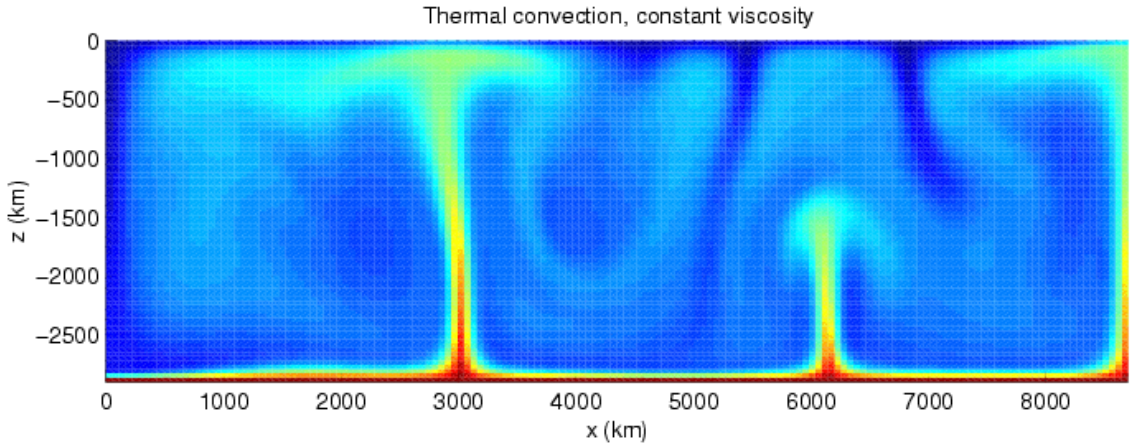
EP2 – Resolvendo a Equação de Calor em Estado Estacionário Usando CUDA

Lucas de Sousa Rosa & Alfredo Goldman

1 de novembro de 2024



Introdução

Neste EP, o objetivo é modificar o código sequencial fornecido para calcular a distribuição de calor em um espaço determinado, paralelizando-o com CUDA. O código sequencial resolve a equação de Laplace que está presente em diferentes contextos na ciência e engenharia.

Inicialmente, utilizaremos um espaço bidimensional (formato quadrado) com condições de contorno fixas (paredes a temperaturas constantes). A partir deste modelo básico, o programa poderá ser adaptado para atender a requisitos adicionais. A equação de calor em estado estacionário é descrita pela equação de Laplace:

$$\frac{\partial^2 h}{\partial x^2} + \frac{\partial^2 h}{\partial y^2} = 0 \qquad (1)$$

A função $h(x, y)$ representa a distribuição de calor em estado estacionário e é uma solução dessa equação. Para resolver a equação de calor e determinar seu valor utilizamos o método de diferenças finitas, já abordado em exercícios anteriores.

Determinação da Distribuição de Calor pelo Método de Diferenças Finitas

Consideremos uma área onde as temperaturas ao longo da borda são conhecidas. O objetivo é calcular a distribuição de temperatura no interior da área. A temperatura em cada ponto interno depende das temperaturas de seus vizinhos. Para encontrar essa distribuição, dividimos a área em uma malha de pontos (x_i, y_j) tal que $i, j \in \{0,1, \dots, n\}$. A temperatura em cada ponto interno pode ser aproximada

pela média das temperaturas de seus quatro pontos adjacentes, como mostra a Figura 1.

Para facilitar os cálculos, descrevemos as bordas usando pontos adjacentes aos internos. Os pontos internos são aqueles em que $1 \leq i, j \leq n-1$ enquanto as bordas estão nos pontos onde i ou j são iguais a 0 ou n. Os valores de temperatura na borda são fixos. Assim, temos ao todo $(N+2)^2$ pontos na malha. A temperatura em cada ponto é calculada iterativamente apenas nos pontos internos:

$$h_{i,j} = \frac{h_{i-1,j} + h_{i+1,j} + h_{i,j-1} + h_{i,j+1}}{4}$$

A iteração continua até que o número de iterações alcance um valor fixo ou que a diferença entre iterações sucessivas seja menor que um valor pré-estabelecido. A dedução dessa fórmula baseia-se no método de diferenças finitas, mas sua explicação detalhada está fora do escopo deste exercício.

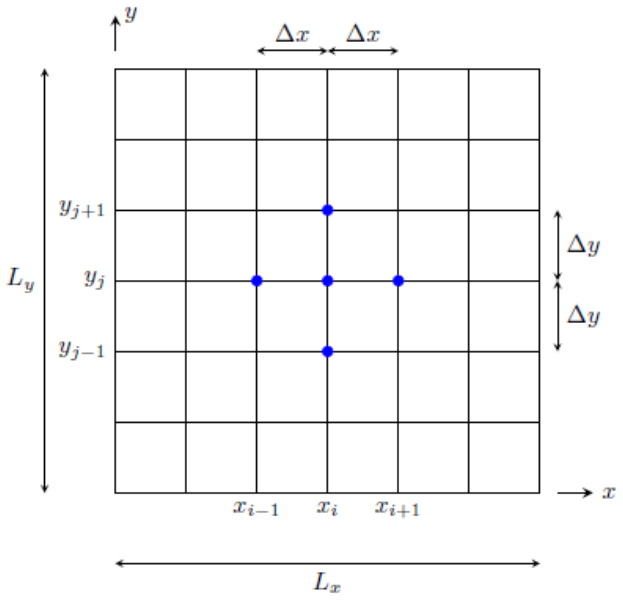


Figura 1. Discretização das coordenadas espaciais. Figura retirada de https://yaredwb.github.io/FDM2D/.

Suponha que a temperatura de cada ponto seja armazenada em um vetor `h[i][j]`, e os pontos de borda `h[0][x]`, `h[x][0]`, `h[n][x]` e `h[x][n]` (para $0 \leq x \leq n$) foram inicializados com as temperaturas das bordas. O cálculo como código sequencial poderia ser:

```
for (iteration = 0; iteration < limit; iteration++) {
    for (i = 1; i < n; i++)
        for (j = 1; j < n; j++)
            g[i][j] = 0.25 * (h[i-1][j] + h[i+1][j] + h[i][j-1] + h[i][j+1]);

    for (i = 1; i < n; i++) // atualização dos pontos, iteração de Jacobi
        for (j = 1; j < n; j++)
```

```
            h[i][j] = g[i][j];
}
```

Note que um segundo vetor `g[][]` é usado para armazenar os novos valores calculados dos pontos a partir dos valores antigos. O vetor `h[][]` é atualizado com os novos valores armazenados em `g[][]`. Isso é conhecido como **iteração de Jacobi**. Multiplicar por 0,25 é feito para calcular o novo valor do ponto, em vez de dividir por 4, pois a multiplicação geralmente é mais eficiente que a divisão. Métodos normais para melhorar a eficiência em código sequencial também se aplicam ao código de GPU e devem ser usados sempre que possível. (Claro, um bom compilador otimizador faria essas alterações.)

Nota: É possível usar o mesmo vetor para os pontos atualizados, utilizando alguns valores recém-calculados para pontos subsequentes (uma iteração de Gauss-Seidel) - isso convergirá significativamente mais rápido, mas pode ser difícil de implementar na GPU, pois implica um cálculo sequencial. No entanto, uma versão sequencial deveria realmente usar a iteração de Gauss-Seidel para fins de comparação ao calcular os *speedups*.

Tarefa 1 – Programa Sequencial

Você deverá modificar o arquivo **`heat.c`** e implementar as funções **`initialize`** e **`jacobi_iteration`**. O código deverá computar a distribuição da temperatura no quarto ilustrado na Figura 2. A temperatura da parede é 20°C e a temperatura da lareira é 100°C. O quarto será dividido em n^2 pontos e o valor de n deve ser lido da linha de comando, assim como o número de iterações.

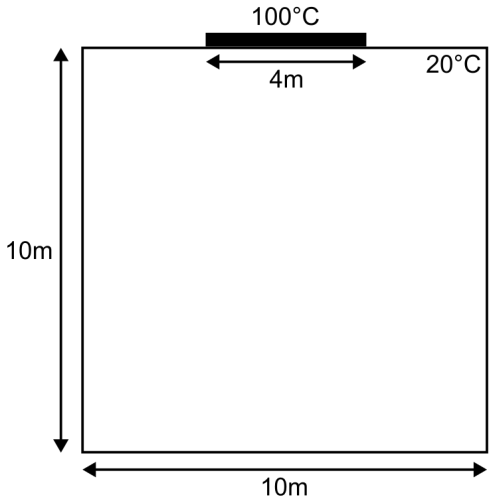


Figura 2. Especificações do quarto.

O código produzirá um arquivo chamado **`room.txt`** que contém os valores de temperatura em cada um dos pontos. O arquivo **`show_room.py`** lê esse arquivo e gera um mapa de calor com os valores (o código depende da biblioteca

`matplotlib`). A Figura 3 ilustra um exemplo de saída. Garanta que seu código seja capaz de produzir uma imagem como essa.

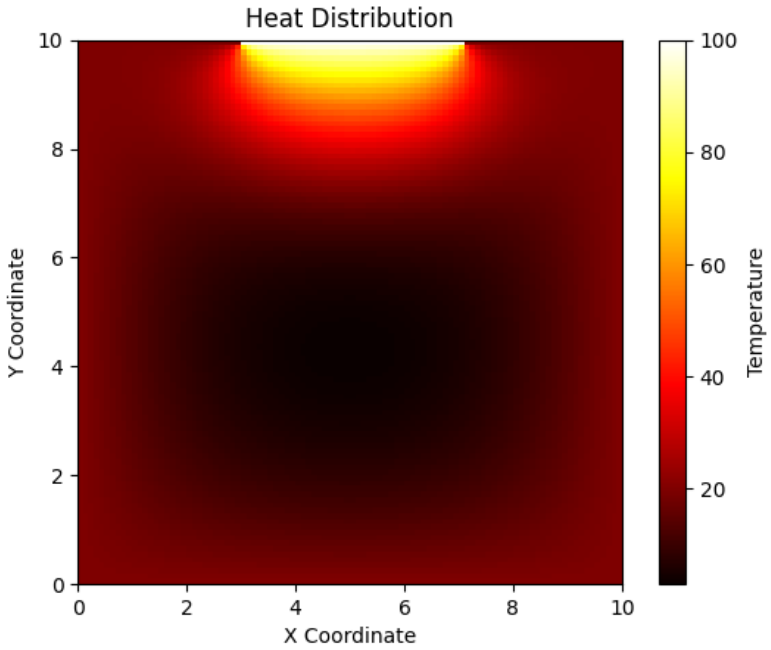


Figura 3. Exemplo de saída. Imagem gerada com 100 pontos e 1000 iterações.

Tarefa 2 – Implementação em CUDA

Crie uma cópia do programa da tarefa 1 e faça as modificações necessárias para que sua implementação utilize CUDA para paralelizar o problema. Sua versão deverá:

- Implementar uma rotina que calcula a distribuição do calor apenas no *host*.
- Implementar algum código que verifica se as versões de CPU e GPU produzem o mesmo resultado correto.
- Suportar diferentes valores para os tamanhos de grade e blocos (CUDA) que sejam obtidos da linha de comando.
    - Número de threads num bloco (t).
    - Número de blocos numa grade (b).
    - Utilizar grades e blocos bidimensionais. Cuidado com os acessos.
- Ser capaz de mensurar o tempo de execução para computação e eventos da GPU. Veja a documentação sobre eventos CUDA para mais informações.
    - Exemplo de eventos: movimentação de dados do *host* para o *device* e vice-versa.
    - A verificação de tempo deverá ser feita tanto para o código de CPU quanto para o código de GPU.

Experimente diferentes valores de entrada e colete os resultados. Teste pelo menos 8 combinações de t, b e n. O número de iterações pode ser fixo. Calcule o *speedup* dessa versão para a versão sequencial.

Tarefa 3 – Corpo Quente no Quarto

Crie uma cópia do código da tarefa anterior e modifique-o para computar a distribuição de calor dentro quarto considerando um corpo quente com temperatura fixa de 37°C (em alguma posição fixada e com tamanho adequado). Lembre-se que a temperatura não deve ser calculada nas dimensões do corpo. Execute um experimento que compara essa versão com a versão da tarefa anterior. Calcule o *speedup*. Observe o que acontece com o desempenho.

Entrega

Cada tarefa e subtarefa especificada será pontuada, por isso, identifique claramente cada parte realizada. Produza um documento em PDF que demonstre que você seguiu as instruções com sucesso e concluiu todas as tarefas, descrevendo as dificuldades encontradas e as ideias desenvolvidas ao longo do processo. Inclua capturas de tela quando necessário e adicione-as ao documento. Apresente conclusões relevantes sobre os experimentos propostos, explicando claramente como foram realizados.

Envie o material até a data limite indicada na página do curso. Inclua todo o código desenvolvido (arquivos **`.c`**/**`.cu`**, **`Makefile`**, etc.) em um arquivo compactado e submeta-o junto com o documento.

Referências

2012. B. Wilkison, C. Ferner. *CUDA Programming Assignment*. Parallel Programming Course.

https://en.wikipedia.org/wiki/Heat_transfer

https://en.wikipedia.org/wiki/Heat_equation

https://yaredwb.github.io/FDM2D/