Relatório Final de Projeto

Vicente Helano Feitosa Batista Sobrinho
vicente.sobrinho@ufca.edu.br

Ikaro Ruan Penha Costa
ikaroruan@outlook.com

Identificação do Projeto

Título
Algoritmos paralelos para triangulações de Delaunay de ordem superior aplicados na modelagem de terrenos

Edital de Referência
PIBIC Edital Nº02/2016

Área do Conhecimento Predominante

Ciência da Computação

Resumo

Diversos problemas científicos recaem no uso de geração de malhas para obtenção de uma modelagem matemática computacional. Dentre essas malhas, as Triangulações de Delaunay são as mais covenientemente usadas devido suas caracteríscas da qualidade dos triângulos formados. Muitos problemas modernos envolvem variadas superfícies e formas a serem modeladas, assim como exigem maior acurácia em seus cálculos, o que requer maior desempenho computacional. Desse modo, uma alternativa relacionada à melhora na performance de tais algoritmos consiste na Computação Paralela. Além disso, códigos em paralelo, por ventura, necessitam de sincronizações para evitar possíveis erros lógicos, o que implica no estudo das estratégias de travas mais adequadas para a estrura de dados que armazena a triangulação. Portanto, este trabalho tem o objetivo de produzir implementações

paralelas de triangulações de Delaunay, de modo a permitir avanços no desempenho de suas construções.

Palavras-chave: Triangulação de Delaunay, algoritmos paralelos, estratégia de travas.

Introdução

Dentre as malhas triangulares, as triangulações de Delaunay representam aquelas que possuem triângulos de boa qualidade em seu interior, ou seja, configuram-se por ângulos internos maximizados, e de menor qualidade no entorno de sua fronteira. Isso dá-se pela propriedade do circuncírculo vazio, em que garante que o circuncírculo formado por qualquer triângulo de uma triangulação T não conterá outro ponto de T em seu interior.

Por conseguinte, mesmo podendo o algoritmo de Bowyer-Watson [1, 2] construir uma triangulação de Delaunay em $O(nlogn)$, algumas ocasiões podem demandar elevados tempos de execução, a depender da quantidade e da disposição espacial de seus pontos. Além disso, considerando a conjuntura atual de microprocessadores, em que se pode ter chegado a um limite de desempenho, percebe-se a necessidade do uso de *Computação Paralela* para construção de triangulações com alto desempenho. A partir do paralelismo é possível repartir diferentes operações entre *threads* (unidades de processamento), com o intuito de obter reduções nos tempos de execução de uma triangulação. A essas reduções dá-se o nome de *speedup*.

Além disso, o algoritmo de inserção proposto por Bowyer-Watson configura-se por uma adição incremental de pontos. Ao inserir um ponto P na triangulação, este poderá estar em desacordo com a propriedade do circuncírculo. Logo, ao conjunto de faces de T em que P viola tal propriedade, dá-se o nome de *região de conflito*. Essa região compõe-se das faces que sofrerão alterações ao longo da inserção de P.

Contudo, considerando a paralelização da inserção, diferentes threads não poderão modificar mesmas regiões da triangulação, pois, como não se pode ter certeza da ordem de execução dos threads, alterações concorrentes de um mesmo elemento (face ou vértice) podem ocasionar em resultados divergentes da correta triangulação a ser formada. Consequentemente, necessita-se de um modelo de sincronização baseado em travas, que garantem a modificação ou o acesso único de threads por vez. A partir disso, nota-se também a necessidade do estudo de mecanismos de

travas que sejam capazes de oferecer speedups eficientemente de acordo com a implementação paralela em questão.

Justificativa

Devido à grande aplicabilidade de triangulações de Delaunay em problemas científicos, especialmente na modelagem de terrenos, torna-se viável o estudo de técnicas de suas contruções de modo a garantir melhor desempenho. Desse modo, estudos que exigem maior precisão em seus dados demandarão maior poder de processamento, de modo a refletir no tempo de execução do processo de construção da triangulação. Nesse sentido, considerando as atuais limitações no desempenho de processadores, é importante o estudo de técnicas de paralelização da contrução de triangulações de Delaunay com a finalidade de obter melhores desempenhos a partir de microprocessadores com memória compartilhada.

Referencial Teórico

Estudos tem demonstrado bons resultados quanto à paralelização de algoritmos para problemas geométricos, incluindo triangulações de Delaunay. Nesse sentido, Batista et al. [3] demonstrou a possibilidade de paralelização da inserção incremental a partir da CGAL, assim utilizando métodos que permitam maior acesso concorrente de faces na localização, dentre eles o de travas melhoradas. Já Kohout [4] apresenta tanto métodos de inserção paralela diferentes como de travamento e oference, ainda, maiores detalhes em sua tese de doutorado [5]. Cignoni et al. [6] apresentou resultados favoráveis para triangulações de Delaunay tridimensionais a partir da construção incremental.

Vale ressaltar que o `omp_lock_t` foi a trava usada em OpenMP, seguindo especificações apresentadas pelo OpenMP Review Board [7] e sugestões de uso de Chapman et al. [8]. Enquanto o `std::mutex` é a trava usada para implementação em C++ Threads, a partir do sugerido por Williams [9].

Objetivos

O objetivo consiste da implementação de algoritmos paralelos para construção de triangulações de Delaunay. Especificamente dispostos da seguinte maneira:

- Determinar qual plataforma de desenvolvimento é mais adequada para esta situação: OpenMP, Intel TBB ou C++ Threads. (T)
- Implementar os algoritmos paralelos de construção de triangulações de Delaunay de ordem superior usando a *Computational Geometry Algorithm Library* - CGAL. (P)

Metodologia

Inicialmente, foi implementado um algoritmo sequencial para construção de triangulações de Delaunay. Esse foi desenvolvido através de uma estrutura de dados baseada em faces, a qual armazena vértices e faces, assim como cada vértice possui informação de uma face incidente e cada face informações sobre suas vizinhas. A técnica de inserção empregada foi baseada em Bowyer-Watson, chamada de *inserção incremental*, que foi dividida em três fases: localização, busca da região de conflito e atualização. A localização consiste da verificação da posição de um ponto P a ser inserido na triangulação. Assim, de posse da localização, pode-se buscar quais faces terão P no interior seu circuncírculo associado, o que implica na construção da região de conflito. Em seguida, as faces em conflito formarão uma cavidade na qual será inserido P e novas faces serão formadas, consistindo da fase de atualização da triangulação.

Os predicados geométricos para verificação de orientações de pontos e teste do circuncírculo vazio são provenientes da implementação de Shewchuk [10]. Quanto ao armazenamento da triangulação, foi utilizado o *Compact Container* e, para melhor eficiência na inserção incremental, a ordenação espacial dos pontos a partir do *Spatial Sort*, ambos da CGAL [11].

Seguidamente, havia a necessidade de escolher a plataforma de desenvolvimento paralelo para essa situação. Assim, iniciou-se com a implementação de uma estrutura de dados em *lista encadeada* com inserção concorrente usando OpenMP, Intel TBB, Intel CilkPlus e C++ Threads. De modo geral, tanto OpenMP quanto C++ Threads ofereceram maior liberdade para implementações paralelas, assim como travas próprias e ferramentas associadas às particularidades de suas plataformas.

Nesse sentido, a ideia principal é de dividir os pontos a serem inseridos em conjuntos disjuntos, denominados *workloads*, de acesso exclusico por thread. Cada thread executará todos os passos da inserção (localização, região de conflito e atualização) até que tenham inserido todos os pontos de seus respectivos workloads. Entretanto, a necessidade de sincronização surge com a possibilidade de modi-

ficações simultâneas de uma mesmas regiões de conflito.

Para isso, estratégias de travas foram estudadas para escolha da melhor forma de sincronização para inserção de pontos na triangulação. Assim, possuindo cada vértice um *lock* (trava) associado, quando uma face tiver seus três vértices travados, será considerada travada. Dessa forma, consideram-se as seguintes estratégias:

- Trava Simples: para leitura ou modificação de faces, deve-se travar todos os vértices de uma face.
- Trava Simples com Prioridades: cada thread terá uma prioridade para travar os vértices. Caso um vértice esteja travado, o thread de maior prioridade deverá esperar até adquirir propriedade sobre ele, mas se for de menor prioridade estará inapto para travamento, devendo desistir da inserção de P momentaneamente e voltar a inseri-lo posteriormente.
- Trava Melhorada: consiste em travar apenas um vértice da face para operações de leitura (para a localização) e todos os vértices da face para modificação (região de conflito e consequente atualização).

Deve-se notar que a Trava Simples com Prioridade foi baseada em Kohout [4], apesar de não ter tido o comportamento esperado no algortimo aqui implementado, logo não foi considerado. Enquanto a Trava Melhorada em Batista [3], que permite, assim, leitura simultânea durante a fase de localização.

Dessa forma, torna-se importante a análise da eficiência da implementação paralela, de acordo com as estratégias de travas utilizadas, visando a obtenção de speedups.

Resultados e Discussão

De início, as plataformas foram escolhidas foram OpenMP e C++ Threads por permitirem maior liberdade de implementação e maior número de ferramentas, incluindo de sincronizações. Para os teste de execução foram usadas distribuições uniformes de pontos no espaço em quantidade 2^k, em que k varia entre 10 e 20, em uma máquina Dell com processador Intel Core i7-4510U, clock máximo 3,10 GHz, cache de 4MB e 4 CPUs. Nesse contexto, pode-se verificar na Figura 1, a qual apresenta resultados para estratégias de travas simples em paralelo com quatro threads, que os resultados não foram satisfatórios, haja vista que não foram obtidos speedups.

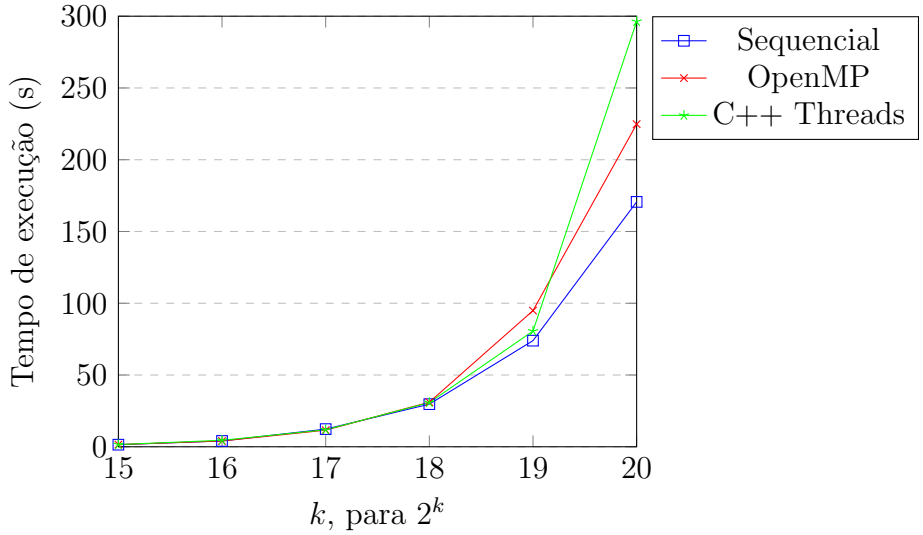

Figura 1: Tempos de execução para implementação senquencial e paralela com travas simples.

Contudo, a partir dos experimentos pode-se perceber que a localização foi a operação que consumiu maior tempo de processamento, como pode ser visto na Figura 2 a qual mostra o consumo de tempo pelas operações de inserção (localização, construção da região de conflito e atualização).

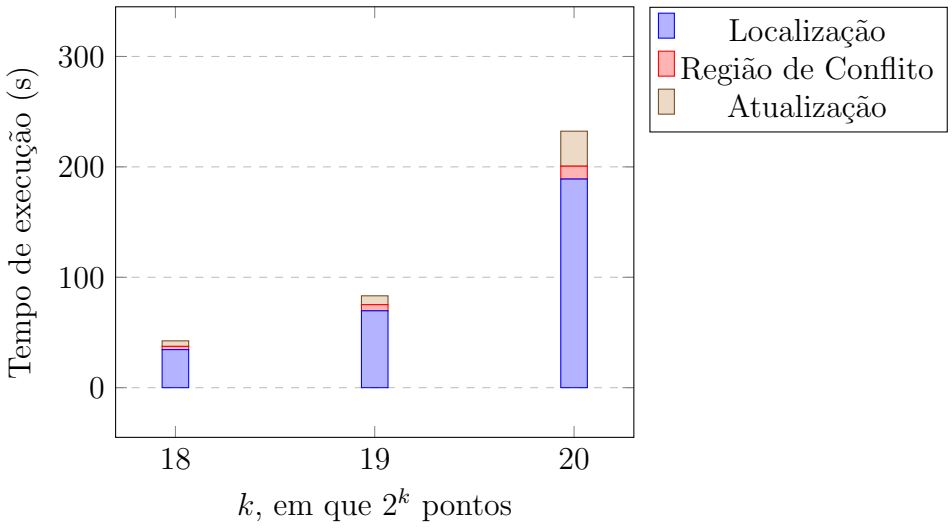

Figura 2: Parcelas de execução de cada etapa da inserção.

Diante disso, percebeu-se que a Trava Simples gerou maiores sincronizações, pois cada thread obtém acesso exclusivo para leitura das faces no instante de localização, sendo que um thread não poderia travar uma face cujos locks já estivessem

em posse de outro thread. Caso isso ocorra, o algoritmo desistirá momentaneamente da inserção daquele ponto, a isso intitula-se de *retreat*, e nova tentativa será feita posteriormente.

Logo, a partir dos dados mostrados pela Figura 3 percebe-se um alto número de retreats ocorrendo na localização e baixo número para região de conflito. Contudo, a localização apenas utiliza-se de leituras dos vértices e faces, o que permite múltiplas leituras com segurança pelos threads, com exceção daqueles travados pela região de conflito. Além disso, deve-se ressaltar que como threads não poderão modificar mesmas faces da triangulação, a construção da região de conflito também poderá ocasionar retreats.

A partir disso, nota-se a necessidade do uso de Travas Melhoradas, que permitem menores sincronizações no processo de localização. Desse modo, a Figura 3 mostra que esta estratégia permitiu uma conveniente redução no número de retreats pela localização. Ademais, quanto à região de conflito houve uma leve redução, mas não considerável.

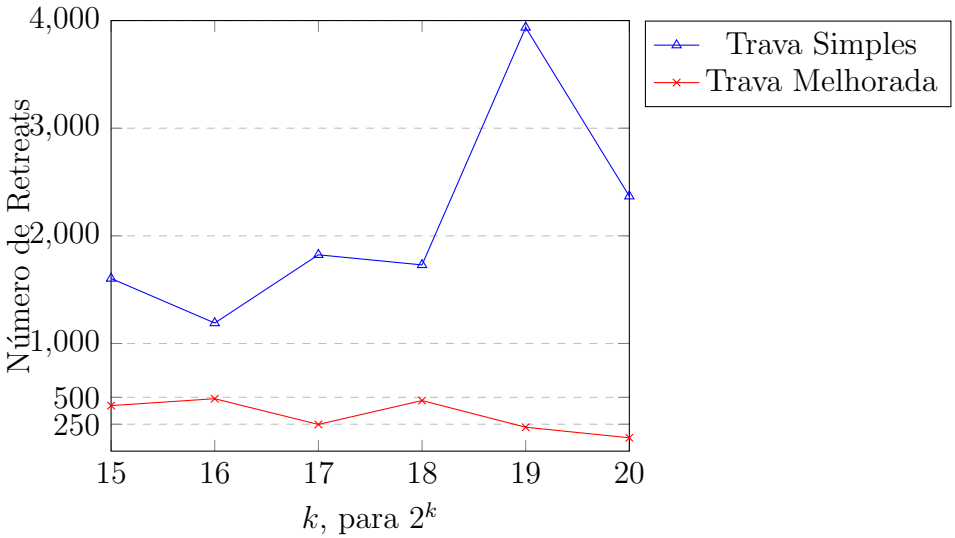

Figura 3: Número de retreats na localização para execução paralela em OpenMP com 4 threads.

Apesar da redução no número de retreats, não houveram melhoras no tempo de execução dos algoritmos paralelos. Apenas a implementação usando C++ Threads apresentou pequena melhora, mas não chegando próximo de um speedup. Tais dados podem ser verificados nas Figuras 4 e 5.

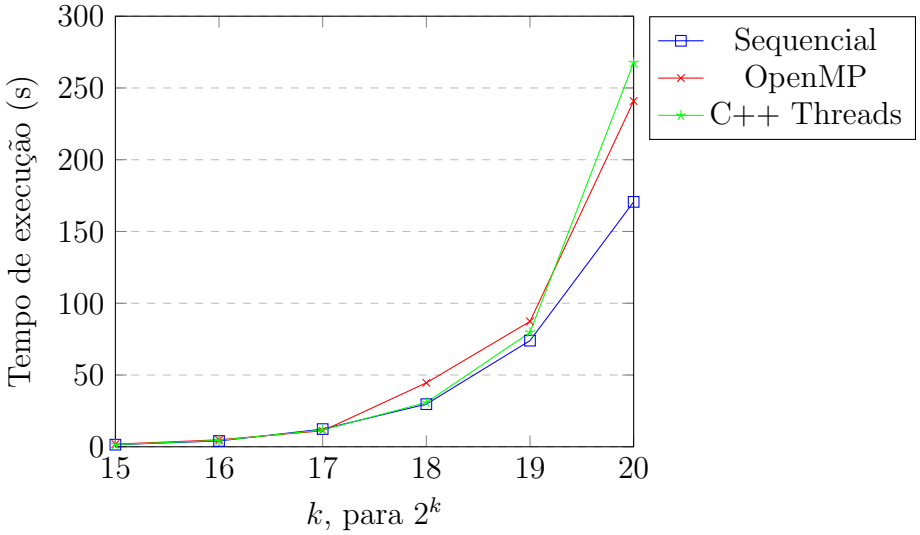

Figura 4: Tempo de execução dos algoritmos sequencial e paralelos com travas melhoradas e 4 threads.

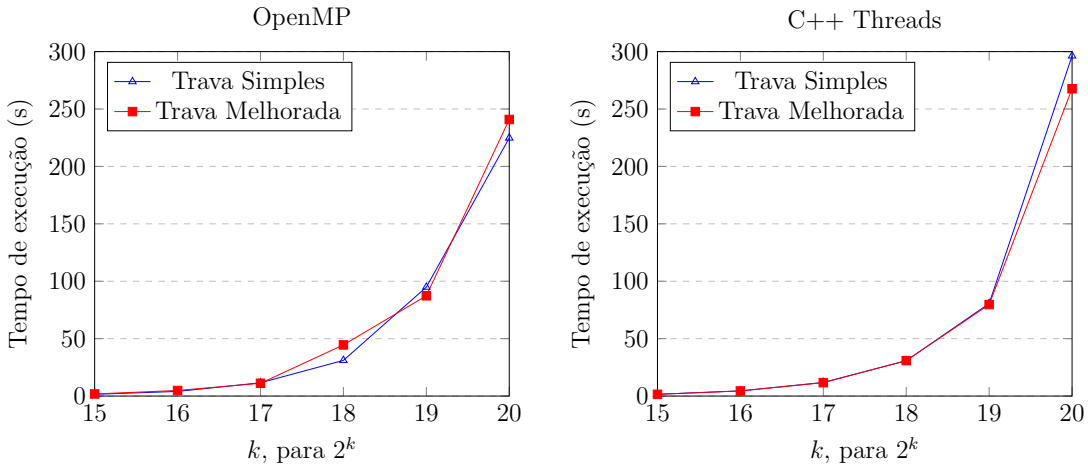

Figura 5: Comparação dos tempos de execução de Travas Simples e Melhoradas.

Nesse sentido, embora melhora no número de retreats, não houve significativa redução no tempo de execução dos algoritmos. Assim, o maior consumo de tempo continua sendo pela localização de pontos.

Conclusão

Ainda que seja uma ferramenta muito útil para ganhos em desempenho computacional, a Computação Paralela necessita diversos estudos e elaborações de estratégias para se obter os melhores resultados. Desse modo, aplicando-a em contruções de triangulações de Delaunay tem sido tema recorrente de estudo atualmante, havendo ainda resultados positivos. Entretanto, apesar das estratégias de sincronização terem performance considerável no gerenciamento de sincronizações, dificuldades na obtenção de tempos de execução foram enfrentadas.

Portanto, haja vista os resultados produzidos por esse trabalho, nota-se a necessidade de uma análisa mais ampla das possívies causas de lentidão geradas nos algoritmos paralelos. Logo, pode-se considerar para o futuro a verifição dos possíveis causadores de menor desempenho na localização de pontos como também o estudo estratégias de travas adequadas ao método de inserção com o intuito de obteção de speedups.

Referências

[1] Adrian Bowyer. Computing dirichlet tessellations. *The computer journal*, 24(2):162–166, 1981.

[2] David F Watson. Computing the n-dimensional delaunay tessellation with application to voronoi polytopes. *The computer journal*, 24(2):167–172, 1981.

[3] Vicente HF Batista, David L Millman, Sylvain Pion, and Johannes Singler. Parallel geometric algorithms for multi-core computers. *Computational Geometry*, 43(8):663–677, 2010.

[4] Josef Kohout, Ivana Kolingerová, and Jiří Žára. Parallel delaunay triangulation in e 2 and e 3 for computers with shared memory. *Parallel Computing*, 31(5):491–522, 2005.

[5] Josef Kohout. *Delaunay triangulation in parallel and distributed environment.* PhD thesis, PhD Thesis, University of West Bohemia, Pillsen, Czech Republic, 2005.

[6] Paolo Cignoni, Claudio Montani, Raffaele Perego, and Roberto Scopigno. Parallel 3d delaunay triangulation. In *Computer Graphics Forum*, volume 12, pages 129–142. Wiley Online Library, 1993.

[7] OpenMP Architecture Review Board. *OpenMP Application Programming Interface*, 2015.

[8] Barbara Chapman, Gabriele Jost, and Ruud Van Der Pas. *Using OpenMP: portable shared memory parallel programming*, volume 10. MIT press, 2008.

[9] Anthony Williams. *C++ concurrency in action*. London, 2012.

[10] Jonathan Richard Shewchuk. Adaptive precision floating-point arithmetic and fast robust geometric predicates. *Discrete & Computational Geometry*, 18(3):305–363, 1997.

[11] The CGAL Project. *CGAL User and Reference Manual*. CGAL Editorial Board, 4.10 edition, 2017.