Paralelização da aplicação RAFEM usando GPU

Marcelo Miletto[1], **Claudio Schepke**[1]

[1]Laboratório de Estudos Avançados – Universidade Federal do Pampa (UNIPAMPA)
Av. Tiarajú, 810, 97546-550, Alegrete – RS – Brasil

marcelocm97@gmail.com, claudioschepke@unipampa.edu.br

Resumo. *RAFEM (Radiofrequency Ablation Finite Element Method) é uma aplicação de simulação computacional para um procedimento médico que envolve um grande tempo de execução. Para reduzir este tempo, o presente trabalho aplica uma metodologia de desenvolvimento iterativa voltada para aplicações paralelas e usa bibliotecas aceleradas por GPU. Os resultados mostram que foi possível obter uma redução do tempo de execução de até 27 vezes.*

1. Introdução

Neste trabalho estudamos a aplicação RAFEM que simula um procedimento minimamente invasivo para o tratamento de alguns casos de câncer de fígado. O problema é que existe um elevado custo computacional nas simulações que esta aplicação realiza, chegando a até 20 horas de tempo de execução. Como o RAFEM utiliza o método dos elementos finitos para a simulação, existem duas etapas principais que contribuem com este custo elevado: a montagem de um sistema de equações e a sua solução. Logo, para obter uma melhora de desempenho podemos focar na paralelização destas duas etapas usando o poder computacional oferecido pelas GPUs. Com isto, espera-se obter um melhor desempenho para a aplicação, contribuindo para que seu uso seja mais viável.

2. Metodologia

RAFEM não foi projetado pensando-se na execução paralela do programa. Assim, as estruturas de dados utilizadas, a movimentação de dados e o fluxo de execução do programa em si não se adéquam ao processamento paralelo. Para que a aplicação possa ser executada paralelamente em GPU de forma eficiente, deve-se levar em consideração aspectos como transferências de memória entre CPU e GPU e a modelagem dos problemas de forma com que explorem o paralelismo de dados. Para isto, deve-se realizar uma refatoração do código e a implementação de novas funções para GPU. Tal tarefa não é trivial, já que a aplicação possui em torno de 4000 linhas de código.

Para guiar o processo de paralelização da aplicação foi adotado um processo de desenvolvimento de *software* iterativo voltado para aplicações paralelas, o APOD (*Access, Paralelize, Optimize, Deploy*) [NVIDIA 2018]. Este é dividido em 4 etapas: a Avaliação consiste na análise da execução da aplicação utilizando casos de teste reais para identificar as partes mais custosas da aplicação. A Paralelização e Otimização envolvem a criação de código paralelo eficiente para as partes custosas identificadas, levando em consideração os resultados produzidos pela aplicação. Por fim, a etapa de Implantação permite que ao fim de um ciclo de desenvolvimento seja gerada uma versão paralela do programa e o ciclo se reinicia. Com isto, é possível adicionar melhorias incrementalmente na aplicação RAFEM, focando em diferentes etapas ao longo dos ciclos de desenvolvimento.

Para auxiliar no desenvolvimento das versões paralelas foram utilizadas algumas ferramentas e bibliotecas, como por exemplo os *profilers* **gprof** e **nvprof** para analisar a execução da aplicação tanto em GPU quanto em CPU. Foi desenvolvido um *script* em **python** para a verificação e validação dos resultados numéricos e também foi utilizada a biblioteca MAGMA (*Matrix Algebra on GPU and Multicore Architectures*) [Anzt et al. 2014] que oferece uma série de métodos numéricos acelerados por GPU para a solução de sistemas de equações esparsos.

Com o intuito de avaliar a execução da aplicação foram utilizados dois casos de teste representando modelos de fígados reais criados a partir do *software* ANSYS. As malhas de elementos finitos dos casos de teste são descritas na Tabela 1 juntamente com os sistemas de equações que são gerados. A matriz gerada é uma matriz quadrada de $M \times N$ termos de precisão dupla com NNZ termos não nulos (diferentes de zero). O ambiente computacional utilizado para a realização de experimentos foi uma *workstation* com um processador **Xeon E5-2650 ($\times$2)** com frequência de 2.0 GHz e uma GPU **Quadro M5000** com 2048 CUDA *cores* trabalhando a uma frequência de 1.04 GHz.

Tabela 1. Malhas de elementos finitos utilizadas

Malha	Elementos	Nós	$M \times N$	NNZ
Caso 1	18.363	3.548	50.353.216	189.462 (0,0037%)
Caso 2	44.811	8.364	279.825.984	450.451 (0,0016%)

No primeiro ciclo de desenvolvimento do processo APOD foi necessária uma refatoração no código, pois a versão original utiliza o método Frontal [Irons 1970] para a etapa de obtenção da solução baseando-se em sub matrizes do problema. Após obtida a representação completa do problema, focou-se a paralelização da etapa de montagem das matrizes dos elementos, uma sub etapa do processo de montagem do sistema de equações. Para isto foi desenvolvido um *kernel* em CUDA que realizava a montagem da matriz de cada elemento de forma paralela, salvando os dados de cada elemento em uma **struct**, para posteriormente serem montados na matriz global. Com o problema montado, foi utilizada o método numérico iterativo GMRES(m) [Saad and Schultz 1986] implementado pela biblioteca MAGMA para obter sua solução. Trata-se de um método iterativo que pode ser usado em casos onde a matriz é esparsa e não simétrica.

Com a **Versão 1** gerada, iniciou-se um novo ciclo do processo dando origem à **Versão 2**, agora focando em realizar toda montagem do problema em GPU para evitar transferências de memória desnecessárias. Assim, foram desenvolvidos novos *kernels* CUDA para esta etapa. É importante ressaltar que a cada nova versão gerada os resultados numéricos obtidos foram comparados através do *script* desenvolvido para verificar a consistência com os resultados gerados pela aplicação original.

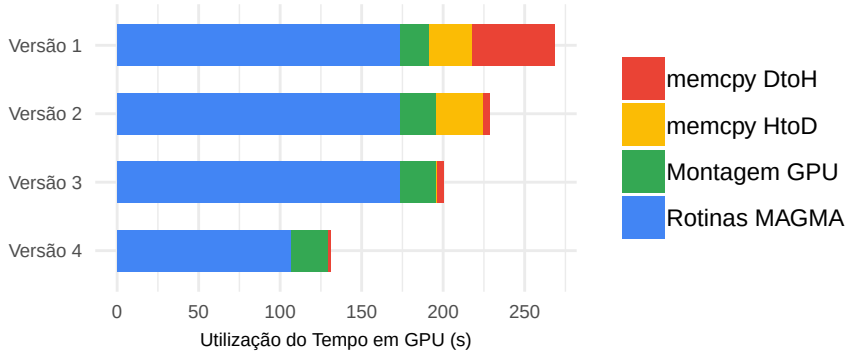
Para a **Versão 3**, focou-se na paralelização de operações de atualização que estavam sendo feitas dentro do laço principal do programa e que geravam um custo de transferência da CPU para a GPU. Foram usados métodos da biblioteca **thrust** para realizá-las em GPU, mitigando assim custos envolvendo a transferência de dados. Para a **Versão 4** foram explorados alguns parâmetros do método GMRES(m) como o valor de tolerância e o parâmetro `m`, que indica quando uma operação de reinício é realizada pelo método. Estes parâmetros impactam no número de iterações necessárias para a obtenção

da solução, afetando diretamente o tempo de execução da aplicação. Com a troca do valor de tolerância de 10^{-10} para 10^{-6} foi possível reduzir o tempo da aplicação mantendo os resultados numéricos consistentes.

3. Resultados Obtidos

Ao longo dos ciclos de desenvolvimento, a etapa de avaliação foi repetida para descobrir novos pontos de melhoria. A evolução proporcionada pelas novas versões pode ser observada na Figura 1, que mostra o *profiling* das diferentes versões evidenciando as etapas executadas em GPU. Estes valores foram obtidos usando a ferramenta **nvprof** para o Caso 1. Pode-se perceber que grande parte do tempo era usada para transferências de dados. Isto foi resolvido ao realizar mais etapas computacionais em GPU.

Figura 1. Profiling do Tempo em GPU das Versões Desenvolvidas para o Caso 1



Para mensurar o desempenho obtido com as versões paralelas desenvolvidas foram realizados experimentos para calcular o tempo médio de execução para os dois casos. A versão original do programa levava 1 hora e 23 minutos para o Caso 1 e 20 horas e 27 minutos para o Caso 2, os tempos de execução aqui apresentados representam a média de 30 execuções. Na Tabela 2 é possível observar os valores de tempo de execução das diferentes versões geradas para os dois casos de teste, o desvio padrão obtido a partir das amostras de tempo de execução e os valores de *speedup* em comparação a versão original da aplicação. O valor de *speedup* foi calculado de forma a representar a *performance* entre duas soluções que resolvem o mesmo problema, assim basta calcular o tempo da versão original dividido pelo tempo da versão que se deseja calcular o *speedup*.

Tabela 2. Valores obtidos através dos experimentos

		Original	Versão 1	Versão 2	Versão 3	Versão 4
Caso 1	Tempo (min)	83.26	8.53	6.69	4.52	3.02
	Desvio Padrão	0.27	0.017	0.037	0.088	0.048
	Speedup	1	11.88	12.43	18.23	27.53
Caso 2	Tempo (min)	1227.75	151.33	116.6	77.78	45.84
	Desvio Padrão	0.832	1.246	0.289	0.457	0.07
	Speedup	1	8.11	10.52	15.78	26.78

4. Discussão dos Resultados

Ao analisarmos os resultados apresentados na Figura 1 podemos perceber que em uma aplicação real o tempo com transferências de dados pode representar uma boa porção do tempo total de atividade em GPU. O *profiling* da aplicação ajudou a focar na otimização de etapas que tinham uma contribuição expressiva para o tempo total de execução. O tempo usado com cópias de memória pôde ser substituído e reduzido por computações realizadas em GPU.

Com relação aos valores de *speedup* calculados pode-se perceber uma diferença de até 3 vezes entre o Caso 1 e o Caso 2 para as primeiras três versões. Isso aconteceu por consequência dos parâmetros utilizados no método GMRES(m), em especial o parâmetro m, que influi na convergência do método. O que foi percebido na **Versão 4** é que para os dois casos o valor de m que apresentava a melhor taxa de convergência eram diferentes. Assim, na **Versão 4** estes valores foram ajustados juntamente com o aumento do valor de tolerância, chegando a um *speedup* de até 27 vezes. É perceptível que podemos garantir uma boa redução no tempo total de execução da aplicação, e para que essa solução não dependa da configuração manual de parâmetros, existem variações do método GMRES(m) onde o valor de m é dinâmico, sendo adaptado ao longo da execução da aplicação.

5. Conclusões

A partir dos resultados obtidos neste trabalho, é possível realizar simulações em um tempo menor para a aplicação RAFEM. Isso contribui para que casos sejam estudados de forma mais eficiente, reduzindo a alocação de recursos computacionais para uma simulação e ajuda no trabalho de processar modelos de fígado mais detalhados. Como trabalhos futuros podem ser feitos estudos em outros casos de teste para verificar o impacto dos parâmetros utilizados no método numérico bem como a comparação dos resultados obtidos usando outros ambientes computacionais, bibliotecas e métodos numéricos.

Referências

[Anzt et al. 2014] Anzt, H., Sawyer, W., Tomov, S., Luszczek, P., Yamazaki, I., and Dongarra, J. (2014). Optimizing krylov subspace solvers on graphics processing units. In *Fourth International Workshop on Accelerators and Hybrid Exascale Systems (AsHES), IPDPS 2014*, Phoenix, AZ. IEEE, IEEE.

[Irons 1970] Irons, B. M. (1970). A frontal solution program for finite element analysis. *International Journal for Numerical Methods in Engineering*, 2(1):5–32.

[NVIDIA 2018] NVIDIA (2018). CUDA C Best Practices Guide. `https://docs.nvidia.com/cuda/cuda-c-best-practices-guide/index.html`.

[Saad and Schultz 1986] Saad, Y. and Schultz, M. H. (1986). GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM Journal on scientific and statistical computing*, 7(3):856–869.