O acesso à memória é mantida enquanto que o armazenamento necessário é reduzido. A Fig. 1 ilustra a diferença entre o formato ELLPACK-R e o formato pJDS. É considerado uma matriz de 12 linhas, e um tamanho de warp de 2 threads. Ambas as imagens mostram o comprimento de cada linha, em elementos não nulos. A cor vermelha mais clara com setas representa elementos zero que foram adicionados.

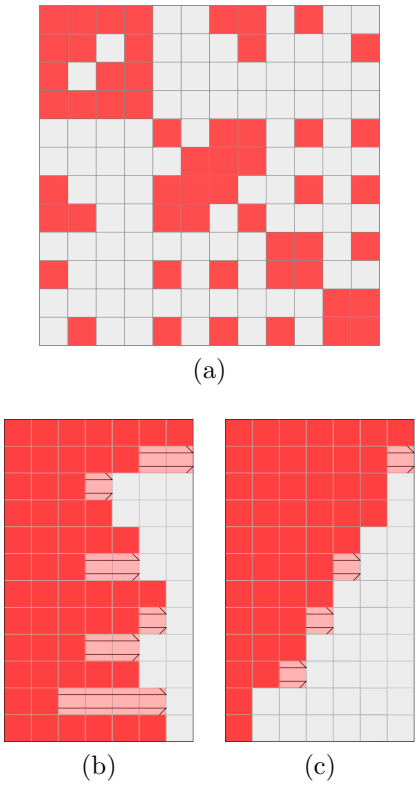

Figura 1: (a) Matriz original. (b) Matriz representada segundo o formato ELLPACK-R. (c) Matriz representada segundo o formato pJDS.

3.2 Colored Padded Jagged Diagonals Storage

Martins, Kian and Yabuki (2012) mostraram que modelos do MEF da TIE podem ser coloridos para melhorar a paralelização de ST. Foi mostrado que um grafo planar pode ser colorido utilizando no máximo 4 cores. Assim, após a ordenação da matriz pela cor, o ST é paralelizável. Porém, como já foi mencionado, o formato pJDS necessita que a matriz seja ordenada de acordo com o comprimento da linha.

Para obedecer a ambas as condições (ordenação pela cor e pelo comprimento de linha), é proposto que a matriz seja ordenada duas vezes. Primeiro, ela é ordenada de acordo com a cor, dividindo suas linhas em blocos coloridos. Dentro de cada bloco colorido, é feito uma ordenação de acordo a cada comprimento de linha. Em seguida, linhas são novamente divididas em blocos em tamanhos do warp, para preenchimento como é no pJDS. É importante manter linhas de diferentes blocos coloridos separados, que implicará linhas vazias a serem adicionadas, a diagonal principal é preenchidas com 1, que é crucial para manter a matriz de rigidez positiva definida. A matriz resultante deve cumprir as seguintes regras:

1. Linhas são divididas em cores e ordenadas de acordo com as cores
2. Linhas em todas as cores são ordenadas de acordo com o comprimento, e subdivididas em blocos de tamanho do warp e preenchidas
3. Pode ocorrer a adição de linhas na matriz para que a matriz possua um número de linhas múltiplo de tamanho do warp.

A Fig. 2 ilustra este processo. A estrutura matricial inicial é a mesma que a mostrada na Fig. 1, consistindo 12 linhas. Na Fig. 2(b), cores foram identificadas, e linhas são ordenadas por cor na Fig. 2(c). Na Fig. 2(d), as cores claras com setas mostram elementos zeros que são adicionados para se obter tamanho de warp - incluindo novas linhas. Considerou-se que o tamanho de warp é composto por 2 threads.

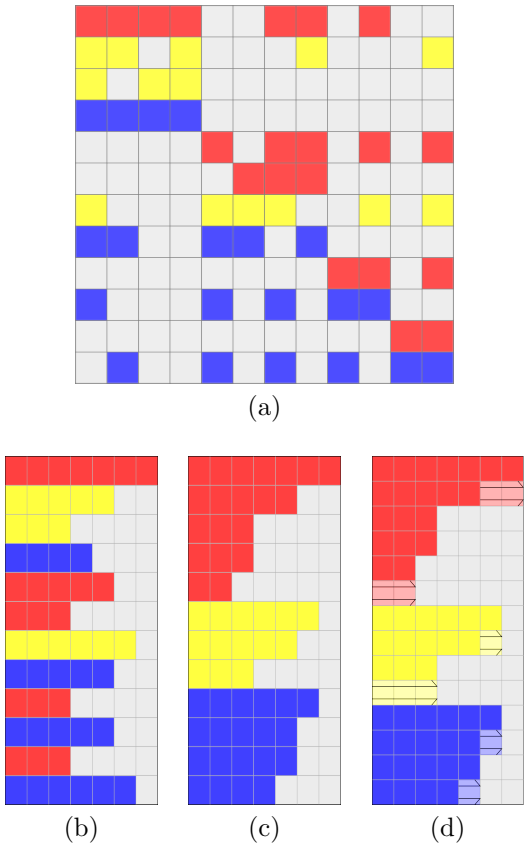

Figura 2: (a) Matriz exibida em Fig. 1(a) colorida. (b) Matriz no formato ELLPACK-R. (c) Matriz ordenada segundo a cor. (d) Matriz segundo o formato proposto.

Considere o exemplo numérico de uma matriz A de dimensão 5×5

$$A = \begin{pmatrix} 10 & 3 & 0 & 2 & 0 \\ 3 & 6 & 4 & 0 & 3 \\ 0 & 4 & 11 & 1 & 0 \\ 2 & 0 & 1 & 8 & 3 \\ 0 & 3 & 0 & 3 & 5 \end{pmatrix}.$$

Sua decomposição triangular inferior incompleta de Choleski é

$$L = \begin{pmatrix} 3.16 & 0 & 0 & 0 & 0 \\ 0.95 & 2.26 & 0 & 0 & 0 \\ 0 & 1.77 & 2.80 & 0 & 0 \\ 0.63 & 0 & 0.52 & 2.69 & 0 \\ 0 & 1.33 & 0 & 1.41 & 0.74 \end{pmatrix}.$$

Um ST utilizando L não pode ser facilmente paralelizável, e são necessários 5 passos de cálculo e 4 passos de propagação.

Mas, se A for colorida e então ordenada de acordo com suas cores, primeiro os elementos cuja cor estiver com maior número de linhas, a matriz resultante ordenada por cor pode ser paralelizada, como

$$csA = \begin{pmatrix} 10 & 0 & 0 & 2 & 3 \\ 0 & 11 & 0 & 1 & 4 \\ 0 & 0 & 5 & 3 & 3 \\ 2 & 1 & 3 & 8 & 0 \\ 3 & 4 & 3 & 0 & 6 \end{pmatrix}.$$

Agora, são necessários somente 2 passos computacionais, e 1 passo de propagação. A matriz resultante, é preenchida (considerado um tamanho de warp de 2 threads) e resulta

$$pcsA = \begin{pmatrix} 10 & 0 & 0 & 0 & 2 & 3 \\ 0 & 11 & 0 & 0 & 1 & 4 \\ 0 & 0 & 5 & 0 & 3 & 3 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 2 & 1 & 3 & 0 & 8 & 0 \\ 3 & 4 & 3 & 0 & 0 & 6 \end{pmatrix}.$$

Este formato utiliza vantagem de todas as qualidades do ELLPACK-R/pJDS, como acesso de memória aglutinada, sem desvios condicionais, computação homogênea dentro de kernels e desbalanceamento reduzido de kernel, e necessidade de armazenamento mínimo. Algoritmos simples para MMExV (ver Algoritmo 5) e o ST com substituição para frente (ver Algoritmo 2) são apresentados. Como algumas linhas foram preenchidos com zeros, podem ocorrer operações vazias. Mas, se todas as threads são sincronizadas, uma operação vazia não adiciona mais tempo de computação que desvios condicionais e acesso de memória aleatória.

Algorithm 1 MMExV.

```
for i = 0 to rows - 1 do
    for idx = 0 to rl[i] do
        y[i] = y[i] + v[indices[idx]] * data[idx]
    end for
end for
```

Algorithm 2 Solucionador triangular.

```
for k = 0 to colors - 1 do
    for i = colors[k] to colors[k + 1] - 1 do
        sum = 0
        for idx = 0 to rl[i] - 1 do
            sum = sum + L[indices[idx]] * x[idx]
        end for
        x[i] = (b[i] - sum) / L[indices[i]]
    end for
end for
```

3.3 Gradiente Conjugado em CUDA

O GC precondicionado é apresentado no Algoritmo 3. Cada iteração do GC precondicionado consiste em: (1) 4× produtos internos, (2) 3× multiplicação escalar-vetor, (3) 2× somatória de

Algorithm 3 GC precondicionado.

$r_0 = b - A * x_0$
$z_0 = M^{-1} * r_0$
$p = z$
for k = 0 to N - 1 **do**
  $\alpha_k = \frac{r_k^t \cdot z_k}{p_k^t * A * p_k}$
  $x_{k+1} = x_k + \alpha_k p_k$
  $r_{k+1} = r_k - \alpha_k A * p_k$
  $z_{k+1} = M^{-1} * r_{k+1}$
  $\beta_k = \frac{z_{k+1} \cdot r_{k+1}}{z_k \cdot r_k}$
  $p_{k+1} = z_{k+1} + \beta_k p_k$
end for

vetor, (4) 2× ST, (5) 1× subtração de vetor, (6) 1× MMExV.

Todas estas operações são paralelizáveis de modo que o tempo de processamento não varie com a tamanho da malha. Todos os kernels podem ter uma thread manipulando uma linha. O ST é o único kernel com um laço adicional, um para cada cor. Assim, independentemente do tamanho da malha, o número de passos, que considera todas as linhas sendo processadas ao mesmo tempo, não varia, em oposição à implementação serial.

4 Resultados

Para a avaliação do formato de armazenamento de matriz proposto, os kernels desenvolvidos foram testados em uma placa NVIDIA GeForce GTX670 com 1344 CUDA núcleos e 2GB DRAM, com compute capability 3.0 e CUDA 6.5. A CPU é Intel Xeon ES645 2.4GHz com 32GB RAM, com Windows 7 Ultimate. Todos os kernels utilizam precisão simples para os dados, e são lançado em blocos de 512 threads. O algoritmo apresentado é comparado com uma aplicação serial padrão em Eigen2. Todos os tempos são medidos considerando cinco malhas semelhantes com tamanhos similares e os tamanhos indicados representam uma média de tais malhas. Um algoritmo de coloração de grafos divide todas as malhas em 5 cores. Como precondicionador, foi utilizado a decomposição incompleta de Choleski.

A Tabela 1 compara o Eigen2 e o algoritmo proposto para três principais operações: produto interno de dois vetores, MMExV e ST. As dimensões dos vetores e matrizes são exbidos, a operação é repetida 1.000 vezes. Pode ser observado que a implementação serial em Eigen é significativamente afetada pelas dimensões da matriz e do vetor, enquanto os algoritmos paralelos não são afetados. O produto interno em média leva 4 ms para todos as dimensões consideradas, mas com o aumento da dimensão do vetor a o formato proposto se torna cada vez mais vantajoso. O mesmo pode ser observado quando se analisa a MMExV, exceto que o formato proposto é no mínimo duas vezes mais rápido mesmo para pequenas dimensões. Finalmente, o ST proposto é mais rápido para todas as dimensões, exceto para matrizes de pequenas dimensões.

Tabela 1: Tempo médio para 3 operações principais - para 1.000 iterações. TM = Tamanho da malha, PI = Produto Interno

Operação	TM	Eigen2 (ms)	CUDA (ms)
PI	456	0,4	4,0
	1.046	1,2	4,4
	4.500	4,2	4,0
	9.050	8,8	4,2
MMExV	456	9,8	4,6
	1.046	23,4	4,8
	4.500	100,3	4,8
	9.050	217,0	5,0
ST	456	7,0	36,7
	1.046	47,5	30,5
	4.500	71,5	26,0
	9.050	164,0	26,0

Tabela 2: Comparação dos GC Precondicionado - tempo médio para 100 iterações.

TM (nós)	Eigen2 (ms)	CUDA (ms)
456	3	12
1.046	5	12
2.130	11	12
4.500	22	12
6.900	34	12
9.050	47	12

A Tabela 2 mostra como um aumento no número de nós da malha não afeta o tempo de processamento na implementação em GPU. Pode ser observado que, para malhas de até aproximadamente 2.500 nós, a implementação em Eigen é executada mais rapidamente que a proposta em GPU. Para malhas com mais nós, existe uma clara vantagem ao se utilizar o formato proposto.

O método proposto inclui uma etapa de pré-processamento que constrói a matriz de rigidez

Tabela 3: Pré-processamento ordenação por cor - tempo médio para 100 iterações.

TM (nós)	CUDA (ms)
456	3
1.046	25
2.130	138
4.500	708
6.900	1.748
9.050	3.152

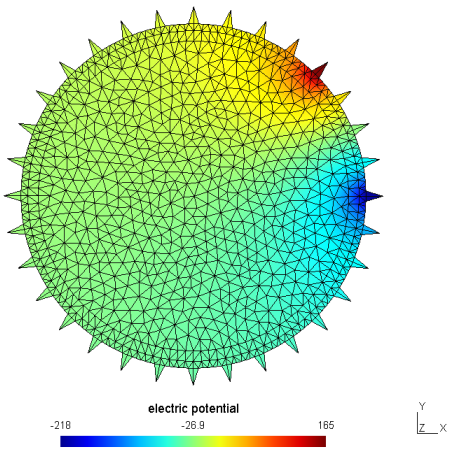

Figura 3: Malha e distribuição dos potenciais elétricos para uma malha de 1046 nós.

segundo o formato também proposto. A Tabela 3 exibe o tempo necessário para executar esta etapa de pré-processamento. Esta etapa de pré-processamento é afetada pelo número de nós da malha, mas ela é executada apenas uma vez para cada malha.

A Fig. 3 mostra uma malha com 1.046 nós e as cores indicam a distribuição de potenciais elétricos resultante do problema direto da TIE. Existem 32 eletrodos no contorno, e o eletrodo na posição mais alta foi utilizada como terra. Os eletrodos com o potencial mais elevados indicam o nó onde ocorre a injeção de corrente, e o eletrodo com o menor potencial, o nó onde ocorre a extração de corrente.

5 Conclusões e Trabalho Futuro

Foi proposto um método para pré-processar e armazenar matrizes esparsas de modo a aumentar o nível de paralelização utilizando o framework CUDA. O método proposto não apresenta divergência de threads e os acessos à memória são sequenciais e alinhados. Dos testes realizados, é possível concluir que a proposta é muito vantajosa para malhas razoável número de nós e, especialmente, para elevado número de nós. Para malhas com baixo número de nós é mais adequado utilizar a implementação serial.

Como trabalho futuro, o formato para representação de matrizes esparsas e o GC implementado serão utilizados para implementar um algoritmo para resolver o problema inverso.

Agradecimentos

R. S. Tavares foi suportado pela FAPESP (processo 2010/18658-4). M. S. G. Tsuzuki e T. C. Martins possuem suporte parcial do CNPq (respectivamente, processos 310.663/2013–0 e 306.415/2012–7). R. Y. Takimoto e E. K. Ueda são suportados pelo CNPq (respectivamente, processos 150508/2015–8 e 140518/2015–0). A. K. Sato é suportado pela CAPES/PNPD.

Referências

Kreutzer, M., Hager, G., Wellein, G., Fehske, H., Basermann, A. and Bishop, A. R. (2012). Sparse matrix-vector multiplication on GPGPU clusters: A new storage format and a scalable implementation., *IPDPS Workshops*, pp. 1696–1702.

Martins, T. C., Camargo, E. D. L. B., Lima, R. G., Amato, M. B. P. and Tsuzuki, M. S. G. (2012). Image reconstruction using interval simulated annealing in electrical impedance tomography, *IEEE T Biomed Eng* **59**: 1861–1870.

Martins, T. C., Kian, J. M., Sato, A. K. and Tsuzuki, M. S. G. (2012). Matrix-vector multiplication and triangular linear solver using GPGPU for symmetric positive definite matrices derived from elliptic equations., *Proc 6th Int Conf Soft Computing and Intelligent Systems*, Kobe, Japan, pp. 1286–1291.

Martins, T. C., Kian, J. M. and Yabuki, D. K. (2012). GPU accelerated reconstruction of electrical impedance tomography images through simulated annealing, *Proc 10th World Congress on Computational Mechanics*, São Paulo, Brazil.

Martins, T. C. and Tsuzuki, M. S. G. (2011). Simulated annealing with partial evaluation of objective function applied to electrical impedance tomography, *Proc 18th IFAC WC*, Milan, Italy, pp. 4989–4994.

Tavares, R. S., Nakadaira-Filho, F. A., Tsuzuki, M. S. G., Martins, T. C. and Lima, R. G. (2014). Discretization error and the EIT forward problem., *Proc 19th IFAC World Congress*, Cape Town, South Africa, pp. 7535–7540.

Trigo, F. C., Lima, R. G. and Amato, M. B. P. (2004). Electrical impedance tomography using the extended Kalman filter, *IEEE T Biomed Eng* **51**: 72–81.

Vázquez, F., Fernández, J. and Garzón, E. M. (2011). A new approach for sparse matrix vector product on NVIDIA GPUs, *Concurrency and Computation: Practice and Experience* **23**(8): 815–826.