Universidade Tecnológica Federal do Paraná - UTFPR

Centro de Pesquisa em Reologia e Fluidos Não Newtonianos - CERNN

Engenharia de Computação

Implementação do LBM em código próprio e validação dos dados obtidos

Aluno:
Waine B. de Oliveira Junior

Relatório

Orientador: Dr. Admilson Teixeira Franco
Coorientador: MSc. Alan Lugarini de Souza

27 de Março de 2019

Conteúdo

1 Introdução

O LBM (*Lattice Boltzmann Method*) vem atingindo grande sucesso para simulação de escoamentos de fluidos. Dois dos motivos são sua otimização computacional e a simplicidade de implementação com relação a outros métodos. Nesse sentido, a otimização do método vem principalmente de sua paralelização, a qual permite que sejam aplicadas técnicas e arquiteturas de computação paralela para sua implementação, resultando assim, em uma diminuição considerável de tempo de processamento com relação ao sequencial [3, 6].

Apesar de existirem vários códigos abertos de LBM disponíveis, sendo o *OpenLB* [4] um dos mais conhecidos, para uma otimização do desempenho a confecção de um código próprio é a alternativa mais viável. Pois, para melhora do desempenho, deve levar-se em conta minúcias tanto do computador quanto do caso a ser simulado.

Na implementação e nos testes foram escolhidos dois casos de escoamento, sendo um deles o escoamento em uma cavidade com tampa deslizante, que consiste num fluido contido em um quadrado, tal que a parede superior desse possui certa velocidade, e o outro o escoamento entre placas paralelas, em que um fluido está entre duas placas paralelas e há uma diferença de pressão entre a entrada e saída.

Os objetivos deste relatório são: a implementação de um código próprio para simulação dos casos de escoamento de cavidade com tampa deslizante e de placas paralelas; a comparação entre os valores obtidos pelo código próprio, com o *OpenLB* e com o valor analítico, no caso das placas, ou um método de maior precisão [2], no caso da cavidade; a validação dos dados obtidos por meio do teste da ordem do erro de truncamento.

2 LBM

O LBM é um método para simulações de escoamento de fluidos que tem como principal característica computacional, em relação a métodos mais convencionais, a facilidade para sua implementação e a possibilidade de sua paralelização.

Por ser um método mesoscópico, toma como base funções de distribuição (também chamadas de populações) como variáveis físicas, ao contrário de métodos convencionais que utilizam pressão ou velocidade, por exemplo, como base para cálculo. É importante ressaltar que variáveis macroscópicas podem ser obtidas a partir das populações de um nó.

O LBM é definido por meio de nós, em que cada um possui populações com

certa velocidade e sentido definidos pelo *velocity set*. Nesses nós são aplicadas as operações de colisão entre as populações e então elas são propagadas para o nó adjacente, no sentido de sua velocidade. A densidade, em função das populações, é dada pela equação 1 [5].

$$\rho(\vec{x}, t) = \sum_i f_i(\vec{x}, t) \tag{1}$$

Em que ρ representa a densidade na posição $\vec{x}$ e tempo t e f_i é a função distribuição i em $\vec{x}$ e tempo t. A velocidade também pode ser obtida por meio das populações do nó, sendo ela dada pela equação 2 [5].

$$\vec{u}(\vec{x}, t) = \sum_i f_i(\vec{x}, t) \cdot \vec{c_i} \tag{2}$$

Em que u representa a velocidade na posição $\vec{x}$, tempo t e na direção e sentido de c_i, vetor unitário na direção de algum dos eixos (x, y ou z), e f_i é a função distribuição i em $\vec{x}$ e tempo t.

O método, a princípio, é dividido em cinco etapas: inicialização, colisão, propagação, aplicação das condições de contorno e atualização dos valores macroscópicos. Seu ciclo pode ser representado pelo fluxograma da figura 1.

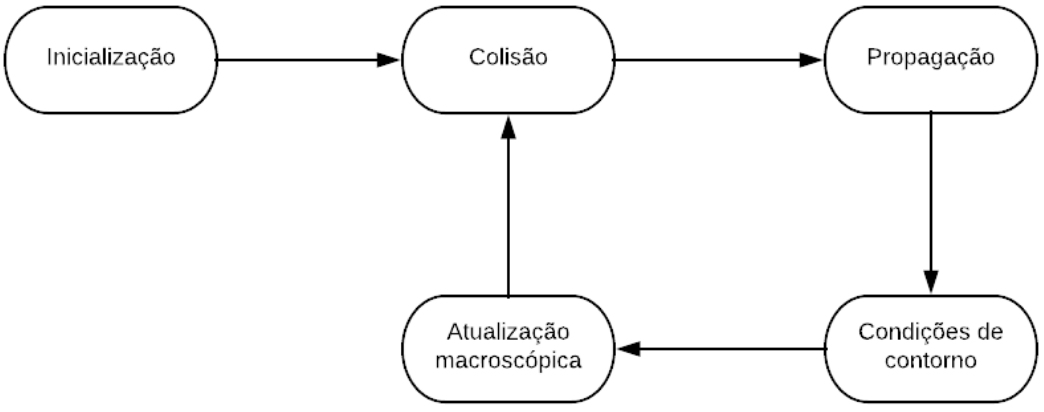

Figura 1: Fluxograma do LBM

A inicialização dos nós é feita usualmente a partir de valores padrões, com

todas velocidades em zero e as densidades em um. Já a colisão é definida por meio da equação 3 [5].

$$f_i^*(\vec{x},t) = f_i(\vec{x},t) + \Omega_i(\vec{x},t) \tag{3}$$

Sendo $f_i(\vec{x},t)$ a população i atual e $f_i^*(\vec{x},t)$ a população i após a colisão em $\vec{x}$ e tempo t, já $\Omega_i(\vec{x},t)$ é o operador de colisão. Para o código próprio foi escolhido o operador BGK, definido pela equação 4 [5].

$$\Omega_i(f) = \frac{f_i - f_i^{eq}}{\tau} \tag{4}$$

Na qual τ é o tempo de relaxamento (para fluidos newtonianos é constante e o mesmo para todos nós), e f_i^{eq} é a população de equilíbrio do nó, dada pela equação 5 [5].

$$f_i^{eq}(\vec{x},t) = \rho w_i \left(1 + \frac{\vec{u}\cdot\vec{c_i}}{c_s^2} + \frac{(\vec{u}\cdot\vec{c_i})^2}{2c_s^4} - \frac{\vec{u}\cdot\vec{u}}{2c_s^2}\right) \tag{5}$$

Em que o peso, w_i, e a velocidade do som, c_s, são dados pelo *velocity set*.

Após a colisão, é feita a propagação, em que, como o nome indica, as populações já colididas são propagadas conforme seu sentido. Ela é definida pela equação 6 [5].

$$f_i(\vec{x} + \vec{c_i}\Delta t, t + \Delta t) = f_i^*(\vec{x},t) \tag{6}$$

Sendo c_i o sentido da população, f_i^* a população após colisão e Δt o tempo para propagação. Então é feita a atualização dos valores macroscópicos a partir das equações 1 e 2. Vale ressaltar que densidade e velocidade são usualmente salvas logo após o passo da atualização.

3 *OpenLB*

O *OpenLB* [4] é um projeto de código aberto, orientado a objetos e escrito em C++, para implementação do método de Lattice Boltzmann. É muito conhecido e utilizado pela comunidade de CFD para simulação utilizando LBM.

Um dos principais pontos positivos da utilização de um código já feito é sua praticidade para simulação, pois as funções, como a de colisão ou de condições de

contorno, já estão implementadas. Isso torna possível simular um escoamento com LBM sem implementar o método computacionalmente. Além disso, outros fatores especificamente do *OpenLB* são: a possibilidade de importar uma malha a partir de um arquivo externo; a fácil alteração entre os esquemas de cálculo das condições de contorno; capacidade de utilizar tempos de relaxamento distintos para cada nó e de adicionar forças externas; salvar variáveis em arquivos de diversos formatos, dentre outros.

Em suma, os principais benefícios que o *OpenLB* traz com relação ao código próprio estão relacionados a gama de opções que ele apresenta e a possibilidade de simular vários casos a partir do mesmo código base. Contudo, para a implementação de rotinas de simulação, é necessária a leitura da documentação do código que, apesar de extensa, deixa a desejar em alguns aspectos, não apresentando diagrama de classes ou diagramas de sequência, por exemplo. Por esse motivo, as rotinas geralmente são feitas a partir de algum tutorial do *OpenLB*, alterando apenas as variáveis e funções necessárias.

Além disso, a paralelização é implementada apenas para MPI, ou seja, não é possível paralelizar por meio de API para placas de vídeo. Vale mencionar a existência de erros no código, encontrados durante os testes do exemplo da "*cavity2d*", no qual os eixos para impressão de dados estavam invertidos. O que mostra como os resultados não podem ser totalmente confiáveis.

O *OpenLB* é uma ótima opção para simulação do LBM do ponto de vista de praticidade, porém apresenta várias restrições de otimização e exige conhecimento de suas classes e suas funções, o que pode tomar considerável tempo de estudo.

4 Código Próprio

A motivação da produção de um código próprio é a possibilidade de otimização deste, posto que simulações utilizando LBM podem tomar muito tempo, chegando a dias de duração. Assim, é de grande interesse a otimização do código para diminuir o tempo de processamento computacional.

Para desenvolvimento, foi escolhida a linguagem de programação C++, por possibilitar controle de memória e processamento a baixo nível, e o ambiente de desenvolvimento integrado *Microsoft Visual Studio* 2017 [1], por possuir ferramentas de depuração e teste de alta qualidade.

O código feito visou simular os casos de escoamento da cavidade com tampa deslizante e entre placas paralelas, utilizando o operador de colisão BGK [5] e a

condição de contorno de Zou-He [8]. Para tal, foram implementadas as funções de inicialização das populações, colisão, propagação, aplicação das condições de contorno, cálculo da densidade e da velocidade macroscópicas e funções auxiliares para salvar os dados da simulação.

Considerando que os maiores gargalos quanto ao desempenho do LBM estão relacionados ao acesso à memória, principalmente a localidade temporal e espacial do programa [5], foram evitados tipos abstratos, os quais ocupam mais espaço que tipos primitivos, diminuindo assim a localidade espacial. Outras técnicas utilizadas para melhora de desempenho foram: o *loop unrolling* para iterações envolvendo as populações de um nó; a propagação caso a caso, não utilizando laços condicionais para checar sua validade; explicitação da somatória para cálculo das velocidades e densidades macroscópicas.

```
int x[5];

// normal loop
for(int i = 0; i < 5; i++)
    x[i] = func(i);

// unrolled loop
x[0] = func(0);
x[1] = func(1);
x[2] = func(2);
x[3] = func(3);
x[4] = func(4);
```

Figura 2: Exemplo *loop unrolling* para um vetor de tamanho cinco

Pelo fato da população zero não ser alterada pela propagação, ela foi definida em uma variável diferente das demais populações. Para as populações zero, as densidades e as velocidade macroscópicas, foram alocados vetores bidimensionais, já para as populações entre um e oito, foi alocado um vetor tridimensional. Como os vetores foram alocados dinamicamente, eles não são contíguos na memória, o que diminui drasticamente a localidade espacial e temporal.

5 Resultados

Para análise do código próprio foram realizadas comparações entre esse e o *OpenLB*, por meio do erro e da performance de ambos para o escoamento em cavidade com tampa deslizante. Já a validação de dados foi feita a partir do teste da ordem de erro de truncamento para os dois casos de escoamento.

A seguir são apresentadas a comparação entre os resultados do código próprio e do *OpenLB* a validação dos dados daquele.

5.1 Código próprio vs. *OpenLB*

Tanto o operador de colisão quanto o esquema de velocidades e de condição de contorno utilizados no *OpenLB* foram os mesmos do código próprio. Os valores utilizados para o caso da cavidade foram definidos a partir da referência [2] e são apresentados na tabela a seguir.

Reynolds	Número de nós	Velocidade mesoscópica	Resíduo entre mil iterações
1000	257	0.1	10^{-6}

Tabela 1: Valores utilizados para simulação do escoamento em cavidade com tampa deslizante

Sendo a velocidade mesoscópica a *Lattice Velocity* do *OpenLB* e a equação utilizada para o cálculo do resíduo.

$$Res = \frac{\sum \mid v_1 - v_0 \mid}{\sum \mid v_1 \mid} \tag{7}$$

Com v_1 representando a velocidade atual, v_0 a velocidade há mil passos no tempo e o somatório atuando em todos os nós. Para o erro, foi utilizado o L2, dado pela equação 8.

$$Erro = \sqrt{\frac{\sum (q_a - q_n)^2}{\sum q_a^2}} \tag{8}$$

Em que q_a representa o valor analítico ou de referência, q_n o valor numérico e o somatório atua sobre todos os valores de referência.

Para comparação dos dados obtidos, foram utilizados como referência valores da velocidade horizontal em x=0.5 e vertical em y=0.5. A figura 3 apresenta os perfis obtidos da velocidade em x em x=0.5 com o código próprio, com o *OpenLB* e os valores de referência [2].

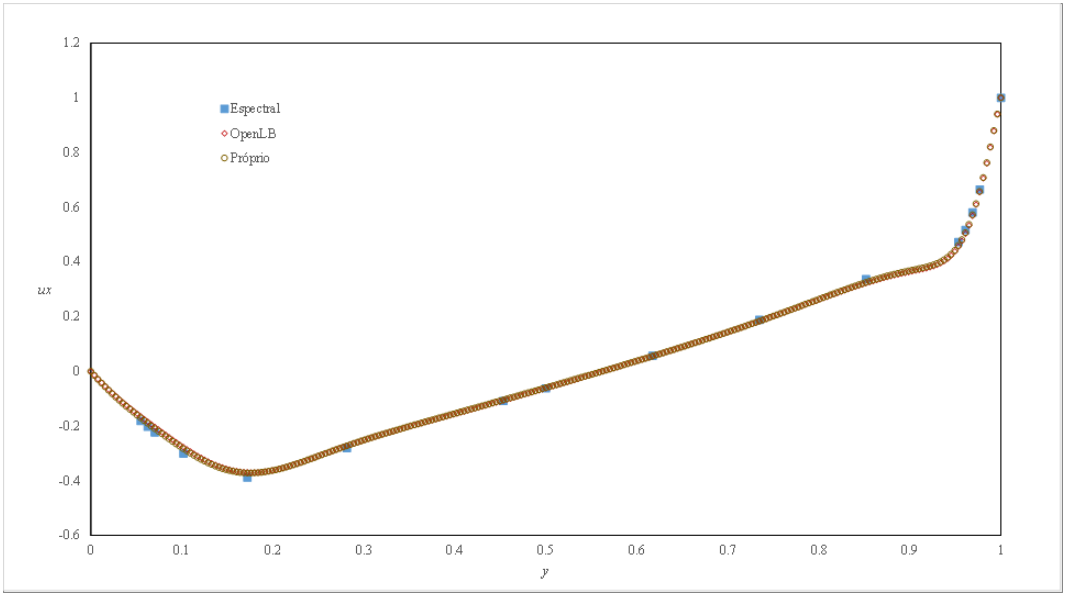

Figura 3: Perfil da velocidade em x em x=0.5 para cavidade

Com relação a referência [2], o erro L2, dado pela equação 8, dos valores obtidos pelo código próprio foi de $2.45 \cdot 10^{-2}$, já o erro obtido a partir dos valores fornecidos pelo *OpenLB* foi de $3.22 \cdot 10^{-2}$. A figura 4 apresenta os perfis da velocidade em y em y=0.5.

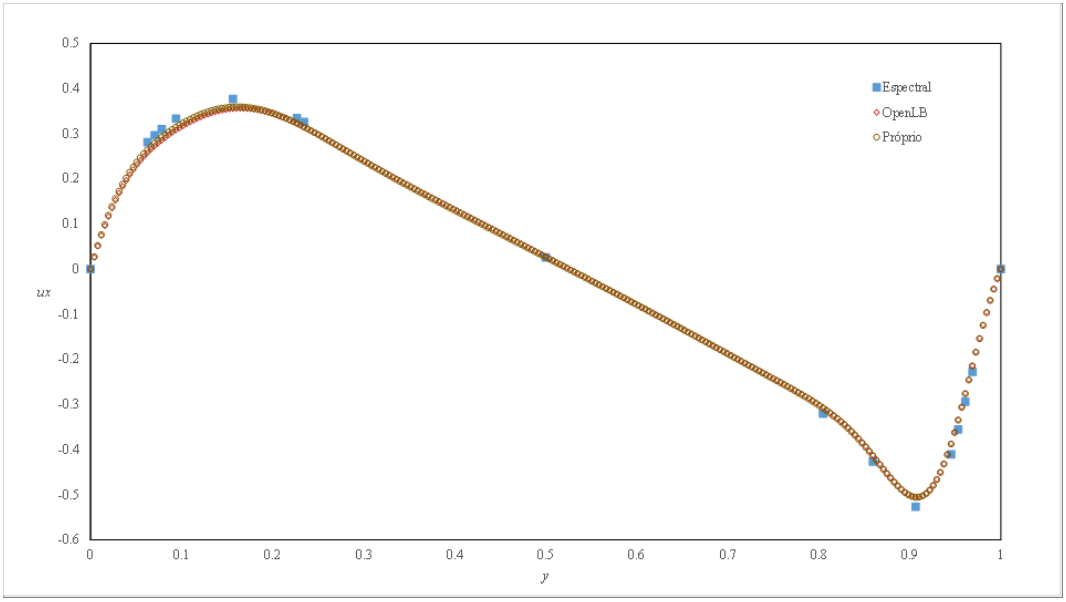

Figura 4: Perfil da velocidade em y em y=0.5 para cavidade

Com relação a referência, o erro, dado pela equação 8, apresentado pelos valores obtidos a partir do código próprio foi de $4.75 \cdot 10^{-2}$, já os valores obtidos a partir do *OpenLB* apresentaram um erro de $5.59 \cdot 10^{-2}$.

Todos os erros apresentaram valores aceitáveis, com o código próprio apresentando um erro pouco menor que o *OpenLB*. Portanto, ambas simulações apresentaram uma boa aproximação com relação à referência, para os dois perfis de velocidade. Já se tratando de desempenho, o *OpenLB* fez cerca de 31 MLUPS (*Million Lattice updates per second*), enquanto o código próprio obteve 11 MLUPS, o que provavelmente se dá pelo fato de algumas otimizações já serem implementadas pelo código aberto.

Para o código próprio, a porcentagem de tempo de CPU tomado por cada função com relação ao tempo total (fornecida pelo perfil de desempenho do *Visual Studio*) foi de 41.48% pela colisão, 26.75% pela propagação, 0.28% pela aplicação de condições de contorno e 29.56% pela atualização dos valores macroscópicos, com alocação, inicialização, cálculo de resíduo e demais funções ocupando a porcentagem restante. Neste segmento, pelo fato da colisão, propagação e a atualização dos valores macroscópicos serem totalmente paralelizáveis, visto que juntas tomam mais de 95% do tempo de CPU, é esperado um aumento considerável no número de MLUPS a partir da paralelização dessas tarefas.

5.2 Validação dos dados

A validação dos dados do código próprio foi realizada a partir do teste da ordem de erro de truncamento para ambos escoamentos, entre placas paralelas e de cavidade com tampa deslizante. A seguir serão apresentadas as validações dos dois casos.

5.2.1 Cavidade com tampa deslizante

A validação dos dados no escoamento em cavidade com tampa deslizante foi feita tomando como referência dois perfis, o primeiro da velocidade em x em x=0.5 e o segundo da velocidade em y em y=0.5. Pelo escoamento não possuir solução analítica, como referência foram utilizados os dados obtidos a partir de um método de maior precisão [2].

Para tal, foram feitas quatro simulações com velocidade mesoscópica e Reynolds constantes e dobrando o número de *lattice* de uma para outra, logo, para

manutenção do tempo físico o número de passos no tempo foi duplicado. Os valores da simulação são apresentados na tabela 2.

	Simulação 1	Simulação 2	Simulação 3	Simulação 4
Reynolds	1000	1000	1000	1000
Número de nós	65	129	257	513
Velocidade mesoscópica	0.1	0.1	0.1	0.1
Passos no tempo	59000	118000	236000	472000
Resíduo	$9.18 \cdot 10^{-6}$	$5.29 \cdot 10^{-6}$	$3.01 \cdot 10^{-6}$	$2.20 \cdot 10^{-6}$

Tabela 2: Valores das simulações do escoamento em cavidade com tampa deslizante

Devido ao alto valor de Reynolds, foi necessário utilizar o operador de colisão regularizado [7] para possibilitar as simulações com um baixo número de *lattice*. Caso contrário, o tempo de simulação seria muito alto, tendo em vista que não regularizando a colisão o valor mínimo de nós para atingir a convergência, fixando-se os outros valores, foi de cerca de 250 nós. Ou seja, a última simulação necessitaria de pelo menos 2000 nós e tomaria alguns dias para ser finalizada.

Os erros obtidos para velocidade em y com relação a referência são apresentados na tabela 3.

	Simulação 1	Simulação 2	Simulação 3	Simulação 4
Erro	$1.28 \cdot 10^{-1}$	$6.65 \cdot 10^{-2}$	$3.75 \cdot 10^{-2}$	$2.17 \cdot 10^{-2}$

Tabela 3: Erros da velocidade em y para simulação de escoamento em cavidade com tampa deslizante

Os erros para a velocidade em x com relação a referência são apresentados na tabela 4.

	Simulação 1	Simulação 2	Simulação 3	Simulação 4
Erro	$5.23 \cdot 10^{-2}$	$3.07 \cdot 10^{-2}$	$1.77 \cdot 10^{-2}$	$1.05 \cdot 10^{-2}$

Tabela 4: Erros da velocidade em x para simulação de escoamento em cavidade com tampa deslizante

Os altos valores dos erros com relação a referência podem ser explicados por alguns motivos. Dentre eles estão os pontos escolhidos serem os mais sensíveis para simulação, com maior gradiente de velocidade e a comparação com uma solução numérica, a qual possui erros que são propagados na comparação.

Pelo LBM com operador BGK e esquema de Zou-He possuir erro de truncamento de ordem dois [5], a curva de erros deve se aproximar de uma curva com expoente igual a menos dois. Os erros obtidos e a curva esperada para a velocidade em y são exibidos na figura 5.

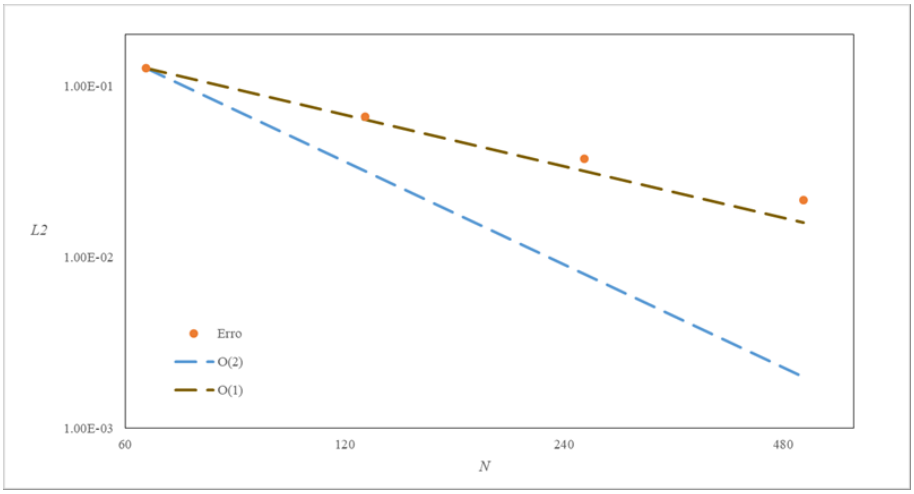

Figura 5: Erro da velocidade em y para simulação de cavidade com tampa deslizante em função do número de nós

Os erros obtidos e a curva esperada para a velocidade em x são exibidos na figura 6.

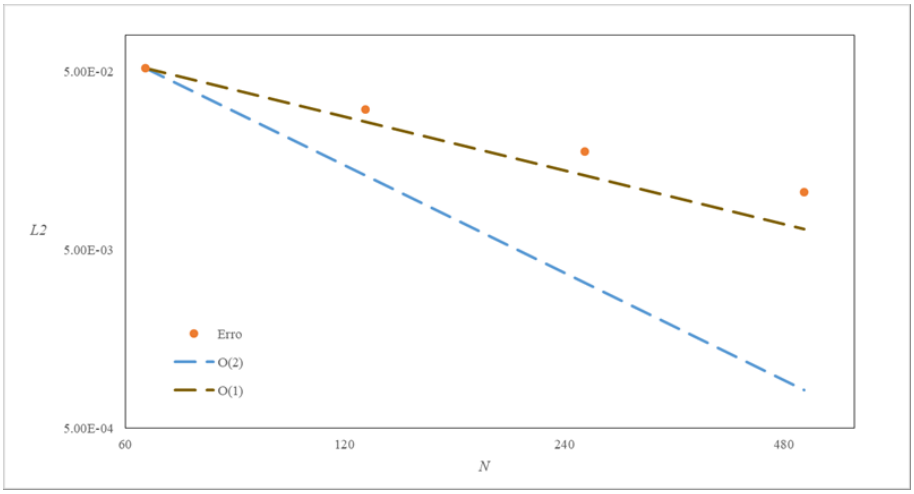

Figura 6: Erro da velocidade em x para simulação de cavidade com tampa deslizante em função do número de nós

A discrepância entre os erros obtidos e a curva esperada, tanto para a velocidade em x quanto para em y, se deve a comparação com valores numéricos, ou seja, não é possível afirmar que esses erros são falhas reais da simulação, visto que o valor de referência não é uma solução analítica.

5.2.2 Placas paralelas

A validação dos dados no escoamento entre placas paralelas foi feita a partir do perfil da velocidade em x em x=0.5, utilizando como referência a equação 9.

$$u = 6(y - y^2) \tag{9}$$

Sendo u a velocidade normalizada pela velocidade média e y o valor normalizado pela distância entre placas.

Foram realizadas quatro simulações visando a comparação dos erros obtidos a partir de cada uma. Foi dobrado o número de nós e dividido a velocidade mesoscópica por dois de uma simulação para outra, portanto, o número de passos foi quadruplicado para manutenção do tempo físico. Os valores são apresentados na tabela 5.

	Simulação 1	Simulação 2	Simulação 3	Simulação 4
Reynolds	10	10	10	10
Número de nós	17	33	65	129
Velocidade mesoscópica	0.1	0.05	0.025	0.0125
Passos no tempo	3000	12000	48000	192000
Resíduo	$8.35 \cdot 10^{-10}$	$3.93 \cdot 10^{-9}$	$5.01 \cdot 10^{-8}$	$2.90 \cdot 10^{-7}$

Tabela 5: Valores das simulações do escoamento entre placas paralelas

Os erros obtidos em cada solução com relação à equação 9 são apresentados na tabela abaixo.

	Simulação 1	Simulação 2	Simulação 3	Simulação 4
Erro	$3.78 \cdot 10^{-3}$	$9.75 \cdot 10^{-4}$	$2.52 \cdot 10^{-4}$	$6.41 \cdot 10^{-5}$

Tabela 6: Erros para simulação de escoamento entre placas paralelas

A figura 7 mostra os erros para o escoamento entre placas paralelas e a curva esperada para esses.

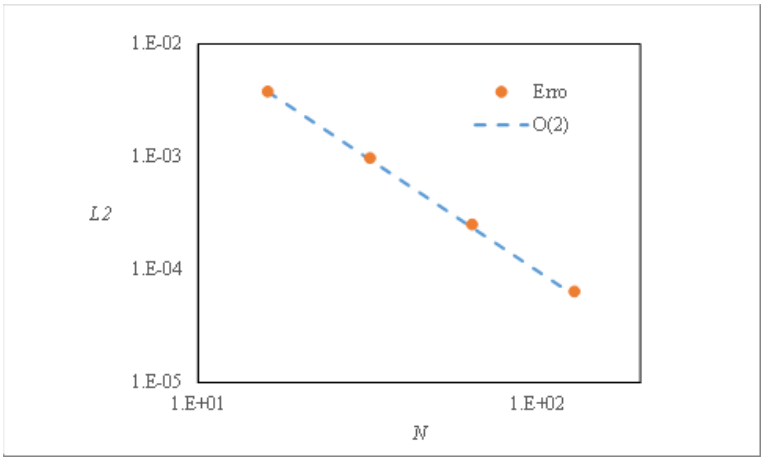

Figura 7: Erro para simulação entre placas paralelas em função do número de nós

Os valores se ajustaram muito bem à curva teórica, o que leva a crer que a ordem de erro de truncamento para o código próprio é a mesma esperada pela teoria.

6 Conclusão

A implementação de um código próprio para simulação de escoamentos utilizando o método de Lattice Boltzmann foi um sucesso, tendo em vista a comparação dos dados obtidos por meio desse com aqueles obtidos por meio do *OpenLB*. Porém, quanto ao desempenho, o código aberto ainda demonstra ser consideravelmente mais rápido que o código próprio.

Ademais, a validação dos dados a partir do teste da ordem de erro de truncamento obteve sucesso, alcançando o valor teórico esperado, fato que permite concluir sobre a validade dos valores fornecidos pelo código. Por fim, a partir da paralelização das funções e de otimizações na memória desse, é esperado um grande aumento no número de MLUPS e consequente diminuição do tempo computacional de simulação.

Referências

[1] Visual Studio Documentation. `https://docs.microsoft.com/en-us/visualstudio`. Acesso em: 14/11/2018.

[2] O. Botella and R. Peyret. Benchmark spectral results on the lid-driven cavity flow. *Computers Fluids*, 27(4):421 – 433, 1998.

[3] M. Januszewski and M. Kostur. Sailfish: A flexible multi-gpu implementation of the lattice boltzmann method. *Computer Physics Communications*, 185(9):2350 – 2368, 2014.

[4] M. J. Krause. *OpenLB User Guide*. OpenLB, 2018.

[5] T. Kruger, H. Kusumaatmaja, A. Kuzmin, O. Shardt, G. Silva, and E. M. Viggen. *The Lattice Boltzmann method: Principles and practice*. Springer, The address, 1 edition, 2017.

[6] F. Kuznik, C. Obrecht, G. Rusaouen, and J.-J. Roux. Lbm based flow simulation using gpu computing processor. *Computers Mathematics with Applications*, 59(7):2380 – 2392, 2010. Mesoscopic Methods in Engineering and Science.

[7] J. Latt and B. Chopard. Lattice boltzmann method with regularized pre-collision distribution functions. *Mathematics and Computers in Simulation*, 72(2):165 – 168, 2006. Discrete Simulation of Fluid Dynamics in Complex Systems.

[8] Q. Zou and X. He. On pressure and velocity boundary conditions for the lattice Boltzmann BGK model. *Phys. Fluids*, 9(6):1592–1598, 6 1997.