Paralelização do Algoritmo *Parametrizações Progressivas* em Arquiteturas *Multicore*

Anderson Cunha Kranen[1], **Andriele Busatto do Carmo**[1]

[1]Universidade do Vale do Rio dos Sinos (Unisinos)
Avenida Unisinos, 950, Cristo Rei – São Leopoldo – RS – Brazil

ackranen@edu.unisinos.br, acarmo@unisinos.br

Resumo. *O algoritmo de parametrizações progressivas resolve de maneira eficiente diversos problemas encontrados na área de computação gráfica. Por ser um problema que demanda poder de processamento, o objetivo deste trabalho é paralelizar o algoritmo para execução em plataformas multicore, utilizando a biblioteca Intel TBB. Os resultados mostraram redução no tempo de execução, mas com pouca escalabilidade para os casos de teste utilizados.*

1. Introdução

O processo de parametrização de malhas consiste em criar um mapeamento entre uma superfície 3D e outra superfície similar para diversos tipos de aplicações. Esse processo é fundamental para a computação gráfica, pois processos de mapeamento de textura, deformação de superfície, animações, entre outras aplicações necessitam de um mapeamento de alta qualidade entre as superfícies [Telau 2012, Aigerman et al. 2014].

A parametrização de malhas, quando utilizada na computação gráfica, demanda alto poder computacional devido ao rápido tempo de resposta requisitado. Para isso, a busca por diferentes tipos de algoritmos para realizar a parametrização de malhas de maneira mais eficiente é constante. O mais recente deles, *parametrizações progressivas*, apresenta um método onde a convergência para o resultado ideal é mais rápida e necessita de menos passos. Os benefícios que a redução do tempo de execução da parametrização traz podem ser maximizados ao executar o programa em paralelo.

Este trabalho descreve a implementação inicial da paralelização do algoritmo de *parametrizações progressivas* em processadores *multicore*. Além disso, são apresentadas e comparadas medições com diferentes quantidades de *threads*. O artigo está estruturado da seguinte forma. A Seção 2, apresenta os trabalhos relacionados. A Seção 3, descreve o funcionamento e a implementação do algoritmo de parametrizações progressivas. A Seção 4, explica a paralelização. A Seção 5, descreve os experimentos e apresenta os resultados iniciais. E, por fim, a Seção 6 apresenta as considerações finais.

2. Trabalhos Relacionados

O critério utilizado para seleção dos trabalhos relacionados foi: algoritmos que resolvam o problema da parametrização de malhas e que apresentem implementação para arquiteturas paralelas (CPU e GPU, especificamente). Desta forma, o primeiro trabalho relacionado foi desenvolvido por [Athanasiadis and Fudos 2011] e apresenta um algoritmo de parametrização de superfícies esféricas implementado para executar em CPU e GPU. Para viabilizar a implementação, um algoritmo de redução de energia foi aplicado combinado

com um passo de otimização não linear. Essa combinação garantiu um ponto de parada do processamento quando a malha não estivesse mais sendo otimizada, produzindo assim um resultado válido. A implementação foi feita em OpenCL 1.1 e o mesmo código utilizado em CPU e GPU.

O segundo trabalho relacionado foi desenvolvido por [Guzik and Riley 2018], e apresenta a paralelização de um algoritmo de refinamento adaptativo em malhas utilizando CPU e GPU. A paralelização foi realizada utilizando funcionalidades de vetorização de código do *hardware* e considerando que a limitação do algoritmo é a banda do acesso à memória. Foi utilizado CUDA 9.0 para paralelização em GPU e OpenMP para a paralelização em CPU.

A contribuição do presente trabalho é dada pela paralelização do algoritmo *parametrizações progressivas* para plataformas *multicore*. O algoritmo foi escolhido pelos bons resultados que apresenta em comparação com a literatura existente.

3. Algoritmo de Parametrizações Progressivas

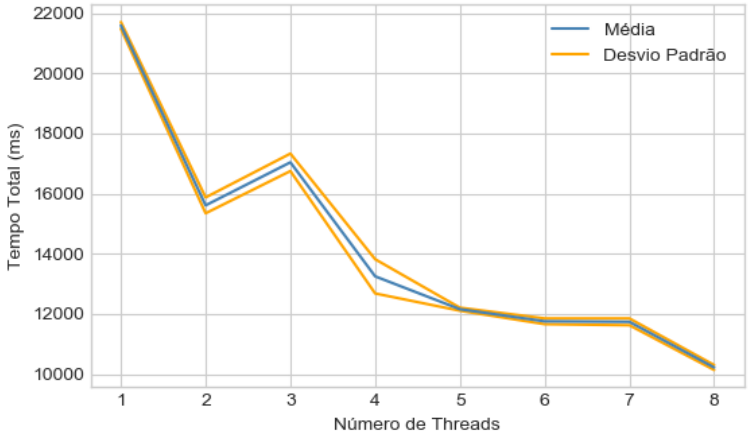
Ao contrário de outros métodos de parametrização que focam em resolver a otimização, o algoritmo de parametrização progressiva foca em observar a distorção ocorrida em cada triângulo da malha através de métricas de medição de distorção. O ponto chave do algoritmo é utilizar triângulos de uma malha de referência intermediária entre a malha original e a malha planar para determinar a energia de distorção. Os estudos de [Liu et al. 2018] afirmam que em comparação com outros métodos de parametrização, o método apresentado é mais simples e com melhor desempenho.

Para obter a distorção da parametrização baseada nos triângulos ideais, é necessário aplicar uma equação não linear com restrições não lineares e não convexas. Para resolver essa equação, o autor propôs um método híbrido utilizando os métodos já existentes: SLIM (*Scalable Locally Injective Mappings*) [Rabinovich et al. 2017] e CM (*Composite Majorization*) [Shtengel et al. 2017]. Ambos possuem características distintas e devem ser usados em determinados passos do algoritmo. O método SLIM é mais apropriado para reduzir distorções mais graves no começo do processo de minimização de distorção, porém possui baixa taxa de convergência para a otimização ideal. O método CM não consegue reduzir superfícies altamente distorcidas, mas possui alta taxa de convergência para a otimização ideal. Portanto, o método SLIM é utilizado nas primeiras iterações do algoritmo, e o método CM é utilizado para convergir para o fim rapidamente quando as distorções mais graves já foram eliminadas [Telau 2012].

4. Paralelização do Algoritmo Parametrizações Progressivas

Para identificar os pontos de necessidade de paralelização, o código original do algoritmo foi instrumentado a fim de medir o seu tempo de execução. As etapas selecionadas foram: o tempo de inicialização do modelo 3D no programa; a fase de pré processamento; e os laços de processamento dos métodos SLIM e CM. Além disso, foi medido o tempo total de execução do programa. Após a determinação dos pontos onde o programa necessitava de mais tempo para executar, a estratégia *map* foi aplicada utilizando o *template parallel_for* da biblioteca Intel TBB. A implementação original do algoritmo faz uso da biblioteca PARDISO *Solver* para paralelizar funções que resolvem sistemas de equações lineares. O uso da biblioteca foi mantido.

Figura 1. Comparação da Execução com Múltiplas *Threads*



Para o particionamento dos dados nos laços onde o *parallel_for* foi aplicado, foi utilizado o particionador *auto_partitioner*. O particionador escolhido tenta balancear a carga de trabalho entre as *threads* dinamicamente. É ideal para execuções com cargas de trabalho de tamanhos diferentes.

5. Experimentos e Resultados Preliminares

Os experimentos foram executados em uma máquina com processador Intel(R) Core(TM) i7-6700HQ com frequência máxima de 3,50GHz em modo turbo e 16GB de RAM DDR4. O sistema operacional utilizado foi o Windows 10 RS3 e o compilador Visual C++ 14.1. A versão da biblioteca Intel TBB utilizada foi a 2019.0. O código foi compilado com a diretiva /O2. A CPU possui 4 *cores* físicos e tecnologia *Hyper Threading*, permitindo que 8 *threads* sejam executadas. A máquina foi configurada para rodar em perfil de desempenho, com o processador no pico de seu funcionamento em toda a execução. Para validar a paralelização do algoritmo, foram utilizados os mesmos *datasets* de modelos 3D da pesquisa original, que estão disponíveis na página dos autores [Liu et al. 2019]. De cada um dos *datasets*, foram selecionados 5 objetos 3D que possuíam maior custo de processamento, totalizando 15 modelos.

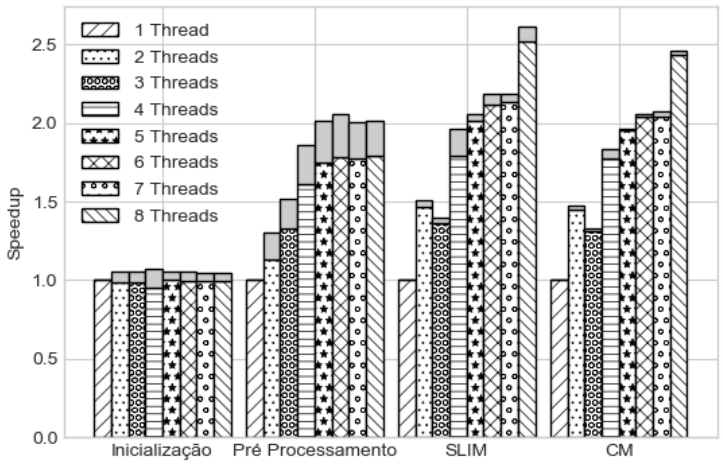
A Figura 1 mostra a redução no tempo de execução de um determinado modelo 3D, onde uma *thread* representa a implementação original do algoritmo. A Figura 2 mostra o *speedup* obtido em cada uma das iterações em cada etapa do algoritmo pela quantidade de *threads* usada. O *speedup* foi calculado medindo o tempo de execução de cada etapa no algoritmo original e no algoritmo paralelizado.

6. Considerações Finais

O algoritmo mostrou uma redução no tempo total de execução para o tipo de aplicação e no contexto que elas se aplicam. Conforme a Figura 1, a redução no tempo de execução total do modelo selecionado foi de 53% quando comparamos a execução em uma e oito *threads*. A paralelização inicial demonstrou que é possível reduzir o tempo de execução da aplicação. O *speedup* atingido não foi o ideal para o número de *threads* utilizado nos testes. Dentre os fatores que podem ter influenciado negativamente o desempenho

nessa aplicação estão o tempo de inicialização da biblioteca de sistemas lineares, a falta de controle sobre as *threads* do sistema operacional e o desalinhamento da memória na *cache*.

Figura 2. *Speedup* Obtido com a Nova Implementação



Referências

Aigerman, N., Poranne, R., and Lipman, Y. (2014). Lifted Bijections for Low Distortion Surface Mappings. *ACM Transactions on Graphics*, 33(4):1–12.

Athanasiadis, T. and Fudos, I. (2011). Parallel Computation of Spherical Parameterizations for Mesh Analysis. *Computers & Graphics*, 35(3):569–579.

Guzik, S. M. and Riley, J. (2018). Adaptive Mesh Refinement on Parallel Heterogeneous (CPU/GPU) Architectures. In *AIAA Aerospace Sciences Meeting*, pages 1–14. American Institute of Aeronautics and Astronautics.

Liu, L., Ye, C., Ni, R., and Fu, X.-M. (2018). Progressive Parameterizations. *ACM Transactions on Graphics(SIGGRAPH)*, 37(4).

Liu, L., Ye, C., Ni, R., and Fu, X.-M. (2019). Progressive Parameterizations. http://staff.ustc.edu.cn/ fuxm/projects/ProgressivePara/. Acessado em: 21 de fevereiro de 2019.

Rabinovich, M., Poranne, R., Panozzo, D., and Sorkine-Hornung, O. (2017). Scalable Locally Injective Mappings. *ACM Trans. Graph.*, 36(4).

Shtengel, A., Poranne, R., Sorkine-Hornung, O., Kovalsky, S. Z., and Lipman, Y. (2017). Geometric Optimization via Composite Majorization. *ACM Trans. Graph.*, 36(4):1–11.

Telau, A. C. (2012). *Parametrizações de Superfícies Triangulares*. PhD thesis, Universidade Federal do Espirito Santo.