MAC0499 – Trabalho de Formatura Supervisionado.

Proposta de Trabalho

Giuliano Augusto Faulin Belinassi
Orientadores: Alfredo Goldman, Marco Dimas Gubitoso

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 provindas 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 de problemas desta espécie é a Dinâmica dos Solos.

Com a evolução dos computadores, era esperado que heurísticas e algoritmos fossem projetados para simular o efeitos de tais vibrações em estruturas. Dentre esses, destacam-se 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 MEC seja interessante do ponto de vista da Engenharia, este trabalho concentra-se na parte computacional do problema, mais especificamente na implementação deste algoritmo de forma a explorar recursos computacionais providenciados por processadores de vários núcleos e Unidades de Processamento Gráfico de Propósito Geral.

2 Objetivo

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

3 Estado do Trabalho

Este trabalho teve inicio em Setembro de 2016 como um projeto de Iniciação Científica da CAPES. A primeira etapa teve como objetivo o estudo da 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 (Versão 77), implicando em:

1. Não compilar com o GFortran [5].
2. Não havia distinções claras entre entrada e saída nas subrotinas.
3. Excessivo uso de GOTOs para efetuar operações básicas, como laços.
4. Dependência de subrotinas específicas do compilador utilizado no projeto original.

Adaptações e manutenções foram necessárias, embora o código estivesse modularizado em subrotinas com comportamento descrito em seus cabeçalhos. Uma descrição breve da estrutura geral do código segue abaixo:

1. Main
    (a) Inputece
    (b) Ghmatece
        i. Singge
            A. Nonsinge
        ii. Nonsinge
    (c) Ghmatecd
        i. Gauleg
        ii. Sing_de
            A. Solfundif
        iii. Nonsingd
            A. Solfund
    (d) DGESV
    (e) Interec
        i. Nonsingd
            A. Solfund
        ii. Sigmaec
    (f) Outputec

Exceto por DGESV, uma subrotina do OpenBLAS [4] que calcula a decomposição LU para solucionar sistemas lineares de variáveis complexas em paralelo, todas as outras subrotinas contém descrições do que fazem em seus arquivos correspondentes.

Tabela 1: *Proffiling* do código sequencial

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

Em seguida, algumas técnicas de paralelização de algoritmos foram estudadas para que fosse possível prosseguir com o trabalho.

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

Observando estes dados, decidiu-se priorizar a paralelização de *Ghmatecd* e *Interec*, pois isto implica em chamadas paralelas das subrotinas *Nonsingd* e *Solfund*.

Como alterar um código pode comprometer funcionalidades que deste dependem, a criação de testes automatizados foi fundamental para assegurar que o resultado obtido condissesse com o original. Embora existam *frameworks* para a criação de testes de unidade em Fortran, por este projeto tratar com código legado com entradas e saídas bem definidas, preferiu-se criar 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 [7].

Concluídos 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 na tabela 3.

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 Inserido	10
Número de Pontos de Gauss	8
Número de Frequências	1
Módulo de Cisalhamento	1.00
Coeficiente de Poisson	0.30
Coeficiente de Amortecimento	0.00
Densidade de Massa	1.00

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

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

Processadores	Tempo
1	1m20s
2	42s
4	22s

plataforma escolhida foi CUDA e o compilador de CUDA para Fortran é pago [6], optou-se por construir uma interface CUDA C $\leftrightarrow$ Fortran.

Prosseguindo com a paralelização em GPGPU, uma implementação de *Ghmatecd* foi codificada em CUDA C [3], porém os resultados não foram satisfatórios, conforme os testes executado em Venus, vide tabelas 4 e 5.

Tabela 4: Tempo gasto em **Venus** na subrotina *Ghmatecd* implementada em CPU

Processadores	Tempo GPU	Flags
1	35s	-Ofast -march=native -flto -funroll-loops
2	26s	-Ofast -march=native -flto -funroll-loops
4	20s	-Ofast -march=native -flto -funroll-loops

Tabela 5: Tempo gasto em **Venus** na subrotina *Ghmatecd* implementada em CUDA

Unidades Gráficas de Proposito Geral	Tempo GPU	Flags
1	30s	-use_fast_math -O3

4 Conclusões

Paralelizar rotinas em um código legado cuja arquitetura não foi projetada para tal pode ser difícil. A ferramenta OpenMP facilita tal trabalho pela baixa quantidade de código que necessita ser modificado para que o objetivo seja concluído. No entanto o mesmo não se aplica à CUDA devido a sua arquitetura peculiar.

Embora Fortran tenha um recurso interessante de declarações implícitas de variável, este deve ser evitado. Em *Solfund.for*, havia um erro de digitação em uma variável que modificava o comportamento do código conforme matrizes eram declaradas em outros métodos. Se o recurso em questão estivesse desabilitado, o erro seria descoberto na implementação original.

5 Apêndice

Informações técnicas dos computadores de teste.

Tabela 6: Pé Grande

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

Tabela 7: Venus

Processador	AMD A10-7700K Radeon R7, 10 Compute Cores 4C+6G @ 3.4GHz
Memória	8Gb
GPU	NVIDIA(R) GeForce(R) GTX 980

Referências

[1] Heinz Antes. A short course on bondary 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] NVIDIA Corporation. Cuda, plataforma de computação paralela. http://www.nvidia.com.br/object/cuda_home_new_br.html. Acessado em 28 de abril de 2017.

[4] OpenBLAS developers. Openblas, an optimizes blas library. http://www.openblas.net. Acessado em 28 de abril de 2017.

[5] GNU Fundation. Gfortran - the gnu fortran compiler. https://gcc.gnu.org/wiki/GFortran. Acessado em 28 de abril de 2017.

[6] PGI Group. Pgi cuda fortran compiler. https://www.pgroup.com/resources/cudafortran.htm. Acessado em 28 de abril de 2017.

[7] David S. Watkins. *Fundamentals of Matrix Computations*, volume 64. John Wiley & Sons, 2 edition, 2004.