CI164
Iniciação à Computação Científica
Trabalho 2

Giancarlo Klemm Camilo
Renan Domingos Merlin Greca

Junho de 2015

Sumário

1 Introdução

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

$$-\Delta u(x,y) + k^2 u(x,y) = f(x,y), (x,y) \in \Omega \tag{1}$$

$$f(x,y) = 4\pi sin(2\pi x) sinh(2\pi y) \tag{2}$$

$$\Omega = [0,2] \times [0,1] \tag{3}$$

$$u(x,1) = sin(2\pi x) sinh(2\pi) \tag{4}$$

$$u(x,0) = u(0,y) = u(2,y) = 0 \tag{5}$$

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

Por processador 24GB

Total 48GB

3 Análise Geral

3.1 Programa Original

3.1.1 Limite Superior da Discretização

$$T \cong n_x + 1 = n_y + 1 \tag{6}$$
$$|A| = ((n_x + 1) \times (n_y + 1))^2 \times 8bytes \cong T^4 \times 8bytes \tag{7}$$
$$|X| = (n_x + 1) \times (n_y + 1) \times 8bytes \cong T^2 \times 8bytes \tag{8}$$
$$|B| = (n_x + 1) \times (n_y + 1) \times 8bytes \cong T^2 \times 8bytes \tag{9}$$
$$|R| = (n_x + 1) \times (n_y + 1) \times 8bytes \cong T^2 \times 8bytes \tag{10}$$
$$|A| + |X| + |B| + |R| = 24000000000 \tag{11}$$
$$(T^4 + 3 \times T^2) \times 8 = 24000000000 \tag{12}$$
$$T^4 + 3 \times T^2 = 3000000000 \tag{13}$$
$$T \cong 234,03 \tag{14}$$

A nova versão do programa aceita valores de até aproximadamente 234 para n_y e n_x numa máquina com 24GB de RAM e desconsiderando o uso de memória virtual e qualquer outro uso de memória que seja necessário.

3.1.2 Tempo de Execução

O programa original foi implementado com a matriz A completa. Isto fez com que houvesse muito uso de memória, de modo que na achel não era possível executar discretizações grandes suficientes para o gráfico de tempo iniciado em 500.

Fizemos uma alteração para reduzir o uso de memória, mas ainda assim só foi possível executar discretizações de até 512 separações em x e y.

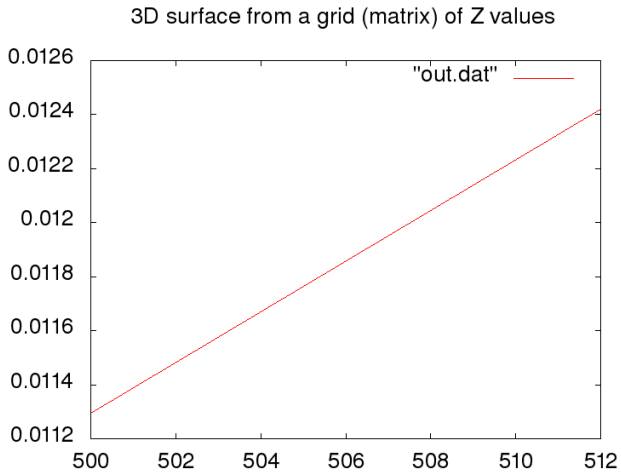

Figura 1: Tempo para N{x,y} = {500, 512} no programa alterado para consumir menos memória

3.2 Programa Otimizado

3.2.1 Limite Superior da Discretização

$$T \cong n_x + 1 = n_y + 1 \tag{15}$$
$$|X| = (n_x + 1) \times (n_y + 1) \times 8bytes \cong T^2 \times 8bytes \tag{16}$$
$$|B| = (n_x + 1) \times (n_y + 1) \times 8bytes \cong T^2 \times 8bytes \tag{17}$$
$$|R| = (n_x + 1) \times (n_y + 1) \times 8bytes \cong T^2 \times 8bytes \tag{18}$$
$$|X| + |B| + |R| = 24000000000 \tag{19}$$
$$3 \times T^2 \times 8 = 24000000000 \tag{20}$$
$$T^2 = 1000000000 \tag{21}$$
$$T \cong 31622 \tag{22}$$

A nova versão do programa aceita valores de até aproximadamente 31.622 para n_y e n_x numa máquina com 24GB de RAM e desconsiderando o uso de memória virtual e qualquer outro uso de memória que seja necessário.

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.

3.2.2 Tempo de Execução

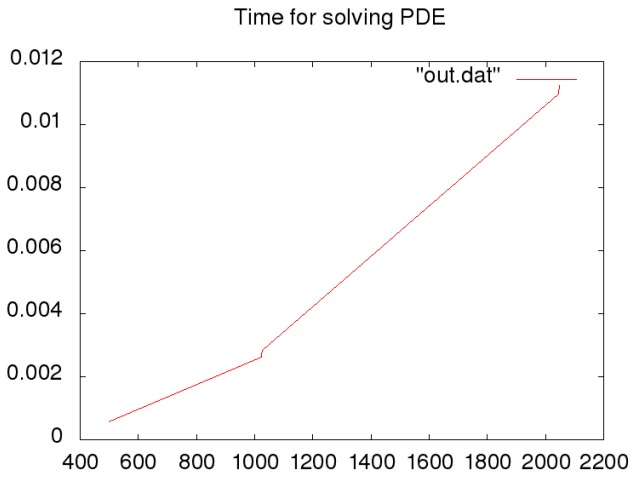

Figura 2: Tempo para N{x,y} = {500, 512, 1022, 1024, 1026, 2046, 2048} no novo programa

4 Análise de Funções

4.1 Programa Original

4.1.1 Método de Gauss-Seidel

4.1.2 Cálculo do Resíduo

4.2 Programa Otimizado

4.2.1 Método de Gauss-Seidel

4.2.2 Cálculo do Resíduo

4.3 Análise dos Dados

5 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.

5.1 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:

$$h_x \;\; \texttt{0} \;\; h_y \;\; \texttt{1} \;\; h_y \;\; \texttt{0} \;\; h_x$$

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, a matriz A passou a ter 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.

Após isso, percebemos que não havia necessidade de armazenar essas informações em um vetor de qualquer maneira. Assim, modificamos o laço do Gauss-Seidel para que percorra apenas as posições relevantes de x e B (que correspondem aos pontos que não são bordas). Dessa forma, é possível

evitar condições durante o laço, pois os elementos de x correspondentes a posições de borda serão 0 e, na multiplicação, tornarão os valores a serem desconsiderados em 0 também.

Na versão nova do programa, então, não há qualquer representação para a matriz A, e essa parte do problema foi resolvida de forma algébrica.

5.2 Código

Na versão anterior do programa, o laço de Gauss-Seidel continha quatro desvios condicionais, um para cada borda. Nesta versão, não temos mais desvios porque, se o resultado de uma operação deve ser desconsiderado, essa operação resultará em 0.

Quatro operações são realizadas utilizando os valores de h_x e h_y e outras posições do vetor x (que são 0 se representa um ponto de borda). 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. Agora, as quatro operações são salvas em variáveis separadas para permitir melhor uso do pipeline. Essas quatro variáveis são então computadas após as quatro operações.

Adicionalmente, utilizamos a técnica de *loop unrolling* de passo 2 para melhorar ainda mais o desempenho do programa. Ao invés de quatro operações principais, cada iteração do laço agora faz oito operações. Experimentamos aumentar o passo do *unroll* para 4, mas a diferença no tempo de execução foi irrisória e optamos por continuar utilizando passo 2 para manter a legibilidade do código.

Todas otimizações foram replicadas no laço que calcula o resíduo do método de Gauss-Seidel e obtivemos melhoras semelhantes de desempenho.

6 Resultados

6.1 Tempo

6.2 Memória