MAC 0219/5742 - Introdução à Computação Paralela e Distribuída

EP 1 - Simulação paralela de um gás ideal usando Pthreads e OpenMP

Alfredo Goldman e Vitor Terra

1. Introdução

Autômatos celulares são um modelo discreto de computação que consiste em um arranjo de células interligadas, cada uma possuindo um estado próprio (por exemplo: “ligado” ou “desligado”). A cada iteração, o estado de cada célula é atualizado simultaneamente de acordo com alguma regra, geralmente em função dos estados das células vizinhas. Em geral, é possível paralelizar a simulação de autômatos celulares, já que a atualização de cada célula é independente das demais, dependendo apenas de estados anteriores.

Uma aplicação interessante de autômatos celulares é a simulação do comportamento físico de fluidos, em particular na presença de barreiras fixas. Uma forma de resolver o problema é modelar o fluido de forma contínua, em termos de grandezas macroscópicas como pressão e densidade, e então resolver numericamente as equações de Navier-Stokes com as devidas condições de contorno.

Surpreendentemente, é possível reproduzir comportamentos macroscópicos previstos pelas equações de Navier-Stokes usando modelos microscópicos discretos baseados em autômatos celulares. Tais modelos são conhecidos como *lattice gas automata* (LGA), ou "autômatos de gás em rede”, em tradução livre. Neste EP, será fornecida uma implementação sequencial de um LGA, que deverá ser paralelizada de duas formas diferentes: usando a biblioteca Pthreads e as diretivas do OpenMP.

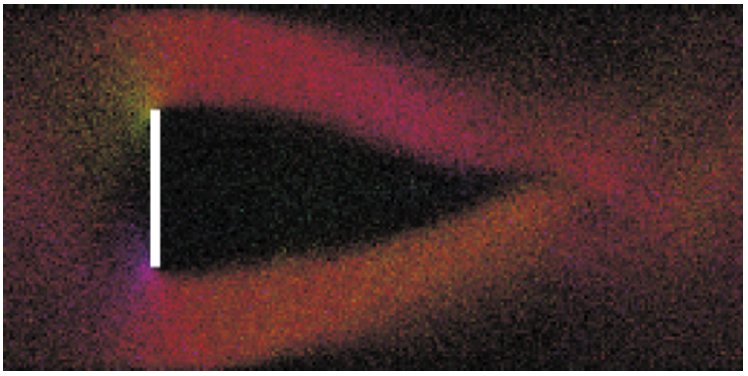

Figura 1: Simulação do fluxo ao redor de uma barreira plana em um LGA. As diferentes cores representam a direção média do fluxo de partículas [1].

2. Descrição do modelo

Para este EP, foi adotada uma versão simplificada[1] do modelo FHP, proposto por Frisch, Hasslacher e Pomeau em 1986 [2]. Neste modelo, o espaço bidimensional é particionado entre as células de um autômato celular, ligadas entre si em um padrão hexagonal, porém armazenado na memória como uma matriz quadrada com algumas conexões na diagonal. Em outras palavras, cada célula possui seis células vizinhas, com exceção daquelas na fronteira do espaço.

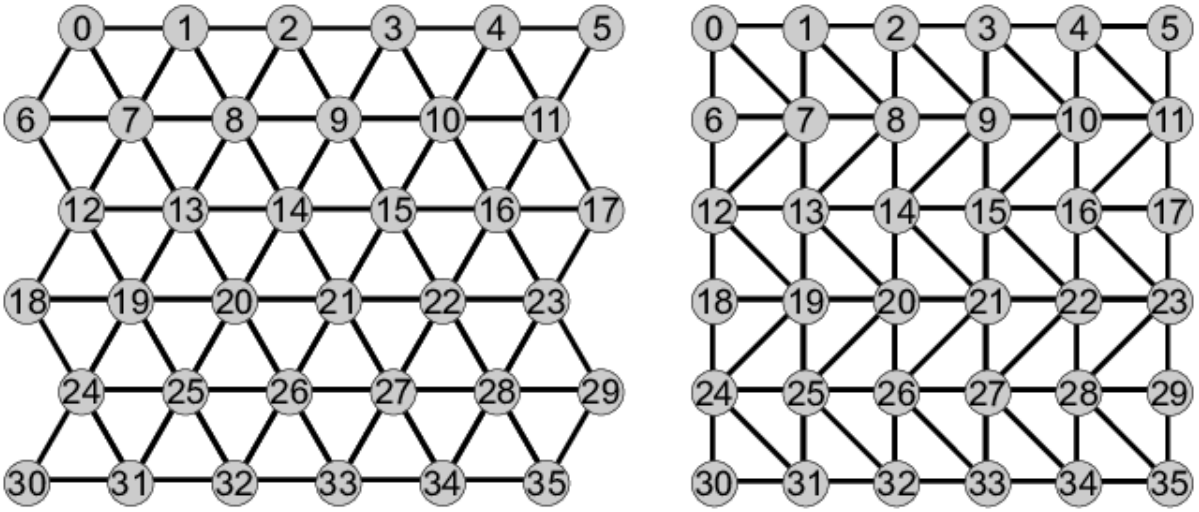

Figura 2: Esquema representando as células de um LGA ligadas em uma rede hexagonal (esq), assim como a sua representação correspondente como uma matriz quadrada [1].

As partículas de gás são idênticas entre si, possuem velocidade de mesmo módulo (movem-se uma célula a cada iteração) e podem se mover ao longo das seis direções da rede hexagonal. Cada célula pode conter até seis partículas de gás, no máximo uma em cada direção. Além disso, uma célula pode fazer parte de uma parede fixa (barreira), mantendo-se imóvel ao longo do tempo.

Dessa forma, o estado de cada célula pode ser codificado em um byte (`unsigned char`). Os possíveis valores numéricos de cada célula são:

- `0b00000000` (0): um espaço vazio;
- `0b00000001` a `0b00111111` (1 a 63): uma ou mais partículas de gás - se o bit de ordem *i* estiver setado, então há uma partícula se movendo na direção *i*;
- `0b01000000` (64): uma barreira imóvel.

[1] No modelo original, algumas colisões dependem de um fator probabilístico. Para este EP, o resultado das colisões é determinístico, para facilitar a verificação da correção da implementação paralela.

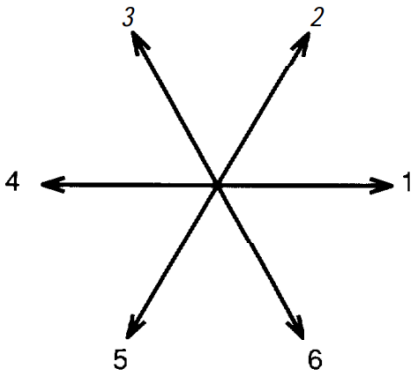

Figura 3: As seis possíveis direções de movimento de uma partícula na rede hexagonal, cada uma correspondente a um bit do estado da célula [3].

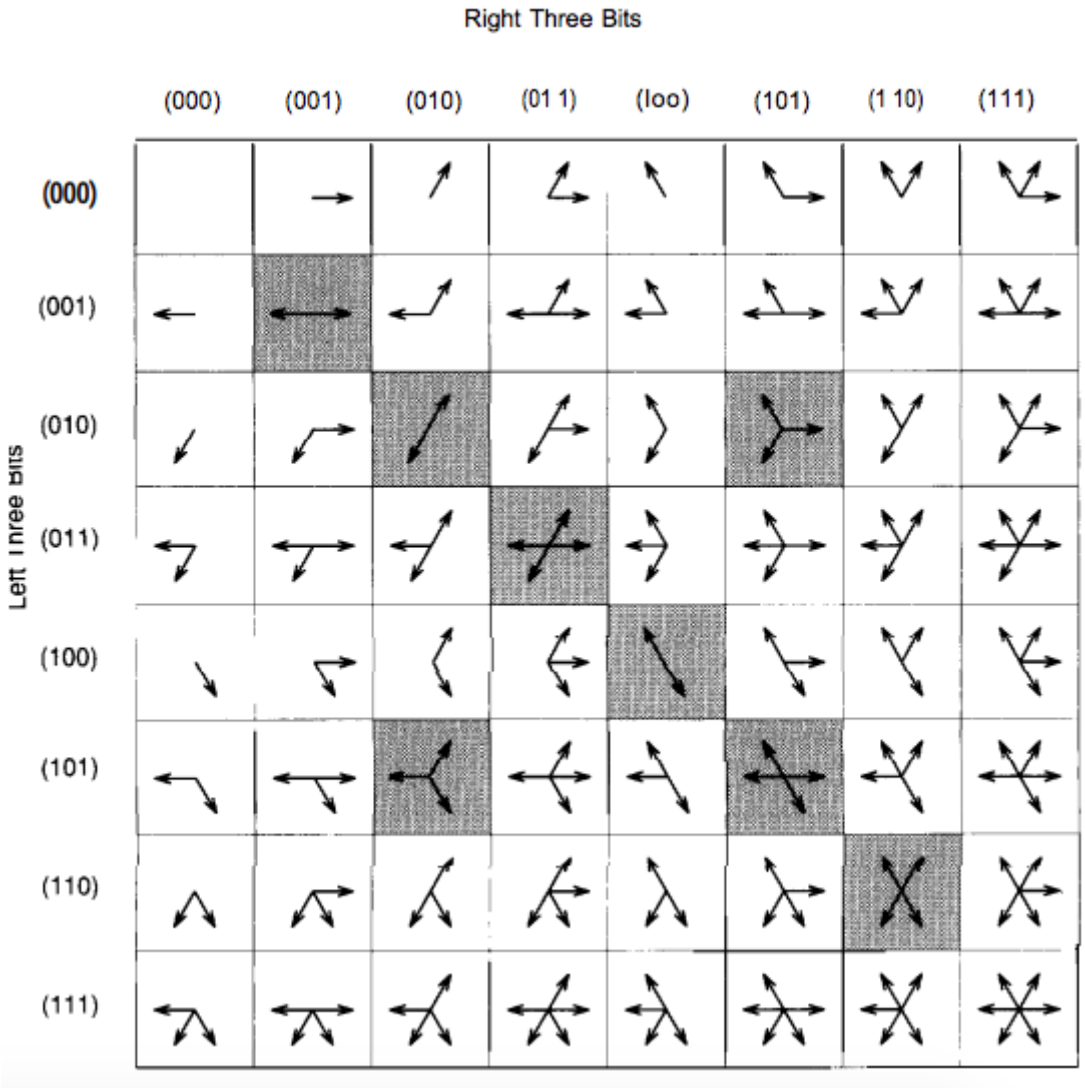

Figura 4: Possíveis estados de uma célula no LGA (exceto a barreira), correspondendo a diferentes números de partículas e direções de movimento. Em destaque, estão as configurações que levam a colisões [3].

A atualização do estado de cada célula é feita em duas etapas: propagação e colisão, respeitando os princípios de conservação da massa e do momento linear. Na etapa de propagação, cada partícula se move uma unidade na direção indicada por sua velocidade. Caso essa partícula encontre uma barreira sólida, a sua velocidade e posição são atualizadas buscando conservar o momento na direção paralela à parede.

Após a etapa de propagação, duas ou mais partículas podem acabar colidindo entre si. Colisões podem ocorrer:

- entre duas partículas que se encontram frontalmente;
- entre três partículas que se encontram simetricamente;
- entre quatro partículas que se encontram frontalmente em pares.

Tais casos estão destacados na tabela de possíveis estados apresentada na Figura 4. Quando ocorre colisão, a velocidade de cada partícula envolvida é defletida em 60 graus no sentido anti-horário. Caso alguma dessas configurações seja encontrada após a propagação, o valor da célula é atualizado de modo a refletir a mudança na direção das partículas. A Figura 5 mostra alguns exemplos de configurações antes e depois de uma colisão:

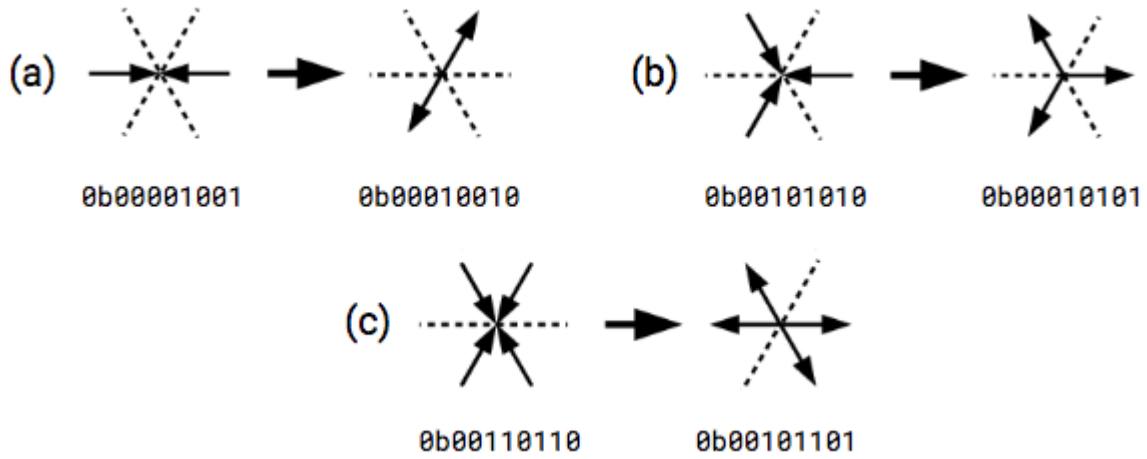

Figura 5: Exemplos de configurações antes e depois de uma colisão (adaptada de [1]).

Todas as regras apresentadas já estão implementadas na versão sequencial do código e foram explicadas aqui para se ter maior clareza a respeito do problema. Para este exercício, não é necessário entender profundamente os detalhes do modelo, e sim identificar quais trechos do código podem ser facilmente paralelizados.

3. Código fornecido

O código fornecido para o trabalho está disponível no GitHub[2] e contém os seguintes arquivos, além de eventuais arquivos *header* (`.h`) correspondentes:

- `lga_base.c/lga_base.h`: contém funções e definições usadas por demais arquivos na simulação, incluindo funções para *debug* (por exemplo, `print_grid`);
- `lga_seq.c`: contém uma versão sequencial da simulação do LGA;
- `lga_omp.c`: inicialmente contém uma cópia da versão sequencial, que deverá ser paralelizada usando OpenMP;
- `lga_pth.c`: inicialmente contém uma cópia da versão sequencial, que deverá ser paralelizada usando Pthreads;
- `check.c`: gera um executável `check`, que executa as três versões para um dado tamanho de entrada e um dado número de *threads*, verificando se o estado final obtido está correto (ou seja, igual ao resultado sequencial);
- `time_test.c`: gera um executável `time_test`, que mede o tempo de execução, dados o tamanho de entrada, a versão do programa (sequencial, OpenMP ou Pthreads), e o número de *threads*;
- `grid_gen.c`: usado para gerar arquivos binários de entrada contendo o estado inicial do LGA;
- `time_extra.c`: contém funções utilizadas para medição do tempo transcorrido;
- `Makefile`: contém instruções para compilar o código;
- Arquivos `x.in`: são os arquivos de entrada gerados por `grid_gen` contendo o estado inicial da simulação, onde `x` é igual ao número de linhas e de colunas.

Para compilar o projeto por completo, use o comando:

```
$ make
```

Para compilar separadamente cada um dos executáveis, use os seguintes comandos:

```
$ make check
```

```
$ make time_test
```

Para testar se as implementações paralelas estão corretas, use o comando `check`:

```
$ ./check --grid_size <grid_size> --num_threads <num_threads>
```

onde `grid_size` é o tamanho da entrada e `num_threads` é o número de *threads*.

[2] https://github.com/vitortterra/MAC0219-5742-EP1-2023

Para medir o tempo de execução, use o comando `time_test`:

```
$ ./time_test --grid_size <grid_size> --impl <impl> --num_threads
                        <num_threads>
```

onde `impl` é a versão do programa a ser executada (`<impl> = seq`, omp ou `pth`). No caso em que `<impl> = seq`, o argumento `num_threads` pode ser omitido (caso seja inserido, é ignorado).

Para remover os arquivos gerados na compilação, use o comando

```
$ make clean
```

Não é necessário entender todos os detalhes do código fornecido para fazer o exercício. Procure entender em alto nível a função `update` do arquivo `lga_seq.c` para pensar em como implementar versões paralelizadas em `lga_omp.c` e `lga_pth.c`.

4. Tarefas

Você e seu grupo deverão paralelizar o código para a simulação do LGA usando a biblioteca Pthreads e as diretivas de compilador fornecidas pelo OpenMP. Depois, vocês deverão medir e comparar o tempo de execução das três versões: sequencial, paralelizada com Pthreads e paralelizada com OpenMP.

Para cada uma das três versões do programa, vocês deverão realizar medições do tempo de execução para diferentes tamanhos de entrada (`grid_size`). Nas versões paralelizadas vocês deverão também medir, para cada tamanho de entrada, o tempo de execução para diferentes números de threads (`num_threads`). Obtenham o tempo de execução combinando os seguintes valores dos parâmetros de entrada:

```
grid_size = 32, 64, 128, 256, 512, 1024, 2048, 4096

num_threads = 1, 2, 4, 8, 16, 32 (versões omp e pth)
```

Vocês devem fazer um número de medições e analisar a variação dos valores obtidos. Sugerimos 10 medições para cada experimento, e também que vocês usem a média e o intervalo de confiança das 10 medições nos seus gráficos. Caso observem variabilidade muito grande nas medições, resultando num intervalo de confiança muito grande, vocês podem realizar mais medições, sempre apresentando a média e o intervalo de confiança. Não é recomendado fazer menos de 10 medições.

Utilize o arquivo executável `time_test` para executar os experimentos, passando diferentes parâmetros por um script via linha de comando. Para os experimentos, utilize os arquivos de entrada já disponibilizados com o código inicial, de modo a diminuir a variabilidade das medições.

Depois de realizar os experimentos, vocês deverão elaborar gráficos que evidenciem o comportamento das três versões do programa com relação à variação dos parâmetros descritos anteriormente. Os gráficos deverão ser claros e legíveis, com eixos nomeados. Deverão apresentar a média e o intervalo de confiança das (pelo menos) 10 execuções para cada cenário experimental. A automação dos experimentos e da visualização dos dados gerados é fundamental para a pesquisa em Ciência da Computação, pois permite gerar e analisar grandes conjuntos de dados sem muito esforço manual.

Os gráficos devem ser incluídos em um relatório, analisando como e por que as três versões do programa se comportam com a variação do tamanho da entrada e do número de *threads*.

5. Entrega

Vocês deverão entregar no e-Disciplinas um único arquivo .zip por grupo, com os nomes dos integrantes, contendo:

- Um relatório em .pdf com as análises e gráficos;
    - Relatórios em .doc, .docx ou .odt não serão aceitos.
- Os arquivos `lga_omp.c` e `lga_pth.c` contendo as versões paralelizadas do programa;
    - Caso tenha sido necessário modificar outros arquivos, inclua-os na entrega também.
- Um arquivo .csv com as medições feitas.

A nota do EP1 vai de **0.0** a **10.0**, e a avaliação será feita da seguinte maneira:

- Apresentação e análise de medições para o programa sequencial: vale **2.0**, divididos da seguinte maneira:
    - Relatório: **2.0**
        - Apresentação e análise dos Experimentos: **1.8**
        - Clareza do texto e figuras: **0.2**

- Apresentação e análise de medições para o programa em OpenMP: vale **4.0**, divididos da seguinte maneira:
    - Implementação: **2.0**
        - Código compila sem erros e *warnings*: **0.9**
        - Código executa sem erros e produz o resultado correto: **0.9**
        - Boas práticas de programação e clareza do código: **0.2**
    - Relatório: **2.0**
        - Apresentação e análise dos Experimentos: **1.8**
        - Clareza do texto e figuras: **0.2**

- Apresentação e análise de medições para o programa em Pthreads: vale **4.0**, divididos da seguinte maneira:
    - Implementação: **2.0**
        - Código compila sem erros e *warnings*: **0.9**
        - Código executa sem erros e produz o resultado correto: **0.9**
        - Boas práticas de programação e clareza do código: **0.2**
    - Relatório: **2.0**
        - Apresentação e análise dos Experimentos: **1.8**
        - Clareza do texto e figuras: **0.2**

Em caso de dúvidas, use o fórum de discussão do e-Disciplinas ou entre em contato diretamente com o monitor (vitortterra@ime.usp.br) ou o professor (gold@ime.usp.br).

6. Referências

Agradecimentos aos monitores de versões anteriores da disciplina, pois partes da estrutura deste EP (tanto código quanto enunciado) foram baseadas em EPs de anos anteriores. Este EP também foi baseado em uma postagem [4] feita por Mohamed Gaber. Um exemplo de referência em portugues sobre o assunto é [5].

[1] - Johnson, Mitchel, Daniel Playne, e Ken Hawick. 2010. “Data-Parallelism and GPUs for Lattice Gas Fluid Simulations.” *PDPTA’10 Inproceedings*, 210-216.

https://www.massey.ac.nz/~dpplayne/Papers/cstn-109.pdf

[2] - Frisch, U., Hasslacher, B. e Pomeau. Y. 1986. “Lattice-Gas Automata for the Navier-Stokes Equation”. *Physical Review Letters,* 56, 1505-150.

https://journals.aps.org/prl/pdf/10.1103/PhysRevLett.56.1505

[3] - Hasslacher, Brosl. 1987. “Discrete Fluids.” *Los Alamos Science*, 187-189. Special issue.

https://sgp.fas.org/othergov/doe/lanl/pubs/00285743.pdf

[4] - A Gentle Introduction to Lattice Gas Automaton for Simulation of Fluid Flow with Python. [The FHP model for Navier-Stokes Equations] | by Mohamed Gaber

[5] - Schepke, Claudio. 2007. “Distribuição de dados para implementações paralelas do Método de Lattice Boltzmann.” *Dissertação (Mestrado em Ciência da Computação) - Instituto de Informática, Universidade Federal do Rio Grande do Sul*, 25-26.

https://www.lume.ufrgs.br/bitstream/handle/10183/8810/000589374.pdf