

# MAC0499 – Trabalho de Formatura Supervisionado.

## Proposta de Trabalho

Giuliano Augusto Faulin Belinassi - 8517272

## 1 Introdução

É de extrema importância para a Engenharia Civil, se não para a Sociedade como um todo, o estudo de vibrações em estruturas para evitar possíveis catástrofes providas de terremotos, ou talvez incômodos de outras fontes, tais como máquinas operatrizes e linhas ferroviárias. Como na grande maioria dos casos as vibrações chegam às construções através do solo, a principal parte da Engenharia que trata problemas desta espécie é a Dinâmica dos Solos.

Com a evolução dos computadores, é de se esperar que heurísticas e algoritmos fossem projetados para simular o efeitos de tais vibrações em estruturas. Dentre estes, deve-se ressaltar o Método dos Elementos de Contorno (MEC), que utiliza pontos na superfície do volume a ser analisado, e o Método dos Elementos Finitos (MEF), que insere pontos internos ao volume do objeto de estudo. Existem vantagens em escolher o MEC ao MEF, entre elas [1]:

1. Menor tempo para preparação dos dados.
2. Maior precisão dos pontos de estresse.
3. Menor uso de recursos computacionais.
4. Menos informações desnecessárias.

e isto motiva o estudo de tal método para solução de problemas deste tipo. Embora toda a análise teórica do BEM seja interessante no ponto de vista da Engenharia, este trabalho tem enfoque na implementação do algoritmo de forma a explorar recursos computacionais providenciados por processadores de vários núcleos e Unidades de Processamento Gráfico do Propósito Geral, portanto a parte computacional do problema.

## 2 Objetivos

Este trabalho tem como objetivo implementar o MEC usando Unidades de Processamento Gráfico de Propósito Geral (GPGPU) partindo da implementação fornecida por [2]. A necessidade de paralelização se dá devido a possibilidade de analisar estruturas com superfícies maiores e com mais pontos, aumentando assim a precisão dos resultados obtidos.

Tabela 1: *Proffiling* do código sequencial

Subrotina	Tempo (%)
solfund	34.11
nonsingd	18.24
nonsinge	14.71
solfone	9.13
gauleg	8.23

### 3 Estado do Trabalho

Este trabalho teve início em Setembro de 2016, com a primeira etapa tendo como objetivo estudar a linguagem de programação usada na implementação entregue por [2] (no caso, Fortran). Algumas dificuldades foram encontradas aí, pois o código foi escrito em uma versão obsoleta desta, além de não compilar com o GFortran [3]. Adaptações e manutenções foram necessárias.

Em seguida, algumas técnicas de paralelização de algoritmos foi estudada para que fosse possível prosseguir com o trabalho, além de testes automatizados.

Um *proffiling* foi efetuado no código, logo em seguida, para que os gargalos tornassem evidentes, conforme a tabela 1.

Analisando mais rebuscadamente o código, descobriu-se que paralelizar o procedimento *Ghmatecd*, responsável por montar as matrizes  $H$  e  $G$  conforme [2], implicaria em chamadas paralelas das subrotinas *Nonsingd* e *Solfund*, sendo assim passou-se a priorizar a paralelização dessa subrotina.

Como alterar um código pode comprometer funcionalidades que deste dependem, a criação de testes automatizados foi fundamental para assegurar que o resultado obtido condiz com o original. Embora existam frameworks para a criação de testes em Fortran, por este projeto tratar com código legado, preferiu-se criar as próprias funções independentes e programas externos que verificam se o resultado é, de fato, o esperado. Portanto, para verificar os erros nas matrizes geradas, preferiu-se utilizar ora a norma infinita, ora a norma 1 da soma da diferença das matrizes, conforme o Teorema 2.1.29 de [4].

Concluindo os testes de aceitação, a etapa de paralelização usando OpenMP teve início, em conjunto com algumas otimizações sequenciais. Utilizando um problema de dimensões conforme a tabela 2, o tempo gasto na implementação original para esta entrada era superior a 4m17s. Após paralelizar todo o programa e realizar algumas otimizações sequências este tempo caiu em função da quantidade de processadores alocados, conforme a tabela 3. É interessante notar o *speedup* linear.

Em seguida, com todo o programa paralelizado na CPU, partiu-se para experimentos com GPGPU na tentativa de obter resultados mais interessantes que os obtidos. Como a plataforma escolhida foi CUDA e como o compilador de CUDA para Fortran é pago, optou-se por construir uma interface CUDA C  $\leftrightarrow$  Fortran.

Proseguindo com a paralelização em GPGPU, uma implementação de *Ghmatecd* foi codificada em CUDA C, porém os resultados não foram satisfatórios:

Tabela 2: Entrada ESOLO2160E\_-5+5

Número de Elementos da Malha	2160
Número de Elementos de Contorno	900
Número de Pontos Extremos dos Elementos	2162
Número de Pontos Internos	10
Número de Pontos de Gauss	8
Número de Frequências	1
Módulo de Cisalhamento	1.00
Coefficiente de Poisson	0.30
Coefficiente de Amortecimento	0.00
Densidade de Massa	1.00

Tabela 3: Tempo gasto em **Pé Grande**, com flags -Ofast -march=native -fno-unroll-loops

Processadores	Tempo
1	1m20s
2	42s
4	22s

## 4 Apêndice

Informações técnicas dos computadores de teste.

Tabela 4: Pé Grande

Processador	Intel(R) Core(R) i7 CPU 920 @ 2.6GHz
Memória	8Gb
GPU	NVIDIA(R) GeForce(R) GTX 470

## Referências

- [1] Heinz Antes. A short course on boundary elements methods. *Technische Universität Braunschweig*, 2010.
- [2] Ronaldo Carrion. *Uma Implementação do Método dos Elementos de Contorno para Problemas Viscoelastodinâmicos Estacionários Tridimensionais de Domínios Abertos e Fechados*. PhD thesis, Universidade Estadual de Campinas, 2002.
- [3] GNU Foundation. Gfortran - the gnu fortran compiler. <https://gcc.gnu.org/wiki/GFortran>. Acessado em 25 de abril de 2017.
- [4] David S. Watkins. *Fundamentals of Matrix Computations*, volume 64. John Wiley & Sons, 2 edition, 2004.