A Tool for Scientific Visualization Based on Particle Tracing Algorithm on Graphics Processing Units

Eduardo Camargo
Sergio Kostin
Raquel Coelho Gomes Pinto

Instituto Militar de Engenharia
Praça Gen Tibúrcio, 80 - Rio de Janeiro, RJ - Brasil
eduardo.ecamargo@gmail.com
kostin@ime.eb.br
raquel@ime.eb.br

Abstract

The need to process large volumes of data associated with the increasing computing power contained in the current graphics cards have encouraged the academic community to produce techniques applied to parallel graphics processing designed for scientific applications. In this context, the goal of this paper is to present an implementation of a scientific visualization tool based on the Particle Tracing Algorithm on graphics processing units using, in particular, the Compute Unified Device Architecture (CUDA). It is also evaluated the performance gains obtained related the combination of some of the settings offered by CUDA.

1 Introdução

As técnicas de modelagem e de simulação computacional requerem grande poder de processamento. A grande eficiência das atuais placas de vídeo no processamento de dados despertou a atenção da comunidade científica como meio de suprir suas necessidades por computação de alto desempenho. O custo e o consumo de energia das atuais placas são baixos se comparado a sistemas dedicados a problemas de alto desempenho [3]. Porém, nem todos os problemas computacionais podem utilizar tal tecnologia mas somente os problemas passíveis de serem paralelizáveis [3].

Uma técnica de visualização científica que não é facilmente encontrada em ambientes de visualização de código aberto é o cálculo de trajetória de partículas (CTP). O CTP manipula dados oriundos de uma simulação numérica gerando, portanto, um processamento geralmente demorado. Logo, existe a exigência de se alcançar melhores desempenhos na execução deste tipo de aplicação.

O CTP é um problema que pode fazer uso das tecnologias oferecidas pela área de Computação de Alto Desempenho (*High Performance Computing* - HPC) na obtenção de melhores desempenhos. A HPC é uma área do mercado mundial suprida por empresas que utilizam tecnologias como o processador *Cell*, a computação de propósito geral em unidades de processamento gráfico e/ou agrupamento de computadores, entre outras arquiteturas, com o intuito de propor a melhor solução, para os mais diversos domínios de aplicação, como: análise financeira, mineração de dados, processamento de imagens, computação científica, modelagem computacional, visualização científica, etc [1].

Esta técnica pode ainda ser utilizada para a visualização do transporte de substâncias pelo fluxo sanguíneo, na visualização deste fluxo no interior de um aneurisma ou na identificação de suas condições de formação [8].

A contribuição do presente trabalho é apresentar a implementação de uma ferramenta de visualização científica baseada no algoritmo de cálculo de trajetória de partículas em unidades de processamento gráfico utilizando-se a arquitetura CUDA. São analisadas ainda diferentes configurações da presente implementação paralela.

O restante deste artigo está dividido da seguinte forma: a seção 2 apresenta os trabalhos relacionados,

posteriormente, na seção 3, é caracterizada a implementação paralela da técnica do cálculo de trajetória de partículas, a seção 4 apresenta os resultados obtidos. Por fim, na seção 5, são apresentadas a conclusão e os trabalhos futuros.

2 Trabalhos Relacionados

Porto *et al* [11] apresentam o ambiente CoDIMS (*Configured Data Integration Middleware System*) que foi desenvolvido para gerar sistemas de *middleware* de grade que suportem a execução de aplicações científicas. Este ambiente possui um componente de controle que instancia e configura objetos de um *framework* próprio, visando a criação de um *middleware* customizado.

Oliveira [5] utilizou o QEF (*Query Evaluation Framework*) para implementar o cálculo de trajetória de partículas em ambiente de grade. O QEF foi projetado para suportar a execução de aplicações complexas em ambiente de grade, neste *framework* as técnicas de processamento da visualização são modeladas como operadores algébricos e integradas a um conjunto de operadores de controle que gerenciam a distribuição de dados na grade. Verificou-se que a implementação do cálculo de trajetória de partículas utilizando QEF teve desempenho superior ao obtido em [11]. Este resultado deve-se ao fato de não existir transferência de dados entre os nós remotos e o nó mestre a cada passo de computação.

Hetch *et al* [6] apresentam e discutem a implementação de uma técnica, semelhante ao CTP, que também é empregada para visualizar simulações de dinâmica de fluidos com base em seu vetor velocidade. A técnica modela partículas que não podem ser vistas individualmente por serem muito pequenas. Tais partículas só podem ser vistas quando em grandes quantidades e com auxílio de técnicas de iluminação. A implementação realizada foi desenvolvida para ser processada em CPU e sua maior limitação é não permitir identificar visualmente a direção do fluido em um quadro de vídeo isolado, sendo necessários mais quadros para determinar sua direção.

Mittmann *et al* [9] descrevem e implementam em GPU a técnica de visualização baseada em linhas de corrente e Kondratieva *et al* em [7] descrevem e implementam em GPU o cálculo de trajetória de partículas. Ambos os trabalhos têm por objetivo localizar e rastrear moléculas de água entre tecidos vivos. Os dados de entrada são um exame de ressonância magnética. O algoritmo identifica e calcula a trajetória de partículas ou linhas de corrente entre as moléculas de água das fibras do tecido, desse modo, reconhecendo estruturas anatômicas. Mittmann *et al* [9] também implementam uma versão da referida técnica para aglomerados de computadores (*clusters*) e comparam com a versão GPU, concluindo que a versão GPU é superior em velocidade de processamento, entrentato não realiza comparações entre diferentes configurações oferecidas pela CUDA.

Estes trabalhos abordam o cálculo de trajetória de partículas salientando sua importância quanto técnica de visualização de resultados para simulações numéricas e para processamento de imagens. Os resultados dos trabalhos também apontam a utilização de tecnologias como *cluster* e GPU para obter ganho de desempenho em suas implementações.

3 Paralelização do Cálculo de Trajetória de Partículas

3.1 Implementação Serial

O cálculo de trajetórias de partículas é um conceitos comum em mecânica de fluidos e serve para caracterizar um fluxo de fluidos, permitindo o estudo do comportamento do fluido instável enquanto as linhas de corrente, como o trabalho de Mittmann *et al* [9], normalmente são utilizadas para estudar fluidos estáveis.

O método *Lagrange* para cálculo de trajetória de partículas em fluidos em um escoamento foi concebido em 1760 pelo matemático Joseph Louis Lagrange podendo ser descrito como [12]: dada uma partícula de massa desprezível e identificada por sua posição inicial $\mathbf{p}$, um vetor velocidade que descreve o fluxo do fluído ($\mathbf{v_p}$) e um intervalo de tempo (t_0 a $T_\mathbf{p}$), calcular a próxima posição da partícula ($\mathbf{x_p}$) em função da velocidade e do tempo ($\mathbf{v_p}(\mathbf{x_p}, t)$).

$$\begin{cases} \frac{d\mathbf{x_p}}{dt} = \mathbf{v_p}(\mathbf{x_p}, t) \quad t \in [t_0, T_\mathbf{p}] \\ \mathbf{x_p}|_{t=t_0} = \mathbf{p}, \end{cases} \tag{1}$$

Na equação 1, $\mathbf{p}$ é o identificador (ID) da partícula, que coincide com sua posição inicial em t_0, e $T_\mathbf{p}$ é o instante de tempo final para a trajetória da partícula $\mathbf{p}$ que foi calculada.

O conjunto de dados de entrada inclui a posição inicial das partículas, a geometria do domínio decomposta em células (tetraedros, cubos, etc.) e um vetor velocidade associado aos vértices das células [2]. As relações existentes, no conjunto de dados, são modeladas como abaixo:

- Pontos (*point-id, coord-x, coord-y, coord-z*): cada ponto de uma geometria possui um identificador (*point-id*) e coordenadas tridimensionais (*coord-x, coord-y* e *coord-z*).

- Geometria (*cell-id, point-list*): no problema em questão é o tetraedro, que é a célula base cada qual possui um identificador (*cell-id*) e um conjunto de vértices associados, representados por uma lista (*point-list*) que contém quatro identificadores de pontos (*point-id*);

- Vetor velocidade (*point-id, time-instant, intensity*): contém o valor da velocidade (*intensity*) para todos os pontos da geometria (*point-id*) em todos os instantes de tempo (*time-instant*);

- Partículas (*part-id, time-instant, point*): cada partícula possui um identificador (*part-id*) e esta posicionada em uma coordenada tridimensional (*point*) em um instante de tempo (*time-instant*);

O primeiro passo do algoritmo, representado como um fluxograma na figura 1, é a definição da quantidade de partículas a serem utilizadas e de suas respectivas coordenadas iniciais (x, y e z).

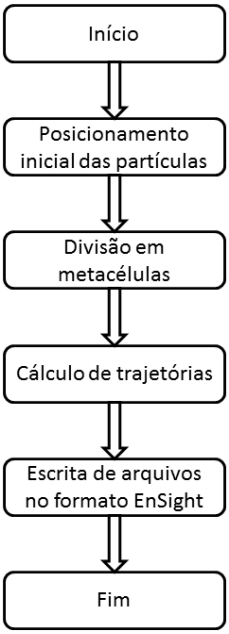


Figura 1. Pipeline geral do algoritmo de trajetória de partículas.

Para calcular a próxima posição da partícula o algoritmo deve encontrar o tetraedro que contém uma dada partícula. Isso faz-se necessário uma vez que precisa-se dos valores de velocidade associados aos vértices para calcular a próxima posição da partícula. Após o cálculo da próxima posição, retorna-se ao problema de descobrir qual tetraedro contém a partícula.

Para reduzir o esforço computacional e o tempo despendido na procura do tetraedro correto, a geometria de entrada é ordenada em grupos, chamados de *metacélulas* que são construídas calculando o baricentro de cada tetraedro e, em seguida, ordenando-os espacialmente, usando o baricentro como referência. Ao final desta ordenação a geometria de entrada estará dividida em oito metacélulas. Neste momento cada metacélula terá seus tetraedros percorridos para a identificação dos valores mínimos e máximos para os eixos x, y e z. Esse processo de identificação dos mínimos e máximos baseando-se nos vértices dos tetraedros, ao invés de em seus centros, é importante pois elimina a possibilidade de situações anômalas onde uma partícula poderia estar no interior de uma tetraedro de fronteira entre duas ou mais metacélulas.

Um cuidado decorrente do processo de criação das metacélulas é a sobreposição de suas fronteiras que pode ser vista na figura 2 onde a metacélula marcada na cor azul possui $xMin = 1$ e $xMax = 3$ e a metacélula marcada na cor laranja possui $xMin = 2$ e $xMax = 2$. Essa deficiência obriga o algoritmo de busca da próxima etapa (cálculo de trajetória) a procurar em todas as metacélulas pelo tetraedro que contém a partícula. No pior caso (partícula próxima da fronteira) pode ocorrer a busca em todas as metacélulas e no melhor caso (partícula afastada da fronteira) a busca será apenas em uma metacélula. O tamanho dos tetraedros também exerce influência no processo de busca da próxima etapa (cálculo de trajetória) pois quanto menor o tetraedro menor também será a área de fronteira e, por consequência, a possibilidade de um tetraedro de borda conter uma partícula.

Uma vez encontrada a metacélula que contém a partícula, inicia-se a busca do tetraedro que está contido na metacélula e que contém a partícula. Encontrado o tetraedro, pode-se recuperar os valores de velocidade pertencentes aos seus vértices.

A etapa seguinte do algoritmo é calcular a próxima posição das partículas. Cada partícula pode estar posicionada dentro ou fora de algum dos tetraedros, ou células, do domínio. Os valores do vetor velocidade, em cada um dos vértices, exercem influência sobre a posição da partícula resultando em sua movimentação no interior do tetraedro, figura 3(a), para tetraedros vizinhos, figura 3(b), ou para tetraedros na vizinhança, figura 3(c). As partículas das figuras 3(a), 3(b) e 3(c) sofrem influência das velocidades (V_1, V_2, V_3 e V_4) no instante t_1 o que resulta em uma nova posição da partícula em t_2. A trajetória da partícula é dada pelo conjunto de posições resultantes [2].

Este cálculo é realizado sequencialmente para cada partícula t vezes, onde t é a quantidade de instantes

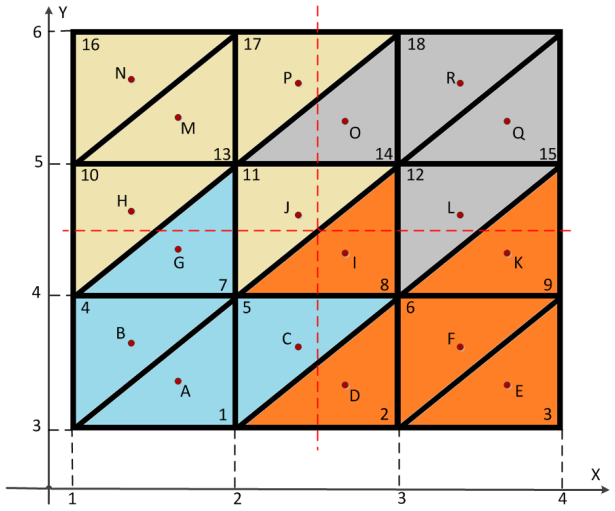
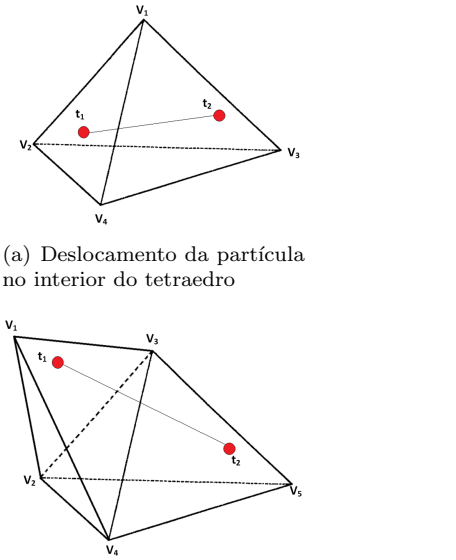


Figura 2. Sobreposição de fronteiras entre as metacélulas



(a) Deslocamento da partícula no interior do tetraedro

(b) Deslocamento da partícula para tetraedro vizinho

(c) Deslocamento da partícula para tetraedro na vizinhança.

Figura 3. Deslocamento da partícula no interior do tetraedro

de tempo da simulação, ou seja, toda a trajetória de uma dada partícula p_i é calculada antes de iniciarmos o cálculo da trajetória da partícula p_{i+1}.

A figura 4 mostra o processo executado pelo algoritmo de CTP em sua versão serial.

A complexidade computacional do algoritmo é $O(p_n * cell_n * t_n)$, onde p_n é a quantidade de partículas, $cell_n$ é a quantidade de células (tetraedros) e t_n o número de instantes de tempo da simulação.

3.2 Implementação Paralela

A figura 5 mostra o fluxograma da versão paralela do CTP que foi implementada utilizando-se CUDA e que será detalhada a seguir.

No passo 1 ocorre a alocação e inicialização do vetor que irá conter todas as trajetórias na memória principal.

O passo 2 faz a leitura e a alocação em CPU da geometria (pontos e células) de todas as metacélulas em memória principal. Após a leitura, as variáveis que serão utilizadas para representar as metacélulas, seus pontos e células são copiadas para a GPU.

No passo 3 segue-se a alocação em GPU do vetor que irá conter os valores de velocidade e pressão de cada ponto em um dado instante de tempo. Cada instante de tempo será carregado em memória principal e depois enviado à GPU, onde todas próximas posições das partículas serão calculadas. *Essa é a principal diferença entre as versões serial e paralela do CTP* e pode ser vista na figura 6. Neste ponto a GPU possui todos os dados necessários para a cálculo do deslocamento das partículas para o instante corrente e então é chamado o método de configuração do *kernel* CUDA.

Cada bloco pode possuir até 512 *threads* mas para a aplicação em questão este número cai para 192 devido a restrições de hardware. No passo de número 4 executa-se a divisão da quantidade de partículas pela quantidade de máxima de *threads* que um bloco comporta, definindo-se a quantidade de blocos que serão executados. Em outras palavras, é gerado um *thread* por partícula e os *threads* são distribuídos entre os blocos. Após a configuração das dimensões dos blocos e do *grid* chama-se o *kernel* CUDA e todo o processamento, a partir deste ponto, é executado por *threads* assíncronas em GPU.

O passo de número 5 tem início pela localização da metacélula que contém a partícula através da comparação entre os limites da metacélula e a coordenada

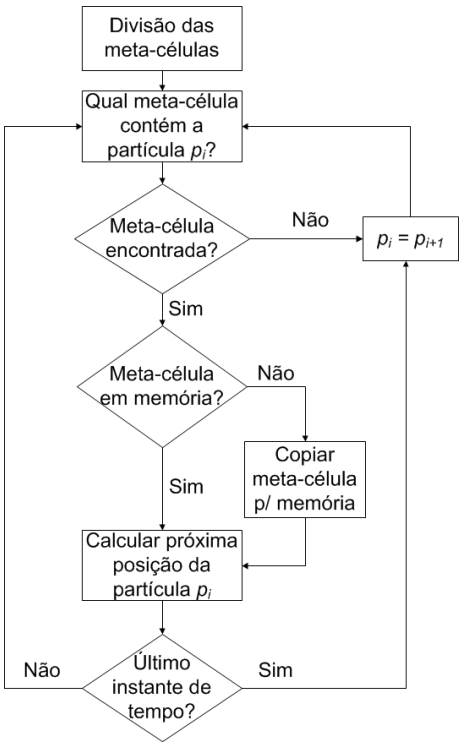


Figura 4. Fluxograma do algoritmo de CTP em CPU

da partícula. Observa-se a necessidade de utilização de um laço *for* que serve para percorrer todas as metacélulas devido à existência de sobreposição de limites entre elas. Após localizar a metacélula tem início a busca por células candidatas a conter a partícula. Uma célula candidata é aquela onde a distância entre todos os seus vértices e a partícula é menor do que o comprimento da maior aresta e que foi calculado na etapa de divisão das metacélulas. Esta operação é menos custosa do que verificar se uma partícula está ou não dentro de um tetraedro.

O último passo determina efetivamente se a partícula encontra-se no interior do tetraedro e os valores de contribuição de cada vértice para o movimento da partícula e pode ser visto na equação 2. Este peso, ou valor de contribuição, varia entre 0 e 1 para cada vértice desse modo, 1 indica que a partícula encontra-se sobre o vértice verificado e a medida que se afasta o valor tende à 0. Valores maiores do que 1 e menores do que 0 indicam que a partícula encontra-se fora da célula candidata.

$$\begin{cases} a1 = 1 - a_2 - a_3 - a_4 & a_4 = \frac{Produto(N_1,V_1)}{Produto(N_1.V_4)} \\ a_2 = \frac{Produto(N_3,V_1)}{Produto(N_3.V_2)} & a_3 = \frac{Produto(N_2,V_1)}{Produto(N_3.V_3)} \end{cases} \quad (2)$$

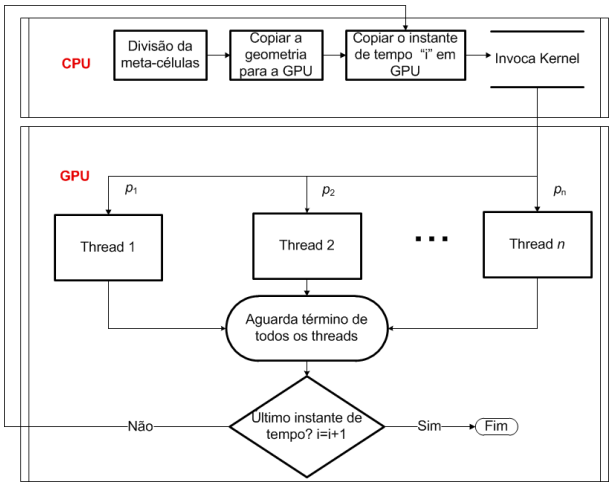
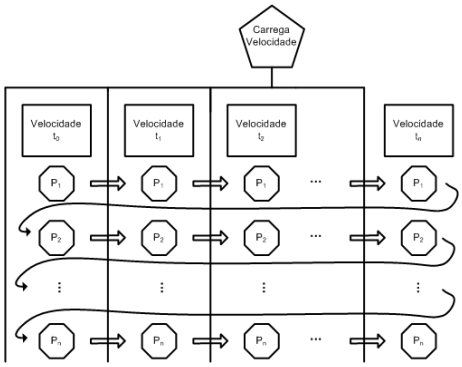
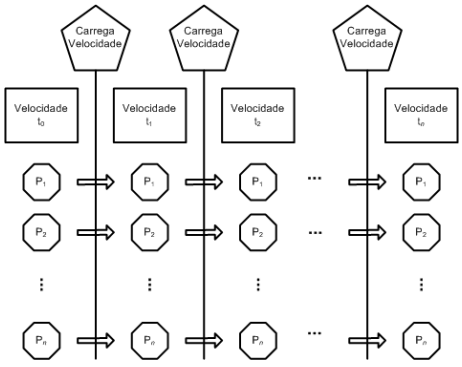


Figura 5. Fluxograma do algoritmo de CTP em GPU.



(a) CPU



(b) GPU

Figura 6. Comparação da execução das versões serial e paralela do CTP

Ao final têm-se quatro valores (a_1, a_2, a_3 e a_4) representando a contribuição de cada vértice para o deslo-

camento da partícula. Caso a partícula esteja situada sobre uma aresta ou algum dos vértices o algoritmo irá considerar que a mesma esta dentro do tetraedro e dará continuidade ao cálculo, outros tetraedros não serão verificados.

No passo de número 6 tem-se a duração de cada instante de tempo (*delta*), a coordenada de uma dada partícula (*actual_part*) bem como os valores de velocidade em cada vértice da célula ($VecListSearch$) e seus valores de contribuição (a_1, a_2, a_3 e a_4).

Por fim, foi feita a validação dos dados obtidos pela versão paralela comparando-a com a obtida na versão serial.

4 Resultados

Para a avaliação de desempenho foi utilizado um computador com processador Intel Quad Core Q9450 @ 2.66 GHz com 8 GB de RAM e uma *GPU* NVIDIA GeForce 9600 GT que possui 8 multiprocessadores de *threads*, cada qual com 8 processadores escalares, resultando em 64 processadores escalares. O diagrama de distribuição de processadores de *threads* desta GPU pode ser vista na figura 7. Somente um dos núcleos do processador Quad-core foi utilizado para a execução do CTP.

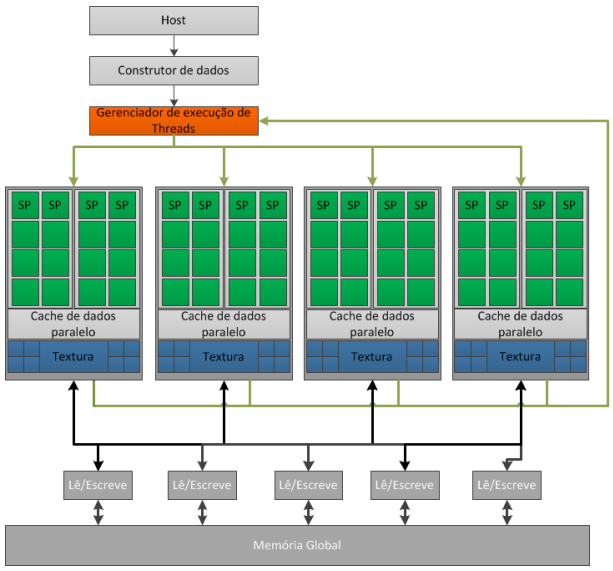


Figura 7. Diagrama da GeForce 9600GT

Os softwares utilizados para o desenvolvimento e para a avaliação de desempenho das versões serial e paralela do CTP foram o Linux OpenSuse 11.0 64 bits, CMake, *VTK* (software destinado à visualização dos resultados), Eclipse e o *SDK* versão 2.3 da NVIDIA.

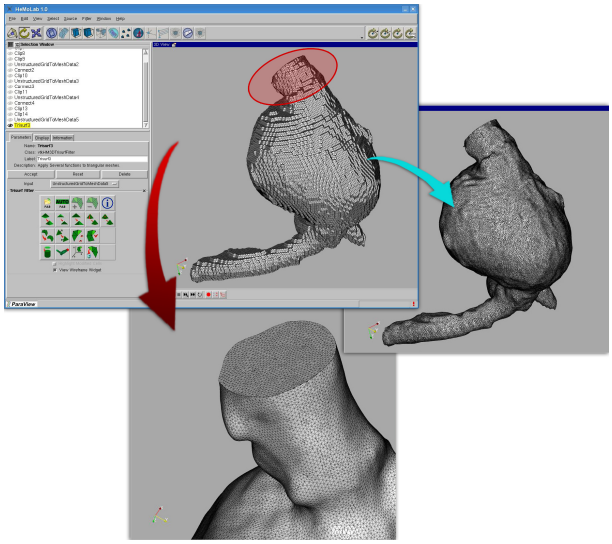


Figura 8. Malha de tetraedros

A simulação utilizada como entrada de dados possui as seguintes características:

- Número de pontos igual a 93.517;
- Número de tetraedros igual a 484.495;
- A maior aresta possui comprimento igual a 0,081734 cm;
- Número de instantes de tempo igual a 101;
- O arquivo descritor da geometria possui 10,7 megabytes (MB);
- Os arquivos descritores dos vetores (velocidade, pressão e deslocamento) para todos os instantes de tempo possuem 252,2 megabytes (MB);

A malha de tetraedros utilizada na simulação é mostrada na figura 8.

O *speedup* obtido em relação à uma única CPU pode ser observado na figura 9. Chegou-se, para o caso de 1536 partículas, a um *speedup* de 56,12, o que corresponderia a uma diminuição de 98,2% no tempo total. Eram necessárias aproximadamente 7 horas para processar o CTP em CPU. Com o uso de GPU, o processamento total passou a ser executado em cerca de 7 minutos. Ao comparar os resultados obtidos com um dos trabalhos relacionados mesmo considerando-se que algoritmo e *hardware* utilizados por Mittmann *et al* em [9] e que obtiveram um *speedup* máximo de 36x são diferentes dos usados no presente trabalho pode-se notar que os resultados obtidos no presente trabalho foram bons.

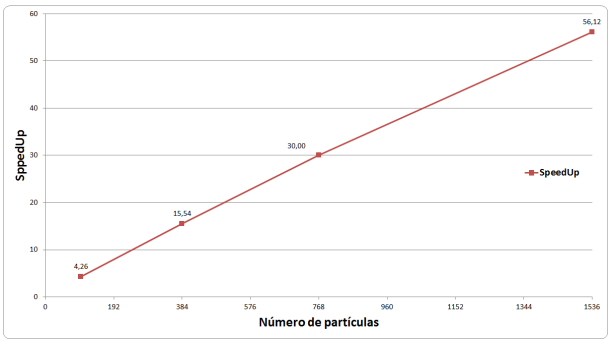


Figura 9. Speedup em relação a uma única CPU

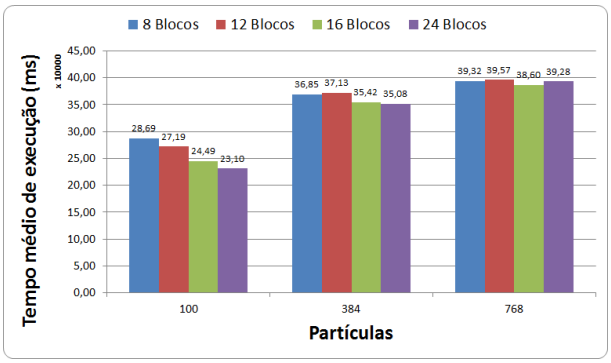


Figura 10. Efeito da distribuição de threads pelos blocos no experimento 3

As *GPUs* com suporte a *CUDA* possuem quantidades limitadas de registradores e de memória que podem ser atribuídos a blocos e *threads*. Foram utilizadas diferentes quantidades de blocos e *threads* na tentativa de determinar o impacto desta característica sobre o desempenho da aplicação. Para os testes foram utilizadas as configurações da tabela 1.

Na tabela 1 a coluna **A** indica os blocos ativos por multiprocessador e a coluna **B** indica os *warps* ativos por multiprocessador.

Partículas	Blocos	Threads/Bloco	A	B
100	8	13	1	1
	12	9	1,5	1,5
	16	7	2	2
	24	5	3	3
384	8	48	1	2
	12	32	1,5	1,5
	16	24	2	2
	24	16	3	3
768	8	96	1	3
	12	64	1,5	3
	16	48	2	4
	24	32	3	3

Tabela 1. Configurações de blocos e de threads utilizadas.

Como expresso na tabela de capacidade de computação encontrada em [10], a GeForce 9600GT pode ter em cada um de seus 8 multiprocessadores até 768 *threads* ativos ou 24 *warps* ativos. A quantidade total de *warps* multiplicado pelo tamanho de cada um (32), totaliza 768 *threads* que é maior do que o número máximo de *threads* por bloco (512) permitido. Logo, conclui-se que não ocorre paralelização no processamento de blocos e sim de seus *warps*.

Na tabela 1 pode-se ver os números de blocos ativos em cada multiprocessador. Estes valores foram calculadas dividindo-se o número de blocos pela quantidade de multiprocessadores existentes na GPU utilizada. Para as configurações de blocos que utilizam números não múltiplos do número de multiprocessadores ocorrerá um desbalanceamento do número de blocos atribuído a cada multiprocessador. Esta situação pode ser vista na figura 11 onde para 12 blocos metade dos multiprocessadores terão 2 blocos e a outra metade apenas 1 bloco.

Figura 11. Distribuição dos blocos não múltiplos dos multiprocessadores.

Para calcular os números de *warps* ativos em cada multiprocessador, primeiramente deve-se lembrar que *warps* que não possuem 32 *threads* serão completados com *threads* extras pelo *runtime* CUDA. Desse modo, para identificar a quantidade de *warps* ativos em um multiprocessador deve-se dividir a quantidade de *threads* de um bloco por 32 e realizar um arredondamento para cima, pois o *runtime* também o realiza, e multiplicar o resultado pela quantidade de blocos ativos em cada multiprocessador. A tabela 1 mostra esses valores.

Nota-se que o desempenho ficou prejudicado quando o número de blocos, neste caso 12 blocos, não é múltiplo da quantidade de multiprocessadores, neste caso 8. Para a hipótese de 100 partículas nota-se que este fator não influenciou decisivamente o desempenho

da aplicação aparentemente devido a pequena quantidade de partículas.

O mecanismo de escalonamento da GPU não escalona *threads* mas sim *warps*. Toda vez que um dos *warps* realiza um acesso à memória global ocorre o escalonamento, se houver *warps* suficientes, na tentativa de ocultar a latência da memória com processamento. Logo, quanto maior a quantidade de *warps* ativos em um multiprocessador melhor esse mecanismo funcionará. Isso pode ser visto pela comparação da tabela 1 com a figura 10 para os casos de 384 e 768 partículas, onde nota-se que quanto maior a quantidade de *warps* ativos melhor o desempenho da aplicação. Ao processar 768 partículas em 24 blocos de 32 *threads* observa-se na tabela 1 que a quantidade de *warps* ativos, neste caso é 3. Enquanto que na linha imediatamente acima tem-se 4 *warps* ativos. Pela comparação destas duas linhas com suas respectivas colunas na figura 10, nota-se que com 4 *warps* ativos o tempo de processamento foi melhor do que com 3 *warps* ativos. Justificando a inferência de que o aumento de *warps* ativos pode melhorar o desempenho da aplicação desenvolvida. Outra forma de atingir um melhor desempenho pode ser a minimização das divergências que ocorrem quando um dos *warps* ativos atinge um desvio condicional utilizando-se a estrutura de dados mencionada por Coutinho *et al* em [4] como *Mapa de Divergências*.

5 Conclusão

Este trabalho demonstrou como a computação de propósito geral em placas gráficas pode oferecer ganhos de desempenho às aplicações de visualização científica através da implementação da técnica de Cálculo de Trajetória de Partículas em *GPU* com o uso da arquitetura *CUDA*.

A análise de diferentes configurações da implementação paralela levou a conclusão de que o desempenho atinge os melhores índices pela maximização dos *warps* ativos em cada multiprocessador. Além disso, verificou-se que para número de blocos que não são múltiplos da quantidade de multiprocessadores, o desempenho é degradado.

Elencam-se como trabalhos futuros: a utilização de memória compartilhada e memória constante no presente algoritmo; a minimização das divergências relativas ao desvio condicional existente no código; e a minimização da quantidade de instruções adicionais utilizadas no controle dos laços através da técnica de desdobramento de laços.

Referências

[1] K. Asanovic, R. Bodik, B. C. Catanzaro, J. J. Gebis, P. Husbands, K. Keutzer, D. A. Patterson, W. L. Plishker, J. Shalf, S. W. Williams, and K. A. Yelick. The landscape of parallel computing research: A view from berkeley. Technical Report UCB/EECS-2006-183, EECS Department, University of California, Berkeley, Dec 2006.

[2] E. Camargo, P. J. Blanco, R. A. Feijóo, and R. L. S. Silva. Efficient implementation for particle tracing in computational hemodynamics. In *Métodos Numéricos e Computacionais em Engenharia - CMNE CILAMCE 2009*, Búzios, Brazil, november 2009.

[3] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, and K. Skadron. A performance study of general-purpose applications on graphics processors using cuda. *Journal of Parallel and Distributed Computing*, 68(10):1370 – 1380, 2008. General-Purpose Processing using Graphics Processing Units.

[4] B. Coutinho, D. Sampaio, F. Pereira, and W. Meira. Performance debugging of gpgpu applications with the divergence map. In *Computer Architecture and High Performance Computing (SBAC-PAD), 2010 22nd International Symposium on*, pages 33 –40, oct. 2010.

[5] D. E. M. de Oliveira. Otimização das aplicações de visualização científica usando o qef. Master's thesis, Instituto Militar de Engenharia, 2010.

[6] F. Hecht, P. J. Mucha, and G. Turk. Virtual rheoscopic fluids. *IEEE Transactions on Visualization and Computer Graphics*, 16:147–160, 2010.

[7] P. Kondratieva, J. Krüger, and R. Westermann. The application of gpu particle tracing to diffusion tensor field visualization. In *IEEE Visualization*, 2005.

[8] I. Larrabide and R. A. Feijóo. HeMoLab: Laboratório de Modelagem em Hemodinâmica. Technical Report 13/2006, Laboratório Nacional de Computação Científica, 2006.

[9] A. Mittmann, M. Dantas, and A. von Wangenheim. Design and implementation of brain fiber tracking for gpus and pc clusters. In *Computer Architecture and High Performance Computing, 2009. SBAC-PAD '09. 21st International Symposium on*, pages 101 –108, 28-31 2009.

[10] NVIDIA. *CUDA Programming Guide*. NVIDIA Corporation, 2701 San Tomas Expressway, Santa Clara, CA 95050, 3.2 edition, 10 2010.

[11] F. Porto, G. A. Giraldi, J. C. de Oliveira, R. L. S. Silva, and B. Schulze. Codims: an adaptable middleware system for scientific visualization in grids. *Concurrency - Practice and Experience*, 16(5):515–522, 2004.

[12] A. V. Xavier. Animação de fluidos via atômatos celulares e sistemas de partículas. Master's thesis, Laboratório Nacional de Computação Científica, 2006.