Universidade Federal do Pampa
Kléber Kapelinski

Aplicação de Processamento Paralelo no Programa RAFEM utilizando OpenMP

Alegrete
2016

Kléber Kapelinski

Aplicação de Processamento Paralelo no Programa RAFEM utilizando OpenMP

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Ciência da Computação da Universidade Federal do Pampa como requisito parcial para a obtenção do título de Bacharel em Ciência da Computação.

Orientador: Prof. Claudio Schepke, Dr.

Alegrete

2016

Kléber Kapelinski

Aplicação de Processamento Paralelo no Programa RAFEM utilizando OpenMP

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Ciência da Computação da Universidade Federal do Pampa como requisito parcial para a obtenção do título de Bacharel em Ciência da Computação.

Trabalho de Conclusão de Curso defendido e aprovado em 23 de junho de 2016

Banca examinadora:

Prof. Claudio Schepke, Dr.
Orientador

Prof. Márcia Cristina Cera, Dr.
UNIPAMPA

Prof. Wang Chong, Dr.
UNIPAMPA

Resumo

O processo de ablação por radiofrequência (ARF) é uma forma de tratamento de hepatocarcinoma que vem ganhando uma utilização gradativamente maior. Para a simulação deste procedimento foi desenvolvido um programa, o RAFEM, que utiliza a modelagem através do método de elementos finitos. Uma desvantagem presente na simulação de ARF com este programa é o alto custo computacional, que gera um baixo desempenho. Este trabalho tem o propósito de melhorar o desempenho do RAFEM através da aplicação de processamento paralelo, utilizando OpenMP. O processo de paralelização é descrito. Com a versão paralela do RAFEM reduziu-se o tempo de execução em até 4 vezes.

Palavras-chave: Ablação por radiofrequência, Método de elementos finitos, desempenho, simulação.

Abstract

The process of radiofrequency ablation is a treatment way to the hepatocellular carcinoma that is achieving an increasing utilization. For the simulation of this procedure a program has been developed, the RAFEM that use the finite element method modeling. A disadvantage of the simulation in this program is the poor performance. This work is to improve the performance of the RAFEM program. The paralelization process is explained. The parallel version of RAFEM took 4 times less execution time.

Key-words: radiofrequency ablation, finite element method, performance, simulation.

Lista de ilustrações

Lista de tabelas

Lista de siglas

API Aplication Programming Interface

ARF Ablação por Radiofrequência

MEF Método de Elementos Finitos

MPI Message Passing Interface

OpenMP Open Multiprocessing

OMS Organização Mundial da Saúde

Pthread POSIX thread

RAFEM Radiofrequency Ablation Finite Element Method

TCC Trabalho de Conclusão de Curso

Sumário

1 Introdução

Atualmente, a saúde é um tema que se apresenta de maneira intrínseca no cotidiano das pessoas, uma vez que, interfere de uma maneira direta em seu estado, desta forma sendo uma influência sobre seus hábitos. De acordo com a OMS (Organização Mundial da Saúde), esta é definida como estado de completo bem-estar físico, mental e social, e não somente a ausência de enfermidade ou invalidez. O estado de saúde de uma pessoa geralmente é afetado por manifestações patológicas que tornam a se apresentar em seu organismo, este estado é caracterizado como doença, e é definido pela OMS como ausência de saúde. Segundo Minayo (1988), a doença define-se como algo que acontece apenas no plano físico, aloja-se em um órgão e assim deve ser tratado. Existem inúmeras doenças atualmente que vem a comprometer a saúde, especificamente, pode-se citar o hepatocarcinoma, que é um mal que afeta gravemente um órgão vital ao organismo humano, o fígado.

1.1 Fígado

O fígado é a maior glândula do organismo, localizando-se ao lado direito do abdômen. É constituído por milhões de células, onde cada uma destas células desempenha uma função específica, que é essencial para o equilíbrio do organismo. O fígado desempenha muitas funções importantes, entre elas tem-se:

- Secreção da bile, que auxilia na dissolução e aproveitamento das gorduras;
- Armazenamento de glicose, onde esta é armazenada sob a forma de glicogênio, ficando à disposição do organismo conforme a necessidade deste;
- Produção de proteínas nobres, como as que são ligadas ao processo de coagulação do sangue;
- Desintoxicação do organismo, onde hormônios e drogas são transformados de forma que o organismo possa excretá-los;
- Sintetizar o colesterol, onde este é metabolizado e então excretado por meio da bile;
- Filtrar micro-organismos, onde várias células do fígado se responsabilizam por segurar bactérias ou outros micro-organismos que transmitem infecções;
- Transformação de amônia em uréia.

1.2 Hepatocarcinoma e tratamento

O hepatocarcinoma é o câncer no fígado, sendo um mau que afeta gravemente este órgão, fazendo mais de meio milhão de vítimas anualmente, e sendo o quinto tipo de câncer que se apresenta mais comumente em homens e o sétimo em mulheres (GOMES et al., 2013). Este câncer se origina a partir dos hepatócitos, que são as principais células do fígado. O surgimento se dá com a mutação genética de uma determinada célula, o que faz com que esta se multiplique de maneira desordenada, formando assim um cisto na região afetada. Em etapas avançadas pode-se ter também a ocorrência de metástases, o que contribui para o agravo do problema. O câncer no fígado é muito agressivo, apresentando um alto índice de óbitos após o início dos sintomas. Como formas para o tratamento do hepatocarcinoma tem-se o transplante do fígado, que, apesar de ser uma boa opção é muito agressivo, bem como também a hepatectomia, onde se retira a porção do fígado onde se encontra o tumor. Outra opção também utilizada de maneira gradativamente maior é a quimioembolização, onde se associa o uso de quimioterápicos com o processo de embolização (SOUZA, 2011), desta forma os vasos sanguíneos que nutrem o tumor são obstruídos, e a injeção de quimioterápicos na região do tumor causam então necrose das células afetadas pelo tratamento. O tumor também pode ser tratado de forma percutânea, onde, com o auxílio de ultrassom, é introduzida uma agulha no centro do tumor e então administrado álcool, que provoca a destruição do câncer. Ainda na modalidade percutânea tem-se também o uso de radiofrequência para se ablar a lesão.

1.3 Ablação por radiofrequência

A ablação por radiofrequência, ou radioablação, é uma forma de tratamento através da pele, onde, utilizando-se de orientação por imagem, inserem-se eletrodos na região do tumor. Através do eletrodo então passa corrente elétrica de alta frequência, criando calor, que vem a destruir as células cancerígenas. Técnicas de diagnóstico por imagem como ultrassonografia e tomografia computadorizada são utilizadas para guiar a agulha para o tumor. As células mortas são gradativamente repostas por tecido cicatricial, que tende a encolher com o tempo.

1.3.1 Procedimento

O paciente é colocado na mesa de exames, onde é conectado aos monitores, que permitem acompanhar a taxa de batimentos cardíacos, a pressão arterial e pulso do paciente durante o procedimento. É então inserido um canal intravenoso em uma veia na mão ou no braço do paciente, assim o medicamento de sedação é dado de forma intravenoso. A área onde os eletrodos serão inseridos será esterilizada e coberta. Caso o procedimento se realize com o paciente acordado então faz-se uso de anestesia local, caso for utilizada anes-

tesia geral, então a respiração do paciente passa a ser feita mecanicamente. Com o uso de orientação por imagem então o eletrodo é colocado através da pele até o local do tumor. Estando o eletrodo no lugar então é aplicada a energia de radiofrequência. No caso de um tumor grande, múltiplas ablações serão necessárias, assim necessitando reposicionar o eletrodo em diferentes partes do tumor para se certificar que não foi deixado nenhum tecido afetado pelo tumor para trás. Ao término do procedimento o eletrodo é retirado e aplica-se pressão para conter qualquer sangramento e aplicam-se curativos na abertura presente na pele. Não necessita-se o uso de suturas (pontos cirúrgicos). Cada ablação por radiofrequência dura entre 10 e 30 minutos. Conta-se também o tempo adicional quando múltiplas ablações são feitas. O procedimento por completo tende a durar entre 1 e 3 horas.

1.3.2 Benefícios

A ablação por radiofrequência é uma técnica minimamente invasiva e que representa uma forte alternativa para pacientes não aptos para o método cirúrgico, sendo extremamente eficaz na eliminação por completo de pequenos tumores no fígado, além do fato de se tratar de um método rápido e também com recuperação rápida por parte do paciente.

1.3.3 Limitações

Existe uma limitação em relação ao volume do tumor que pode ser eliminado com o método de ablação por radiofrequência, isto se deve a limitações de equipamentos. A radioablação também não consegue destruir tumores de tamanho microscópico e não pode prevenir o câncer de crescer novamente.

1.3.4 Simulação computacional

A distribuição de temperatura durante o processo de ARF (Ablação por Radiofrequência) é simulada utilizando-se o MEF (Método de Elementos Finitos), onde se tem uma malha que discretiza o fígado em vários elementos, desta forma, obtêm-se a temperatura presente em cada um dos elementos e, combinando-se estas informações então se tem a distribuição de temperatura na malha (fígado) para um dado instante. A distribuição de temperatura é calculada para cada instante de tempo definido, desta forma mostrando o comportamento da temperatura ao longo do tempo. Quando se utiliza o MEF para modelar ablação com temperatura controlada, a tensão elétrica aplicada será regulada de forma a manter a temperatura no eletrodo constante, o que antes era efetuado de forma manual com métodos de tentativa e erro (HAEMMERICH; WEBSTER, 2005).

1.4 RAFEM

No trabalho de Jiang et al. (2010) foi implementada uma ferramenta para fim de simulação de distribuição de temperatura no fígado durante um processo de ARF. A ferramenta, denominada RAFEM, foi implementada em linguagem C++.

Para cada simulação envolve-se um tempo alto, onde para simular um procedimento de ARF de duração de minutos leva-se um tempo que está na ordem de horas.

1.5 Objetivos

Tendo em vista a implementação do programa RAFEM e a limitação que existe deste em relação ao desempenho, objetiva-se neste trabalho a melhoria de desempenho do RAFEM através da aplicação de processamento paralelo neste. Com isso, reduzindo-se o tempo de simulação, reduz-se também o tempo de espera para um procedimento.

1.6 Objetivos Específicos

Identificar etapas custosas durante a simulação, através disso então reduzindo o tempo de execução destas etapas, de forma que contribua significativamente para o tempo final de simulação.

A aplicação de processamento paralelo será no modelo de memória compartilhada, utilizando-se a API OpenMP.

1.7 Organização do trabalho

Neste TCC será tratado do procedimento de ARF e sua simulação através do RAFEM, bem como sua melhoria de desempenho. No Capítulo 2 serão apresentados os conceitos que envolvem o MEF, o processamento paralelo e também o RAFEM. O Capítulo 3 mostra algumas referências que se relacionam com a modelagem e simulação de ARF, bem como também outras relacionadas a paralelização do MEF. No Capítulo 4 apresenta-se a análise realizada, bem como os detalhes da paralelização do RAFEM. Por fim, o Capítulo 5 apresenta as conclusões e considerações sobre a realização deste trabalho.

2 Fundamentação

Para muitos problemas práticos não existe uma solução analítica, portanto, se utiliza a abordagem numérica, onde a solução se dá por meio de aproximações (MOAVENI, 2003). O método de elementos finitos é a abordagem numérica que é empregada no RAFEM para a simulação de ARF e será descrito a seguir.

2.1 Método de elementos finitos

O MEF é um método numérico que trata da discretização de um dado problema, para a solução deste em parcelas, sendo muito utilizado para a análise de tensões e deformações, transferência de calor, fluidodinâmica e eletromagnetismo. O problema é aproximado por um conjunto de formas geométricas não sobrepostas, chamadas de elementos finitos e, a malha destes elementos finitos caracteriza a representação discreta do problema (SERGEY, 2013). A forma de um elemento é definida pelos nós que compõem o modelo. Os nós representam os pontos de conexão entre elementos adjacentes. Um sistema linear de equações é montado para se determinar a solução dos nós. Este sistema de equações é chamado matriz global de rigidez, que é dada pela combinação das matrizes locais de rigidez. A matriz global de rigidez tende a ser esparsa e tem caráter simétrico. No MEF também tem-se a especificação de condições iniciais e de contorno que vem a impactar no sistema gerado.

Segundo Moaveni (2003), a análise de elementos finitos é composta de 7 passos básicos, que se distribuem entre 3 fases:

- Fase de pré-processamento.
  - Criar e discretizar o domínio de solução em elementos finitos, ou seja, decompor o problema em elementos e nós;
  - Assumir uma função para representar o comportamento aproximado de um dado elemento, ou seja, a sua solução;
  - Desenvolver as equações para um elemento;
  - Montar os elementos para apresentar o problema como um todo. Montar a matriz global de rigidez;
  - Aplicar condições de contorno e condições iniciais.
- Fase de solução.

  - Solução do sistema algébrico de equações simultaneamente para se obter os resultados em cada nó, no caso do problema de transferência de calor, a temperatura em cada nó.

- Fase de pós-processamento.
  - Obter informações relevantes.

2.2 Paralelismo

No processamento paralelo busca-se explorar a sobreposição temporal de tarefas, de maneira a reduzir a ociosidade de componentes de processamento, de forma que, vários processadores trabalhem sobre a execução de um mesmo programa. O paradigma de paralelismo pode se aplicar com memória compartilhada ou distribuída. No modelo de memória distribuída, tem-se o conceito de troca de mensagens associado na comunicação entre os processos. Já no modelo de memória compartilhada a comunicação entre processos se dá através de uma região de memória, que é compartilhada entre estes.

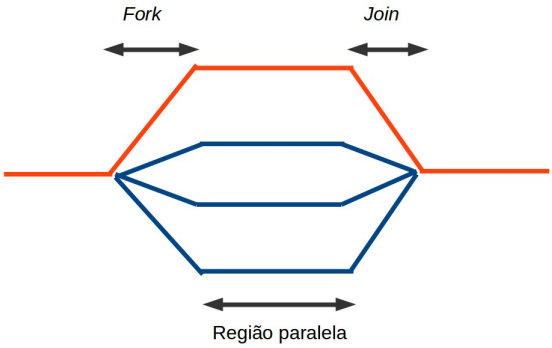
2.2.1 Programação

A arquitetura paralela em uma máquina é um fator muito importante para aplicação de processamento paralelo, no entanto, outro fator necessário para o paralelismo é que o software também deve estar adaptado a este formato de execução. Para isto, existem várias bibliotecas que oferecem suporte à aplicação de paralelismo em programas. No paradigma de memória compartilhada, tem-se como exemplos as bibliotecas Pthread e OpenMP (OPENMP, 2016). Já no paradigma de memória distribuída tem-se como destaque a API MPI (MPI, 2016). O modelo empregado neste trabalho será com memória compartilhada, utilizando OpenMP.

2.2.2 OpenMP

O OpenMP é uma API para programação paralela em modelo de memória compartilhada, que é implementado sobre a biblioteca Pthread, onde a programação ocorre por meio da inserção de diretivas de pré-compilação no código. O OpenMP funciona de acordo com o modelo *fork-join*, mostrado na Figura 1. Uma determinada *thread*, denominada *thread* mestre (representada na cor laranja) executa o código sequencialmente até se aproximar de uma região paralela, a *thread* mestre então faz um *fork*, que é onde surgem várias *threads* para executar aquele segmento de código em paralelo. Quando se atinge o final da região paralela ocorre o *join*, que é a sincronização entre as *threads*, onde para a *thread* mestre voltar a executar sequencialmente todas as *threads* devem terminar a execução da região paralela.

Figura 1 – Modelo *fork-join* do OpenMP



Eventualmente pode acontecer de várias *threads* executarem sobre uma mesma região de memória, o que tende a gerar condições de corrida. Para controlar estas situações o OpenMP provê suporte à regiões críticas, que são regiões especificadas dentro de uma região paralela, que só podem ser executadas por uma *thread* por vez. A desvantagem é que o uso de regiões paralelas tende a serializar a execução naquele trecho de código. O OpenMP também provê mecanismos de dividir iterações de laços `for` entre as *threads*, podendo esta divisão ser de tipo estático, onde cada *thread* já recebe um dado número de iterações para executar, ou dinâmico, onde cada *thread* recebe uma iteração a medida que finaliza a que havia recebido. Na questão de paralelização de laços pode também ocorrer o problema de ordenamento, onde é necessário que determinadas operações sejam executadas em ordem de iterações.

Tendo em vista uso da API OpenMP, temos a questão de análise do funcionamento do programa, para saber etapas que sejam sujeitas à paralelização, e que sejam também custosas. Se faz necessário então compreender o programa em questão, no caso, o RAFEM.

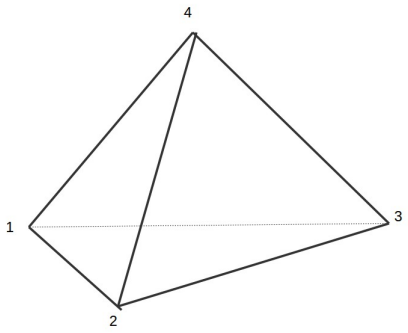
2.3 O programa RAFEM

O RAFEM foi desenvolvido com o propósito de simulação de distribuição de temperatura durante um procedimento de ARF. Para a simulação é necessária uma malha de elementos finitos já modelada. Com base nesta malha o RAFEM computa o problema, levando em conta alguns parâmetros de entrada.

O RAFEM utiliza elementos tetraedrais (Figura 2) para o modelo de elementos finitos. Para isto são tratados dois casos específicos de elementos no programa, os elementos com 4 nós, que são os internos do modelo, e os elementos com 3 nós, que são os elementos presentes em regiões de contorno.

Como entrada o programa toma uma malha de elementos finitos, que é o modelo

Figura 2 – Modelo tetraedral usado no RAFEM



do fígado que será simulada a distribuição de temperatura. Da malha de elementos finitos é dada a informação de quantos elementos e quantos nós compõem o problema, o que é um fator que influencia bastante no custo do problema. Após isto pode ser feito o reordenamento dos elementos, que é uma etapa que ajuda a reduzir muito a complexidade do problema. Posteriormente o usuário deve informar o número máximo de iterações por passo e a duração do procedimento.

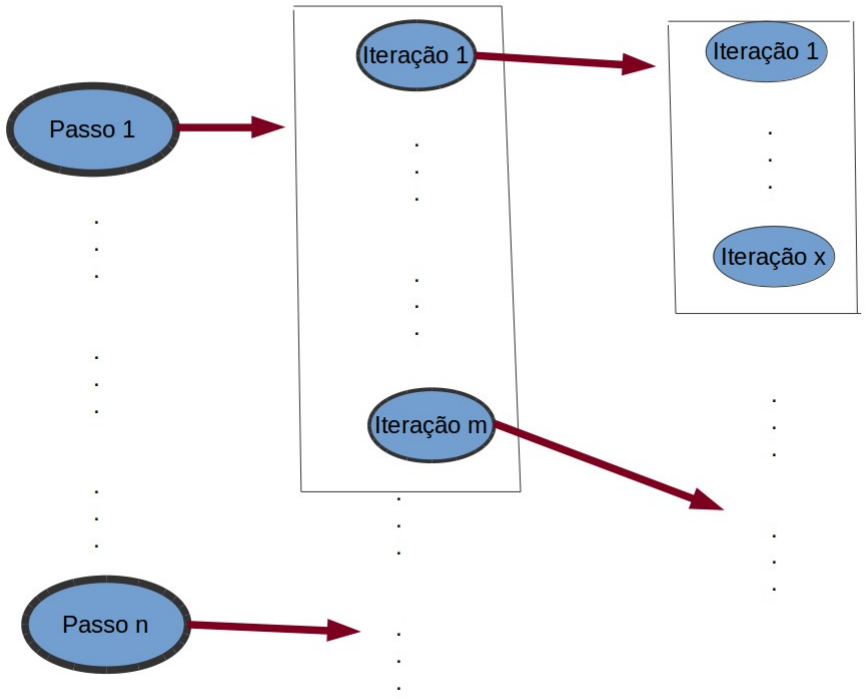
O programa realiza algumas etapas custosas no começo, mas a parte que mais demanda tempo é a parte iterativa do programa, que é o onde se tem várias iterações para convergência em um determinado passo. Em cada iteração é chamada uma função bastante custosa, que também realiza uma parte iterativa de grande porte. A Figura 3 apresenta um esquema da relação entre passos e iterações do programa. A parte iterativa do programa se dá com um certo número de passos (n passos), esse número é limitado ao máximo de 3000 passos. Cada passo executa um número máximo de iterações (m iterações), que o usuário especifica na entrada (geralmente entre 30 e 50 iterações para convergir). Em cada uma destas iterações é chamada uma função, que realiza um número de iterações (x iterações), que depende do tamanho do problema, este número é dado pelo número de elementos, que em geral é um número grande.

O programa também faz bastante acessos a arquivos (leitura e escrita) durante a execução, o que impede a execução de dois processos simultâneos, mesmo sobre malhas diferentes. Para a execução de dois processos do RAFEM necessita-se que cada um dos programas que esteja rodando se localize em um diretório diferente.

2.4 Considerações

Neste capítulo, foram abordados os conceitos que envolvem o MEF, que é o método utilizado para simulação no RAFEM, bem como também alguns conceitos a respeito de

Figura 3 – Esquema de passos e iterações do RAFEM



paralelismo e um detalhamento maior do OpenMP que é a API que será utilizada neste trabalho para a paralelização. Foi introduzido um pouco sobre o programa RAFEM, que entradas ele recebe, como é estruturada a parte iterativa que realiza a computação pesada do problema e como são os elementos tratados por ele. O RAFEM será melhor detalhado no Capítulo 4.

3 Trabalhos relacionados

Como não existe nenhum trabalho que esteja relacionado com a paralelização do RAFEM ainda, a seção de trabalhos relacionados foi dividida em duas linhas de pesquisa, uma delas voltada para a simulação de ARF através do MEF e, a outra voltada para processamento paralelo sobre o MEF em diferentes contextos.

3.1 Simulação de ARF

No trabalho de Jiang et al. (2010) tem-se o desenvolvimento da ferramenta RAFEM e comparação dos resultados obtidos de simulações a partir desta com os resultados obtidos por experimento. O trabalho mostra que os resultados são coerentes, tornando o uso do RAFEM plausível. A ferramenta desenvolvida baseia-se no MEF, tendo-se a modelagem através de elementos tetraedrais. De maneira similar, no trabalho de Haemmerich e Webster (2005) também utiliza-se a modelagem através de elementos tetraédricos. Nesse trabalho o propósito é o controle automático de modelos de elementos finitos para o processo de ARF com temperatura controlada, onde desenvolveu-se um programa de controle, escrito em C++. O processo ocorre de maneira que se espera manter a temperatura no fígado constante, desta maneira os outros parâmetros são ajustados para que isto seja atendido.

No trabalho de Lukeš et al. (2014) tem-se a utilização do MEF para a simulação de perfusão sanguínea no fígado. O objetivo principal do trabalho é construir um modelo de uma simulação a partir de leituras de imagens geradas a partir de processos de tomografia computadorizada e ressonância magnética.

Jiang et al. (2007) apresenta um modelo analítico da distribuição de temperatura no processo de ARF considerando-se um ponto de corrente elétrica. Nesse trabalho apresenta-se uma solução analítica e é provada a sua corretude.

No trabalho de Santos et al. (2009) tem-se o desenvolvimento de um modelo probabilístico para o MEF, usado para simulação de ARF, almejando-se assim o controle de erros presente no modelo determinístico usado, para isso então se faz uso da transformada da incerteza, proposta por Julier e Uhlmann (2004). No trabalho é mostrado como o uso de um modelo determinístico permite estimar a variação presente na zona de coagulação devido a incertezas nos parâmetros do tecido.

Tungjitkusolmun et al. (2002) propõe a análise do processo de ARF utilizando MEF a partir de um modelo tridimensional e também a partir de um modelo bidimensional mais simples. Nesse trabalho é estudada a interferência devido a presença de vasos

sanguíneos no modelo, onde se conclui haver o aumento de temperatura no tecido hepático localizado entre o eletrodo e o vaso sanguíneo devido ao efeito Joule. Outro fator concluído nesse trabalho é a dissipação de calor térmico no tecido ao redor do vaso sanguíneo.

3.2 Paralelismo em método de elementos finitos

O trabalho de Slobodan et al. (2012) é aplicado ao contexto de análise eletromagnética. Tem-se dois passos principais, o preenchimento da matriz de impedância e a solução da matriz gerada, onde o tempo total é dado pela soma dos tempos em cada um destes passos. Segundo Liu, Chua e Y. (2009), dependendo do tamanho da matriz, a etapa de preenchimento desta compreende uma parte significante do tempo total. Slobodan et al. (2012) menciona a integração numérica como sendo o que toma maior parte do tempo no preenchimento da matriz. Partindo disso esse trabalho apresenta a paralelização na etapa de integração utilizando OpenMP. Segundo (SLOBODAN et al., 2012), como maior parte das operações que apresentam considerável consumo de tempo possuem caráter local, isto permite que sejam efetuadas para vários elementos ao mesmo tempo, a nível global.

Pantalé (2005) tem seu trabalho aplicado sobre o contexto de simulação de impactos, e propõe o uso de OpenMP para melhoria de desempenho. Em seu trabalho ele explora bastante a questão de computação dos elementos do modelo em paralelo, tentando assim diversas abordagens, sendo a mais a de balanceamento dinâmico de carga entre as *threads*.

Sergey (2013) associa o processo de construção da matriz global de rigidez e a subsequente solução do sistema linear algébrico como sendo a maior demanda de recurso computacional, desta forma em seu trabalho então considera-se a adoção de processamento paralelo nestas duas etapas, através de OpenMP. No caso da montagem da matriz global de rigidez, considera-se o uso de vários núcleos de processamento, sendo que cada *thread* monta uma matriz local, para posteriormente as matrizes serem combinadas em uma matriz global. Nesse trabalho é também focada a etapa de multiplicação entre matrizes para paralelização. Jimack e Touheed (2000) também referem as etapas de montagem da matriz global de rigidez e a solução do sistema de equações como as etapas mais custosas, e propõem o uso de MPI para paralelizar estas duas etapas. Assume-se um domínio que é discretizado por meio de elementos triangulares. O conjunto destes elementos então forma a malha, esta malha por sua vez é particionada em várias sub-malhas, de maneira que não haja sobreposição entre estas. Almeja-se que cada processo tenha uma quantia igual de elementos, buscando uma distribuição homogênea. Cada processo será responsável por montar uma parte da matriz global de rigidez. Após a conclusão da montagem da matriz, esta é armazenada de maneira distribuída, e a solução se dá desta mesma forma, de maneira que cada processo resolve uma parcela. Uma desvantagem mencionada por

Jimack e Touheed (2000) é o custo de desempenho envolvendo trocas de mensagens para comunicação entre processos.

No trabalho de Mahinthakumar e Saied (2002) tem-se uma abordagem híbrida, utilizando-se MPI e OpenMP combinados, onde utiliza-se MPI para paralelismo com memória distribuída, e OpenMP para paralelismo com memória compartilhada. O método com o MPI é feito de maneira similar ao utilizado no trabalho de Jimack e Touheed (2000), onde tem-se a divisão do domínio do problema em vários sub-domínios. Diferente do trabalho de Jimack e Touheed (2000), nesse trabalho utiliza-se a sobreposição de uma camada de elementos de contorno das sub-malhas (sub-domínios) para evitar a troca de mensagens quando se estiver operando sobre a região de contorno. Assim como Sergey (2013) e Jimack e Touheed (2000), Mahinthakumar e Saied (2002) também menciona as etapas de montagem da matriz global de rigidez e multiplicação de matrizes como as maiores fontes de demanda de tempo. No modelo proposto nesse trabalho, tem-se a divisão do problema em sub-domínios, onde cada um destes sub-domínios pertence a um processo MPI. Caracterizando a abordagem híbrida então, tem-se vários processos MPI distribuídos, e em nível local é feita a paralelização através de OpenMP. A aplicação de diretivas é feita sobre os loops que são caracterizados por consumir maior parte do tempo, no caso, sobre a montagem da matriz global de rigidez e sobre a multiplicação de matrizes. No processo de montagem da matriz pode haver a ocorrência de conflitos entre threads devido ao possível compartilhamento de um nó entre dois elementos pertencentes a threads diferentes, para isto pode-se utilizar o esquema de ordenamento intercalado, o que evita o uso de regiões críticas dentro do laço.

3.3 Considerações

Neste capítulo, buscamos relacionar de alguma forma o nosso trabalho com outros trabalhos. Levamos em conta o contexto de simulação de ARF, que é um tema presente neste trabalho, e também a questão de processamento paralelo no MEF, de maneira que pudéssemos tomar como referência as etapas custosas mencionadas e possíveis soluções. O RAFEM trabalha na solução do modelo de uma forma diferente, através de uma função, chamada `VFrontalMethod`, onde se aborda a solução por meio da matriz local gerada para cada elemento. De tal forma, a relação existente entre a paralelização do MEF e a paralelização do RAFEM não é direta, pois este possui características particulares em relação aos outros trabalhos, que são aplicados em outros contextos.

4 Metodologia

Neste capítulo será descrito o processo de paralelização do RAFEM realizado em nosso trabalho. Primeiramente, foram realizadas algumas modificações em relação ao makefile e aos arquivos de código, mantendo-se apenas o que era efetivamente utilizado. O outro passo foi inspecionar a execução do programa de forma a diagnosticar etapas demoradas no algoritmo. Feito isso então, através de análise do código, teve-se a identificação da função que estava sendo custosa na computação. Sobre esta função então dirigimos a análise do código, para posteriormente então efetuar a paralelização. Por fim então, serão apresentados os resultados de desempenho obtidos com o nosso trabalho, bem como direções futuras para a melhoria dos resultados.

4.1 Organização

O RAFEM é dividido em vários arquivos de código, e entre estes arquivos de código existem dependências. No entanto, um fator que dificulta bastante para a modificação dos códigos é a redundância presente nestes, ao invés de ter-se um arquivo com os protótipos de funções, as funções são todas prototipadas no começo de cada arquivo `.cpp`, ou seja, para se alterar os parâmetros de uma função tem-se que substituir todas as ocorrências do protótipo desta. Outro fator que polui bastante o código é a presença de muitas variáveis não utilizadas, e muitas utilizadas de maneira irrelevante, o que já não é detectado pelo compilador.

Para melhor trabalhar com o código então, colocamos o RAFEM em um novo diretório contendo apenas os arquivos de código relevantes ao makefile. Algumas modificações tiveram de ser feitas no makefile para compilar o programa no Linux. Feito isso então partimos para a etapa de execução do programa.

4.2 Execuções

Foram realizadas as execuções afim de se identificar o fluxo de execução do programa, utilizando as impressões geradas pelo programa como auxílio. Desta maneira, quando ocorria uma impressão do programa e demorava para se prosseguir, identificamos que existia uma etapa custosa. Baseando-se nisto obtivemos os números das linhas no código onde estava um foco de demora.

Com base no diagnóstico realizado chegamos até a função `VFrontalMethod`, que é a função que é chamada a cada iteração da computação, sendo a parte mais custosa do

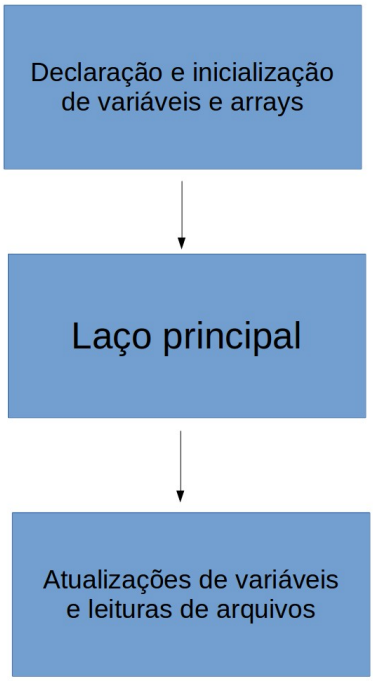
laço. Esta função acaba contribuindo muito para o desempenho do programa pois ela se localiza na parte iterativa do programa. Com base nesta constatação passamos a estudar a paralelização da da função `VFrontalMethod`.

4.3 A função VFrontalMethod

Como mencionado anteriormente, a computação ocorre em um determinado número de passos, e em cada passo tem-se várias iterações. Para cada uma das iterações presentes em um passo, a função `VFrontalMethod` é chamada. Existem outras funções consideravelmente custosas, mas que acabam não sendo tão relevantes ao desempenho pois se situam apenas na parte inicial da computação, ou seja, não são chamadas com tanta frequência quanto o `VFrontalMethod`. De maneira geral, no contexto do RAFEM esta é a função responsável pela computação de todos os elementos da malha, que ocorre a cada uma das iterações presentes em cada passo. Nesta seção será apresentada a estrutura e também o escopo de variáveis da função.

4.3.1 Estrutura da função

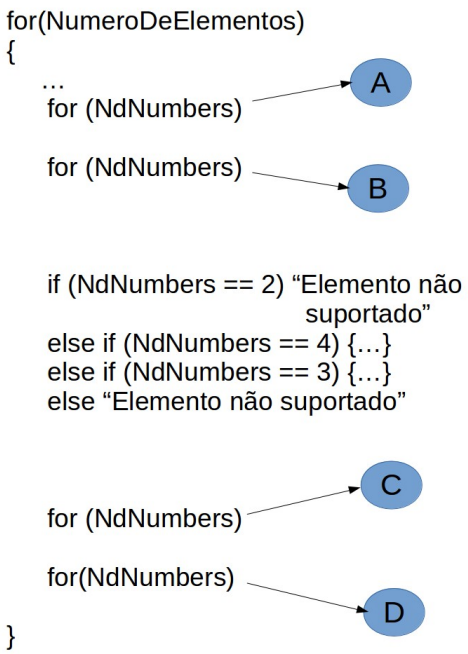
Figura 4 – Esboço da função `VFrontalMethod`



A função `VFrontalMethod` tem a seguinte estrutura, como mostrado na Figura 4. Em uma primeira etapa tem-se a declaração e inicialização de variáveis e *arrays*, depois disso então é atingido o laço principal, onde ocorre a computação de cada elemento do

modelo, e por fim, tem-se a atualização de variáveis e algumas leituras de arquivo. A etapa mais custosa da função acaba sendo o laço principal, e é sobre o qual dedicamos nosso foco.

Figura 5 – Esboço do laço principal da função `VFrontalMethod`



O laço principal da função `VFrontalMethod` segue a estrutura representada na Figura 5. O laço `for` mais externo é feito sobre a variável `NumeroDeElementos`, que em geral é um número grande. Dentro deste laço tem-se outros laços menores, feitos sobre a variável `NdNumbers`, que indica o número de nós de um elemento, esse número varia entre 3 ou 4. Os primeiros dois laços, A e B, são para atribuições e inicialização de variáveis, em geral não são custosos. Para o número de nós sendo 3 ou 4 tem-se uma sequencia de operações (decorrente do `if`), que envolvem laços em geral pequenos (todos sobre `NdNumbers`), o que não chega a ser uma etapa muito custosa também. Os dois últimos laços, C e D, chegam a ser mais custosos, pois envolvem bastante operações, mesmo sendo com poucas iterações. O laço C tem um porte maior, e envolve sucessivas chamadas a função getFreeLine, que realiza um busca linear sobre um arranjo de tamanho considerável, dado pela variável `maxFrontWidth` (em torno de 1000 posições). O laço D é o que apresenta maior custo computacional, pois dentro dele existem mais laços, de tamanho considerável. Desta maneira optamos então por detalhar os laços C e D, que são os mais custosos.

Na Figura 6 é mostrado um esboço do laço C, interno ao laço principal da função VFrontalMethod. Neste laço existem outros laços internos, todos com índice pequeno,

Figura 6 – Esboço do laço C, interno ao laço principal da função `VFrontalMethod`

```
for(NdNumbers)
{
  ...
  for(NODFN)
  {
    ...getFreeLine

    for(NdNumbers)
    {
      ...
      for(NODFN)
      {
        ...getFreeLine
      }

    }

  }

}
```

tendo em vista que a variável `NdNumbers` varia entre 3 e 4, e que `NODFN` é uma constante, definida como 2. A parte mais custosa que ocorre no loop C são as chamadas à função `getFreeLine`, que trata de uma busca sequencial por uma posição vazia (posição de valor 0) para em seguida atribuir um valor para a posição.

Figura 7 – Esboço do laço D, interno ao laço principal da função `VFrontalMethod`

```
for (NdNumbers)
{

  for (NODFN)
  {

    ...

    for (maxFrontWidth)

    ...

    for (maxFrontWidth)

    ...

    for (maxFrontWidth)

  }

}
```

A Figura 7 mostra a estrutura do laço D. O laço mais externo se dá sobre a variável `NdNumbers`, que em geral é um número pequeno. Internamente há um laço sobre

a constante `NODFN`, que é 2. Dentro deste laço então existem 3 laços sobre a variável `maxFrontWidth`, que é um número maior (na ordem de milhar).

4.3.2 Escopo de variáveis

A função `VFrontalMethod` conta com 40 parâmetros e 144 variáveis. Muitos dos parâmetros da função são *arrays* dos quais somente são feitas leituras, bem como também índices de laços, flags e valores de entrada. Do total de parâmetros, 3 deles são alterados por referência no decorrer da função. O número de variáveis citado já exclui variáveis não utilizadas, que foram diagnosticadas e removidas do código.

4.4 Paralelização

Nesta seção será descrita a forma como foi realizada a paralelização da função `VFrontalMethod`, iniciando-se pela etapa de análise e classificação das variáveis da função. Em seguida teve-se a análise de etapas custosas para paralelização, e por fim a aplicação de processamento paralelo.

4.4.1 Análise e classificação das variáveis

O primeiro passo foi a identificação e eliminação de variáveis não utilizadas, para isso utilizamos da flag `-Wunused` do compilador g++. Em alguns casos foi necessária a análise da variável, pois existem várias ocorrências de variáveis declaradas, atribuídas, mas não utilizadas em nenhum caso.

Posteriormente realizamos a classificação das variáveis em relação a um dado elemento do modelo, podendo ser de caráter local ou global. A variável ter caráter local significa que cada elemento possui ela de maneira privada, ou seja, é uma característica de cada elemento. O caráter global de uma variável ocorre quando ela é acumulada com o decorrer da computação dos elementos, é o caso também de alguns *arrays* que são preenchidos ao decorrer do processo. Esta classificação teve que ser realizada manualmente, onde buscamos a ocorrência de cada variável dentro do laço principal. Os acumuladores com ocorrência dentro do laço principal em geral já são globais, pois são iterados entre vários elementos diferentes. Existem casos de *arrays* privados de cada elemento e também o caso de globais, de maneira geral, o caráter global é constatado pelo preenchimento parcial do *array*, dado pela iteração em cada elemento. As variáveis globais de maneira geral ficaram sendo *arrays*, acumuladores, flags e delimitadores de iterações. As variáveis locais ficaram sendo alguns *arrays* e variáveis que são atribuídas a cada iteração, ou seja, pra cada elemento terão um valor particular, dado por aquele elemento. Retomando a Figura 5, a atribuição de variáveis locais se dá nos laços A e B, e também no controle de fluxo `if`, os laços C e D então realizam a computação a nível global.

Por fim, realizada a classificação das variáveis, as declarações das variáveis locais foram trazidas para dentro do laço principal, de maneira que ficaram claras as variáveis locais a cada elemento.

4.4.2 Análise de etapas custosas

Tendo em vista a estrutura da função `VFrontalMethod`, mostrada na Figura 4, sabemos que o laço principal é a etapa mais custosa da função. A abordagem utilizada foi a análise dos laços internos ao laço principal, onde buscamos compreender o quanto o custo do laço cresce de acordo com o tamanho do problema, concluindo assim se é vantajoso paralelizar.

O laço C, mostrado na Figura 6 é significantemente mais custoso devido as chamadas da função `getFreeLine`, que é uma busca sequencial no *array* `freeLines`. O *array* `freeLines` tem tamanho dado pela variável `maxFrontWidth`, que é um número alto, e que tende a aumentar com o tamanho do problema. Uma grande limitação que surge neste laço é que `freeLines` é de caráter global, então a chamada à função `getFreeLine` e a posterior atualização da posição livre devem ocorrer de maneira exclusiva. Com isso, a presença de regiões críticas tende a serializar estes trechos de código, que de maneira geral são a parte mais custosa do laço C. Outro fator que dificulta o ganho de desempenho em escala neste laço é o fato de ter poucas iterações (3 ou 4), então se tem praticamente uma limitação no número de threads que podem atuar.

O laço D, apresentado na Figura 7 em geral é a parte mais custosa de todo o laço principal, pois tem bastante operações, caracterizando um porte grande. Além do grande porte do laço, este também apresenta laços internos sobre a variável `maxFrontWidth`, que é um número grande, e que cresce com o tamanho do problema. A hipótese de paralelizar o laço mais externo (laço D) acaba enfrentando o mesmo problema que foi mencionado anteriormente para o laço C, que é o fato de que o laço tem poucas iterações e o ganho em escala não pode ocorrer, o número de threads acaba ficando limitado.

Com base nesta análise então tomamos a decisão do trecho a ser paralelizado.

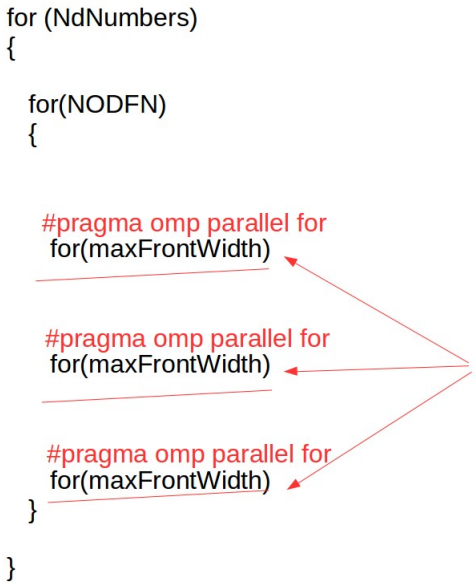
4.4.3 Aplicação de processamento paralelo

Considerando que ao paralelizarmos os laços C e D encontraria-se o problema de ganho em escala, então optamos por abordar as etapas internas destes laços.

O laço C (Figura 6), como mencionado, tem o problema de serialização devido a necessidade de regiões críticas envolvendo a função `getFreeLine`, além do fato de ter laços internos pequenos, o que não possibilita escalabilidade e limita bastante o desempenho. Devido a isso, esta etapa não foi focada.

O laço D (Figura 7), apresenta laços internos de tamanho grande (sobre a variável `maxFrontWidth`). A paralelização do laço D enfrentaria o problema de ganho em escala. Já a paralelização dos laços internos ao laço D tende a enfrentar o problema de *overhead*, gerado pelo *fork-join* feito a cada iteração, mas possibilita o ganho em escala.

Figura 8 – Esboço do laço D, com as etapas que foram paralelizadas



Dedicamos o foco na paralelização dos laços internos ao laço D. Na Figura 8 observam-se destacadas em vermelho as etapas que optamos por paralelizar, no caso, os laços sobre `maxFrontWidth`. Cada laço é executado por vez, em paralelo.

Adicionalmente, paralelizamos outras etapas como teste para verificar o impacto no desempenho. Com a paralelização das etapas anteriores ao laço D não se obteve um desempenho muito vantojoso, em alguns casos o programa paralelo estava obtendo desempenho inferior ao sequencial. Frente a isso mantivemos a paralelização apenas sobre o laço D.

A seguir serão mostrados casos de teste realizados com a versão sequencial e com a versão paralelizada do RAFEM. Também será apontada uma direção futura para se melhorar a paralelização do RAFEM.

4.5 Testes e resultados obtidos

Nesta seção serão apresentados os casos de teste, bem como os resultados obtidos a partir destes.

A máquina de testes utilizada foi uma workstation Dell Precision T7600. A arquitetura possui 128 GB de memória e dois processadores Intel Xeon E5/Core i7, de 2.00

GHz, totalizando 32 núcleos de processamento, onde 16 são físicos e outros 16 são lógicos.

Abordamos dois casos de teste, um com uma malha de tamanho pequeno e outro com uma de tamanho grande. A seguir apresentamos cada um dos casos de teste. Para os testes utilizamos 16 threads, que é o número de núcleos físicos que a máquina de testes possui.

4.5.1 Primeiro caso de teste: malha grossa

Neste caso foi utilizada uma malha com 18363 elementos e 3548 nós. Como entrada para o programa definimos o número máximo de iterações por passo como 30 e a duração do procedimento como 300 segundos. Realizamos 5 execuções para cada versão do programa para estimar uma média de duração para o sequencial e para o paralelo.

Na tabela 1 são apresentados os resultados de tempo de computação do problema, dado em minutos. Comparando-se em relação as médias obtidas, no sequencial temos um tempo de execução de aproximadamente 1 hora e 21 minutos, e no paralelo 23 minutos isto nos dá um *speedup* de cerca de 3.49 vezes.

Tabela 1 – Tempo total de computação (min) com malha grossa

Execução	Tempo de computação	
	Sequencial	Paralelo
#1	81,73	23,20
#2	81,96	23,55
#3	82,00	23,18
#4	81,35	23,70
#5	81,10	23,23
média	81,63	23,37

Tabela 2 – Tempo médio da função `VFrontalMethod` (s) com malha grossa

Execução	Tempo médio de VFrontal	
	sequencial	Paralelo
#1	17,60	4,97
#2	17,65	5,04
#3	17,66	4,96
#4	17,52	5,08
#5	17,46	4,98
média	17,57	5

A tabela 2 apresenta o tempo médio, em segundos, gasto com a chamada da função `VFrontalMethod` ao decorrer da computação do problema. Neste caso, relacionando-se as médias, cada chamada à função `VFrontalMethod` no sequencial dura cerca de 17 segundos e meio, já no paralelo dura cerca de 5 segundos, com isto temos um *speedup* de aproximadamente 3.51 vezes.

4.5.2 Segundo caso de teste: malha fina

Para este caso de teste utilizamos uma malha com 44811 elementos e 8364 nós. Neste caso, o número máximo de iterações teve que ser definido em 50, pois com um máximo de 30 iterações não estava convergindo. A duração do procedimento foi definida para 300 segundos. Foram realizadas 3 execuções para o sequencial e para o paralelo.

Na tabela 3 é apresentado o tempo de computação (em minutos) no sequencial e no paralelo com uma malha fina. Em média, o tempo de execução no sequencial foi a cerca de 20 horas e 12 minutos, contra 4 horas e 20 minutos aproximadamente na versão paralela. Com isto o *speedup* obtido neste caso foi de cerca de 4.65 vezes.

Tabela 3 – Tempo total de computação (min) com malha fina

Execução	Tempo de computação	
	Sequencial	Paralelo
#1	1212,72	257,51
#2	1212,13	262,05
#3	1212,9	262,93
média	1212,58	260,83

Tabela 4 – Tempo médio da função `VFrontalMethod` (s) com malha fina

Execução	Tempo médio de VFrontal	
	Sequencial	Paralelo
#1	74,65	15,81
#2	74,62	16,09
#3	74,67	16,15
média	74,64	16,01

A tabela 4 apresenta o tempo médio (em segundos) gasto com cada chamada à função `VFrontalMethod`. Na versão sequencial, tem-se cerca de 1 minuto e 14 segundos para cada chamada da função `VFrontalMethod`, contra cerca de 16 segundos na versão paralela. Isto nos dá um *speedup* de aproximadamente 4.66 vezes.

4.5.3 Considerações

Levando em conta que o segmento paralelizado é iterado dependendo da variável `maxFrontWidth`, e que o valor desta variável cresce com o tamanho do problema, a execução deste segmento sequencialmente se torna mais demorada, e a execução em paralelo passa a ter um ganho um pouco maior. Considerando que se em uma única iteração do laço principal da função `VFrontalMethod` tivermos um ganho, mesmo pequeno, ele tende a se multiplicar com o número de iterações do laço, que é dado pelo número de elementos. Logo, para uma quantia de elementos maior no modelo o ganho tende a se amplificar. Com isso a função `VFrontalMethod` fica mais rápida. Para a computação do

problema temos n passos, e cada passo tem um número máximo de iterações, onde em cada uma destas iterações temos uma chamada ao `VFrontalMethod`. A partir disso podemos considerar que um pequeno ganho obtido em uma única iteração do laço principal do `VFrontalMethod` já se torna bem significante na computação do problema. Portanto, para problemas maiores o ganho de desempenho tende a ser maior, devido ao crescimento de `maxFrontWidth`. Este fenômeno pode ser visualizado nos gráficos das Figuras 9 e 10, onde a discrepância é nitidamente maior nos casos de teste com a malha fina.

Figura 9 – Tempo total de computação do problema (min)

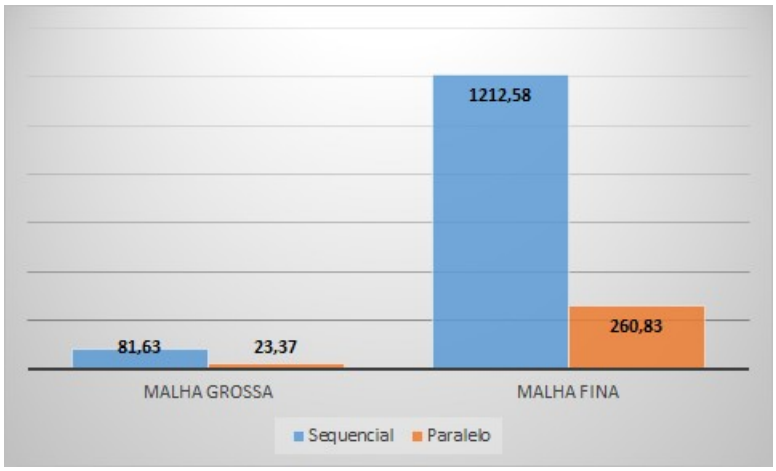
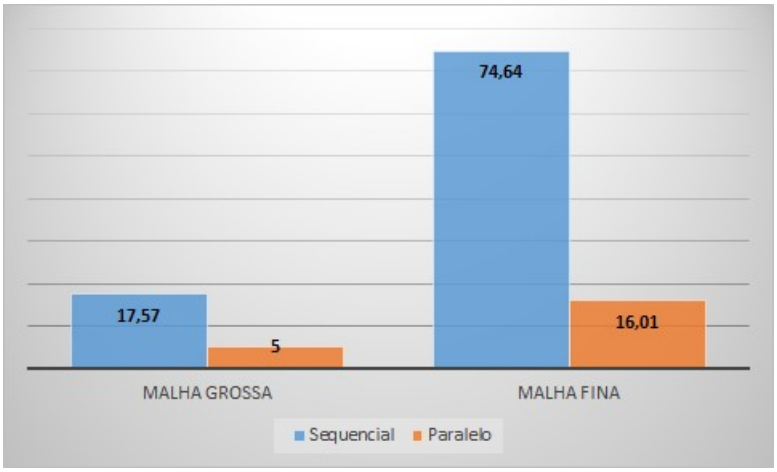


Figura 10 – Tempo de cada chamada a `VFrontalMethod` (s)



4.6 Direções futuras

A etapa que identificamos como foco para paralelização foi o laço principal da função `VFrontalMethod`. A abordagem que utilizamos foi a aplicação de processamento paralelo interno ao laço principal. Uma solução que pode atuar melhor, e que é mais escalável é a paralelização do laço principal como um todo.

Neste trabalho chegamos a abordar a paralelização do laço principal, no entanto é uma tarefa não concluída ainda. O problema de paralelização neste caso se torna muito maior devido ao grande número de variáveis envolvidas, as dependências e até mesmo o ordenamento de iterações em alguns trechos. A análise se torna muito maior, considerando que o laço principal tem cerca de 600 linhas, desta forma esta abordagem demanda mais tempo para ser implementada.

5 Conclusão

O processo de ARF constitui uma forma de tratamento do hepatocarcinoma de uma maneira minimamente invasiva. O RAFEM é proposto com o intuito de simular o procedimento de ARF. Para cada paciente que realiza o procedimento, é necessária uma simulação, desta forma o desempenho do programa se apresenta como um aspecto muito importante. Neste trabalho foi aplicado processamento paralelo no RAFEM através da API OpenMP. Os testes mostraram que o ganho foi satisfatório em questão de tempo de execução, e que a tendência é que a solução apresentada fucione bem para modelos maiores. Por fim apresentamos uma solução melhor para o problema de paralelização do RAFEM, mas mostrando-se como uma direção futura.

Referências

GOMES, M. A. et al. Carcinoma hepatocelular: epidemiologia, biologia, diagnóstico e terapias. *Revista da Associação Médica Brasileira*, Elsevier, v. 59, n. 5, p. 514–524, 2013. Citado na página 18.

HAEMMERICH, D.; WEBSTER, J. G. Automatic control of finite element models for temperature-controlled radiofrequency ablation. *Biomedical engineering online*, 2005. Citado 2 vezes nas páginas 19 e 27.

JIANG, Y. et al. Analytical solution of temperature distributions in radiofrequency ablation due to a point source of electrical current. *6th Brazilian Conference on Dynamics, Control and their Applications*, 2007. Citado na página 27.

JIANG, Y. et al. Formulation of 3d finite elements for hepatic radiofrequency ablation. *International Journal of Modelling, Identification and Control*, Inderscience Publishers, p. 225–235, 2010. Citado 2 vezes nas páginas 20 e 27.

JIMACK, P.; TOUHEED, N. Developing parallel finite element software using mpi. *High Performance Computing for Computational Mechanics*, Citeseer, p. 15–38, 2000. Citado 2 vezes nas páginas 28 e 29.

JULIER, S. J.; UHLMANN, J. K. Unscented filtering and nonlinear estimation. *Proceedings of the IEEE*, IEEE, v. 92, n. 3, p. 401–422, 2004. Citado na página 27.

LIU, Z. H.; CHUA, E. K.; Y., S. K. Accurate and efficient evaluation of method of moments matrix based on a generalysed analytical approach. *Progress In Electromagnetics Research*, 2009. Citado na página 28.

LUKEŠ, V. et al. Numerical simulation of liver perfusion: from ct scans to fe model. *arXiv preprint arXiv:1412.6412*, 2014. Citado na página 27.

MAHINTHAKUMAR, G.; SAIED, F. A hybrid mpi-openmp implementation of an implicit finite-element code on parallel architectures. *The International Journal of High Performance Computing Applications*, 2002. Citado na página 29.

MINAYO, M. C. d. S. Saúde-doença: uma concepção popular da etiologia. *Cadernos de Saúde Pública*, SciELO Public Health, v. 4, n. 4, p. 363–381, 1988. Citado na página 17.

MOAVENI, S. *Finite element analysis: theory and application with ANSYS*. [S.l.]: Pearson Education India, 2003. Citado na página 21.

MPI. 2016. <https://www.open-mpi.org/>. Acessado em: 2016-06-13. Citado na página 22.

OPENMP. 2016. <http://openmp.org/wp/>. Acessado em: 2016-06-13. Citado na página 22.

PANTALÉ, O. Parallelization of an object-oriented fem dynamics code: influence of the strategies on the speedup. *Advances in Engineering Software*, Elsevier, n. 6, p. 361–373, 2005. Citado na página 28.

SANTOS, I. D. et al. Probabilistic finite element analysis of radiofrequency liver ablation using the unscented transform. *Physics in medicine and biology*, IOP Publishing, v. 54, n. 3, p. 627, 2009. Citado na página 27.

SERGEY, C. Parallel computing technologies in the finite element method. *Third International Conference "High Performance Computing"*, 2013. Citado 3 vezes nas páginas 21, 28 e 29.

SLOBODAN, V. S. et al. Acceleration of higher order matrix filling by openmp paralellization of volume integrations. *20th Telecommunications forum TELFOR*, 2012. Citado na página 28.

SOUZA, V. C. Tratamento paliativo do chc - resultados da quimioembolização. proposta de indicação. *GED gastroenterol*, 2011. Citado na página 18.

TUNGJITKUSOLMUN, S. et al. Three-dimensional finite-element analyses for radio-frequency hepatic tumor ablation. *Biomedical Engineering, IEEE Transactions on*, IEEE, v. 49, n. 1, p. 3–9, 2002. Citado na página 27.