

# Iniciação a Computação Científica

## Trabalho 2

Giancarlo Klemm Camilo  
Renan Domingos Merlin Greca

Junho de 2015

# Sumário

<b>1</b>	<b>Introdução</b>	<b>3</b>
<b>2</b>	<b>Análise de Arquitetura</b>	<b>4</b>
2.1	Topologia dos Processadores . . . . .	4
2.2	Topografia de Cache . . . . .	4
2.3	Memória . . . . .	4
<b>3</b>	<b>Limite Superior da Discretização</b>	<b>5</b>
<b>4</b>	<b>Tempo de Execução</b>	<b>6</b>
4.1	Programa Original . . . . .	6
4.2	Programa Otimizado . . . . .	7
<b>5</b>	<b>Análise de Funções</b>	<b>8</b>
5.1	Programa Original . . . . .	8
5.1.1	Método de Gauss-Seidel . . . . .	8
5.1.2	Cálculo do Resíduo . . . . .	8
5.2	Programa Otimizado . . . . .	8
5.2.1	Método de Gauss-Seidel . . . . .	8
5.2.2	Cálculo do Resíduo . . . . .	8
5.3	Análise dos Dados . . . . .	8
<b>6</b>	<b>Otimização do Ponto de Interesse</b>	<b>9</b>
6.1	Estrutura de dados . . . . .	9
6.2	Código . . . . .	9
<b>7</b>	<b>Resultados</b>	<b>11</b>
7.1	Tempo . . . . .	11
7.2	Memória . . . . .	11

# 1 Introdução

O objetivo deste trabalho é a implementação de programa para resolver o PDE:

...

Após o programa inicial foi feito, várias alterações foram feitas para melhorar o desempenho. Os métodos utilizados para análise do código, sistema de testes, otimizações de código e de estruturas de dados são descritas nas seções seguintes.

## 2 Análise de Arquitetura

A máquina escolhida para testes foi a **achel** do departamento de informática da UFPR.

### 2.1 Topologia dos Processadores

**Tipo do CPU** Intel Core Westmere processor

**Número de processadores** 2

**Núcleos** 12

### 2.2 Topografia de Cache

**Level 1** 32kB por núcleo

**Level 2** 256kB por núcleo

**Level 3** 12MB por processador

### 2.3 Memória

24GB por processador

### 3 Limite Superior da Discretização

$$T \cong n_x + 1 = n_y + 1 \quad (1)$$

$$|X| = (n_x + 1) \times (n_y + 1) \times 8bytes \cong T^2 \times 8bytes \quad (2)$$

$$|B| = (n_x + 1) \times (n_y + 1) \times 8bytes \cong T^2 \times 8bytes \quad (3)$$

$$|R| = (n_x + 1) \times (n_y + 1) \times 8bytes \cong T^2 \times 8bytes \quad (4)$$

$$|X| + |B| + |R| = 24000000000 \quad (5)$$

$$3 \times T^2 \times 8 = 24000000000 \quad (6)$$

$$T^2 = 1000000000 \quad (7)$$

$$T \cong 31622 \quad (8)$$

A nova versão do programa aceita valores de até aproximadamente 1.732 para  $n_y$  e  $n_x$  numa máquina com 24GB de RAM e desconsiderando o uso de memória virtual. Ou seja, pode computar até 3.000.000 de pontos.

## 4 Tempo de Execução

### 4.1 Programa Original

O programa original foi implementado com a matriz  $A$  completa. Isto fez com que houvesse muito uso de memória, de modo que na achel só foi possível executar discretizações de até 512 separações em  $x$  e  $y$ .

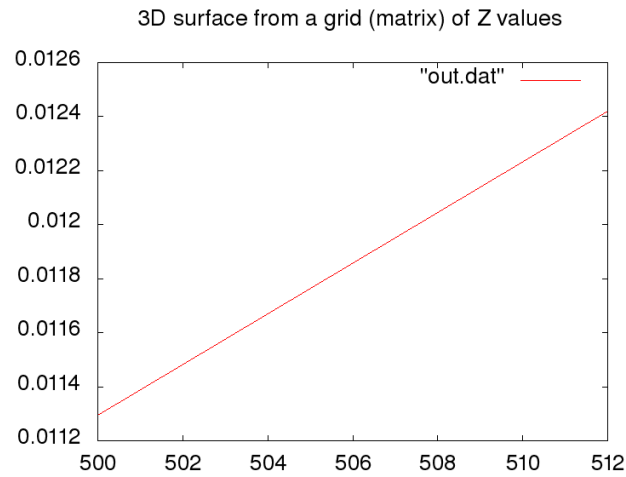


Figura 1: Tempo para  $N\{x,y\} = \{500, 512\}$

## 4.2 Programa Otimizado

Apesar do programa original ser um desastre, o novo programa é razoável, mas mesmo assim uma bosta. Sabe porque? por causa da porra da alemanha chamada likwid! caralho.

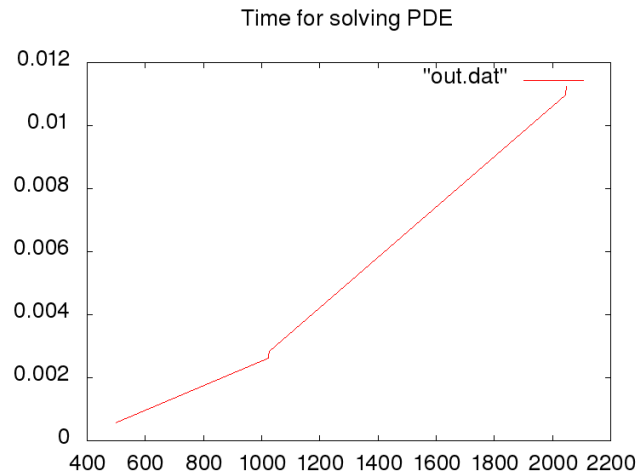


Figura 2: Tempo para  $N\{x,y\} = \{500, 512, 1022, 1024, 1026, 2046, 2048\}$

## 5 Análise de Funções

### 5.1 Programa Original

#### 5.1.1 Método de Gauss-Seidel

#### 5.1.2 Cálculo do Resíduo

### 5.2 Programa Otimizado

#### 5.2.1 Método de Gauss-Seidel

#### 5.2.2 Cálculo do Resíduo

### 5.3 Análise dos Dados



## 6 Otimização do Ponto de Interesse

O ponto de interesse escolhido foi o cálculo do vetor  $x$  no método de Gauss-Seidel. Para isso, otimizações foram feitas nas estruturas de dados usadas durante o cálculo e na estrutura do laço em si.

### 6.1 Estrutura de dados

A estrutura de dados que mais sofreu alterações foi a matriz  $A$ . Na versão original do programa,  $A$  tinha o tamanho de  $((n_x + 1) \times (n_y + 1))^2$ , representando a matriz inteira do método analítico de Gauss Seidel.

Olhando para a matriz  $A$ , percebemos que grande parte das posições tinham valor 0 e que os valores de interesse de cada linha estavam numa distância de  $(n_y + 1)$  da diagonal principal da matriz. Ou seja, os dados que estavam além desse intervalo eram sempre 0 e poderiam ser ignorados.

Além disso, percebemos que as posições ao redor da diagonal principal sempre seguiam o seguinte padrão:

$$h_x \ 0 \ h_y \ 1 \ h_y \ 0 \ h_x$$

Onde o elemento na diagonal principal é sempre 1,  $h_x$  e  $h_y$  representam a dependência dos pontos adjacentes e, no exemplo acima,  $n_y = 2$ . Sendo assim, podíamos ignorar as posições que sempre continham 1 ou 0, além de evitar a repetição de  $h_x$  e  $h_y$ .

Também foi possível ver que os valores de  $h_x$  e  $h_y$  permaneciam constantes em quase todas as linhas da matriz, exceto nas linhas em que não estavam presentes. As linhas que não continham  $h_x$  e  $h_y$  representavam os pontos das bordas da grade, que são calculadas separadamente. Logo, foi possível ver que uma matriz que simplesmente nos dizia se um determinado ponto é ou não uma borda era suficiente para fazer os cálculos de Gauss-Seidel, se salvássemos  $h_x$  e  $h_y$  em variáveis separadas.

Portanto, na versão atual, a matriz  $A$  tem o tamanho de  $(n_x + 1) \times (n_y + 1)$  e utiliza o tipo de dados *short int*, pois apenas armazenamos 0 quando o ponto é uma borda ou 1 caso contrário.

### 6.2 Código

Na versão anterior do programa, o laço de Gauss-Seidel continha quatro desvios condicionais, um para cada borda. Nesta versão, temos apenas um

desvio que utiliza a matriz  $A$  para verificar se o ponto em questão é borda ou não.

Caso o ponto não seja uma borda, quatro operações são realizadas utilizando os valores de  $h_x$  e  $h_y$  e outras posições do vetor  $x$ . Na versão original do programa, as quatro operações eram armazenadas na variável `temp` utilizando o operador `+=`. Isso causava problemas no pipeline da execução, pois gerava uma dependência de dados onde todas as operações em ponto flutuante da linha anterior precisavam ser computadas antes do início dos cálculos da próxima.

## 7 Resultados

### 7.1 Tempo

### 7.2 Memória