Otimizando uma implementação do BEM com GPUs

Giuliano Belinassi, Rodrigo Siqueira, Ronaldo Carrion, Alfredo Goldman, Marco D. Gubitoso

Universidade de São Paulo

Tuesday 17 October, 2017

Introdução

- The Boundary Elements Method (BEM).
- Aplicação: Simulação de propagação de ondas no solo.

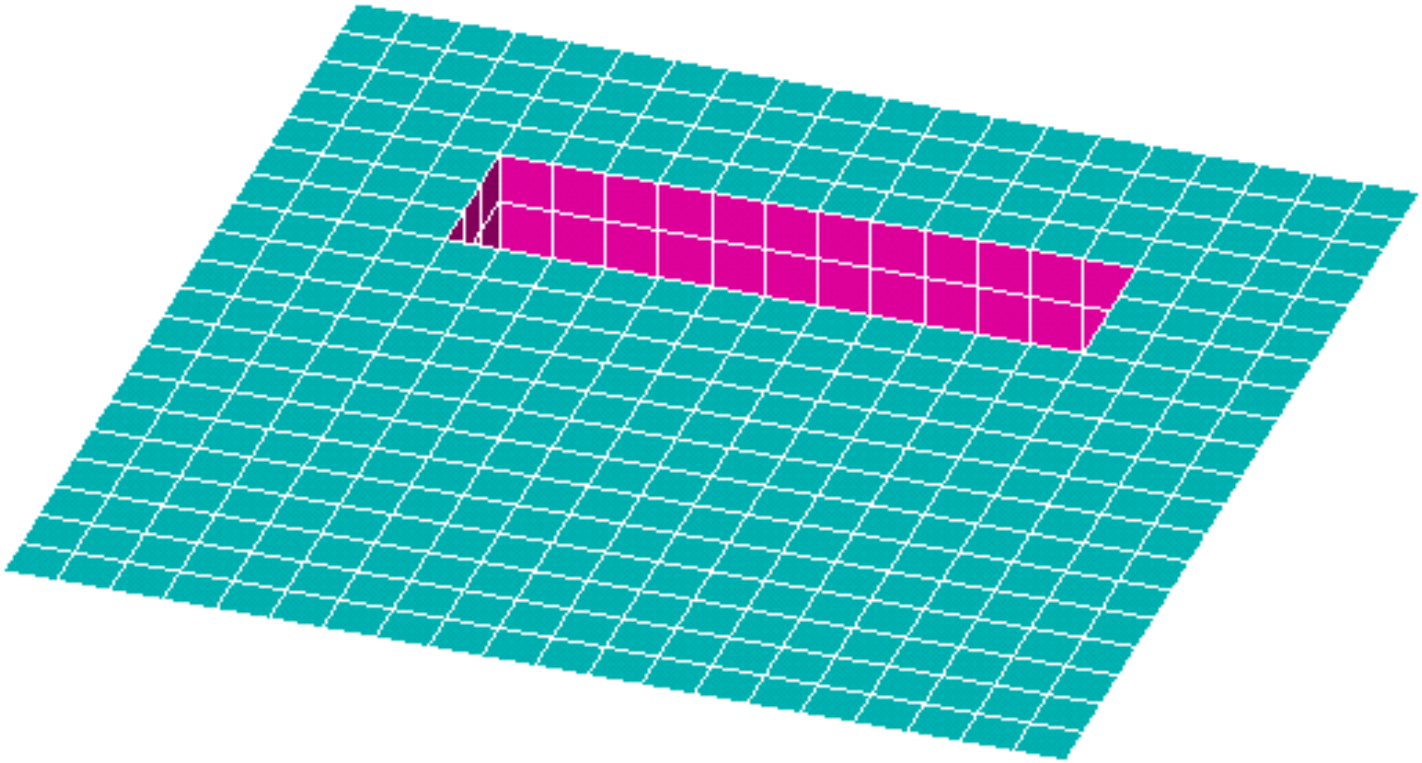

Figura: Exemplo de superfície

O Problema

- Implementação fornecida era sequêncial.
- Para 4000 elementos de malha, o tempo total era de 167s.
- Objetivo: Encontrar as rotinas mais custosas e otimizá-las.
- Subrotina mais custosa: `Ghmatecd`.

  Constrói as matrizes H & G do problema dinâmico.

O Problema

Construir as seguintes matrizes:

$$H, G = \begin{bmatrix} \begin{pmatrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{pmatrix} & \begin{bmatrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{bmatrix} & \begin{bmatrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{bmatrix} & \begin{bmatrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{bmatrix} \\ \begin{bmatrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{bmatrix} & \begin{pmatrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{pmatrix} & \begin{bmatrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{bmatrix} & \begin{bmatrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{bmatrix} \\ \begin{bmatrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{bmatrix} & \begin{bmatrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{bmatrix} & \begin{pmatrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{pmatrix} & \begin{bmatrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{bmatrix} \\ \end{bmatrix}$$

Algorithm 1 Creates $H, G \in \mathbb{C}^{(3m)\times(3n)}$

```
1:  procedure Ghmatecd
2:      for j := 1, n do
3:          for i := 1, m do
4:              ii := 3(i − 1) + 1
5:              jj := 3(j − 1) + 1
6:              if i == j then
7:                  Gelement, Helement ← Sing_de(i)
8:              else
9:                  Gelement, Helement ← Nonsingd(i, j)
10:             G[ii : ii + 2][jj : jj + 2] ← Gelement
11:             H[ii : ii + 2][jj : jj + 2] ← Helement
```

Como paralelizar com OpenMP?

Algorithm 2 Creates $H, G \in \mathbb{C}^{(3m)\times(3n)}$

```
1:  procedure Ghmatecd
2:      #pragma omp parallel for collapse(2)
3:      for j := 1, n do
4:          for i := 1, m do
5:              ii := 3(i − 1) + 1
6:              jj := 3(j − 1) + 1
7:              if i == j then
8:                  Gelement, Helement ← Sing_de(i)
9:              else
10:                 Gelement, Helement ← Nonsingd(i, j)
11:             G[ii : ii + 2][jj : jj + 2] ← Gelement
12:             H[ii : ii + 2][jj : jj + 2] ← Helement
```

E na GPU?

O Problema

- `Nonsingd` e `Sing_de` computam uma integral numericamente.

$$\int_a^b f(x)\mathrm{d}x \approx \sum_{i=1}^{g} w_i f(x_i)$$

- Em nosso caso, avaliar $f(x)$ em um ponto x_i é custoso.
- Uma forma de paralelizar estas rotinas e fazer uma redução.

Expandindo a rotina `Nonsingd`, temos:

```
 1: procedure Ghmatecd_nonsingd
 2:     for j := 1, n do
 3:         for i := 1, m do
 4:             ii := 3(i − 1) + 1; jj := 3(j − 1) + 1
 5:             Allocate Hbuffer & Gbuffer, buffer of matrices 3 × 3 of size g^2
 6:             if i ≠ j then
 7:                 for y := 1, g do
 8:                     for x := 1, g do
 9:                         Hbuffer(x, y) ← GenerateMatrixH(i, j, x, y)
10:                         Gbuffer(x, y) ← GenerateMatrixG(i, j, x, y)
11:             Gelement ← SumAllMatricesInBuffer(Gbuffer)
12:             Helement ← SumAllMatricesInBuffer(Hbuffer)
13:             G[ii : ii + 2][jj : jj + 2] ← Gelement
14:             H[ii : ii + 2][jj : jj + 2] ← Helement
15: procedure Ghmatecd_Sing_de
16:     for i := 1, m do
17:         ii := 3(i − 1) + 1
18:         Gelement, Helement ← Sing_de(i)
19:         G[ii : ii + 2][ii : ii + 2] ← Gelement
20:         H[ii : ii + 2][ii : ii + 2] ← Helement
21: procedure Ghmatecd
22:     Ghmatecd_Nonsingd()
23:     Ghmatecd_Sing_de()
```

Estratégia

- Aloque uma *thread* para cada ponto da quadratura
  - Como a integral é sobre um elemento quadrilateral, temos $g \times g$ pontos.
  - A soma gerará uma matriz 3×3.

$$\underbrace{\begin{bmatrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{bmatrix}}_{\text{Final}} = \underbrace{\begin{bmatrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{bmatrix}}_{\text{Thread 1}} + \underbrace{\begin{bmatrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{bmatrix}}_{\text{Thread 2}} + \cdots + \underbrace{\begin{bmatrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{bmatrix}}_{\text{Thread } g \times g}$$

  - Armazene as fatias na memória compartilhada.
  - Some.

O Problema

$$H, G = \begin{bmatrix} \begin{pmatrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{pmatrix} & \begin{bmatrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{bmatrix} & \begin{bmatrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{bmatrix} & \begin{bmatrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{bmatrix} \\ \begin{bmatrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{bmatrix} & \begin{pmatrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{pmatrix} & \begin{bmatrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{bmatrix} & \begin{bmatrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{bmatrix} \\ \begin{bmatrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{bmatrix} & \begin{bmatrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{bmatrix} & \begin{pmatrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{pmatrix} & \begin{bmatrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{bmatrix} \\ \end{bmatrix}$$

- Aloque um bloco para cada matrix 3×3.

$$\begin{bmatrix} B_{1,1} & B_{1,2} & B_{1,3} & B_{1,4} \\ B_{2,1} & B_{2,2} & B_{2,3} & B_{2,4} \\ B_{3,1} & B_{3,2} & B_{3,3} & B_{3,4} \end{bmatrix}$$

- Compute os elementos da diagonal principal (em vermelho) na CPU.
  A diagonal tem menos elementos.

Testes

- Todos os testes executados em precisão simples.
- Foram utilizados 8 pontos de gauss, totalizando 64 *threads* por bloco.
- Verificamos que $||H_{cpu} - H_{gpu}||_1 < 10^{-4}$ e $||G_{cpu} - G_{gpu}||_1 < 10^{-4}$.
- Todos os experimentos foram executados 30 vezes.
- Processador: AMD A10-7700K; GPU: GeForce 980GTX.

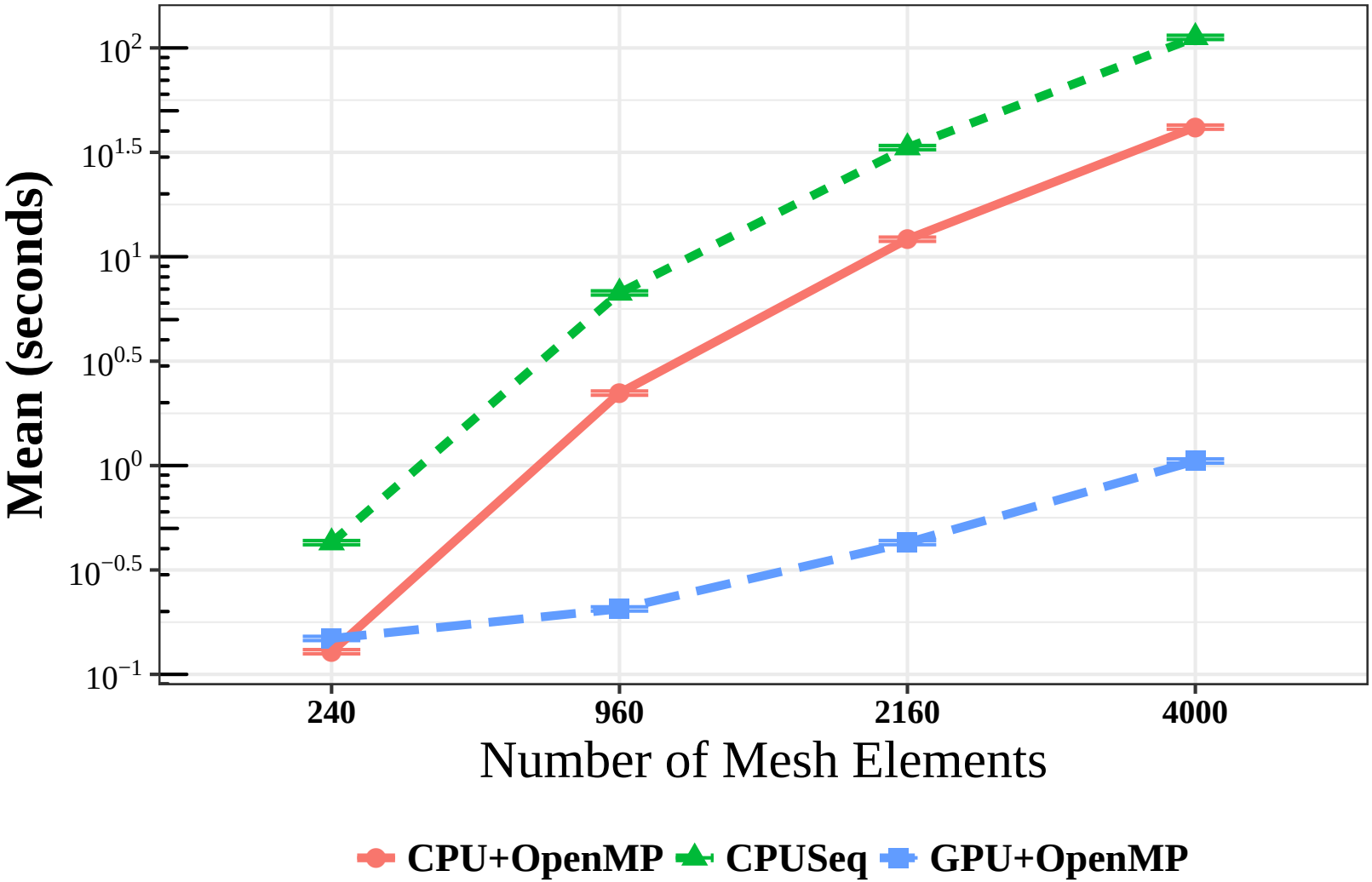

Figura: Time elapsed by each implementation in logarithm scale

Testes

Obrigado!