Proceeding Series of the Brazilian Society of Computational and Applied Mathematics, v. 10, n. 1, 2023.

Trabalho apresentado no XLII CNMAC, Universidade Federal de Mato Grosso do Sul - Bonito - MS, 2023

Proceeding Series of the Brazilian Society of Computational and Applied Mathematics

Otimização de um Algoritmo para Solução do Problema da Cavidade com Tampa Móvel

Guilherme C. Tomiasi[1]
FCT - Unesp, Presidente Prudente, SP
Rafael L. Sterza[2]
ICMC/USP, São Carlos, SP
Analice C. Brandi[3]
DMC/FCT - Unesp, Presidente Prudente, SP

O problema da cavidade com tampa móvel descreve o comportamento de um fluido dentro de uma cavidade, em que uma tampa - adjacente ao fluido contido na mesma - é movimentada no instante t_0, assumindo uma velocidade $U > 0$. A interação entre a tampa e o fluido adjacente à mesma acaba por causar um escoamento. O escoamento gerado pelo movimento da tampa pode ser descrito utilizando da função corrente-vorticidade e suas condições iniciais de contorno:

$$\frac{D\omega}{Dt} = \frac{1}{Re}\nabla^2\omega, \tag{1}$$

$$\nabla^2\psi = -\omega. \tag{2}$$

$$\begin{aligned} \mathbf{u}(x,y,0) &= \mathbf{0} \quad \text{em} \quad \Omega, \qquad \mathbf{u}(x,y,t) = \mathbf{0} \quad \text{em} \quad \partial\Omega_1 \times [0,t], \\ u(x,y,t) &= U \quad \text{e} \quad v(x,y,t) = 0 \quad \text{em} \quad \partial\Omega_2 \times [0,t]. \end{aligned} \tag{3}$$

onde $\mathbf{u} = (u, v)$ é o vetor velocidade, com u e v representando suas componentes em direções perpendiculares.

Feita essa formulação, o problema pode ser solucionado usando métodos de diferenças finitas de segunda e quarta ordem. Originalmente, fora solucionado utilizando a linguagem de programação MATLAB, comparando a solução encontrada com resultados presentes na literatura anteriormente [1]. Os algoritmos de diferenças finitas de segunda e quarta ordem, escritos originalmente em MATLAB [2], foram adaptados para utilizar a linguagem Julia [3], uma linguagem de programação de código aberto destinada à aplicações de Computação Científica. Após adaptar o algoritmo, os resultados obtidos em cada linguagem foram comparados para garantir que a adaptação fora realizada corretamente. A partir disso, foram feitas otimizações visando reduzir o total de memória alocada, melhorar a legibilidade do código tal como sua modularização para permitir o teste de porções menores do código, o que permite realizar micro-otimizações.[4]

Tendo em vista que a maior parte do tempo de execução ocorre durante a resolução de um sistema linear esparso - que deve ser resolvido a cada iteração do método - qualquer otimização deve visar a aceleração desse sistema. Utilizando o pacote "Debugger.jl" e a macro "@enter", foi possível, através do acesso ao código-fonte da linguagem, verificar quais passos são realizados para resolver o sistema linear esparso nos dois algoritmos utilizados.

[1]gtomiasi@gmail.com
[2]rafael.sterza@usp.br
[3]analice.brandi@unesp.br

[4]O repositório que contém as versões comparadas nesse resumo pode ser acessado via https://github.com/GuiCT/accelerating-ldcflow-solution

Notou-se que para cada execução do método utilizado para resolver o sistema linear esparso, é realizada uma fatoração da matriz esparsa. No caso do método de segunda ordem, é realizada uma decomposição LU, e no caso do método de quarta ordem uma fatoração LDL^t. A matriz esparsa é constante durante toda a execução dos métodos, então é possível otimizar o algoritmo ao armazenar o resultado dessa fatoração, visto que ela também será constante. Essa medida conferiu uma melhora substancial no tempo de execução de ambos os métodos. Essa mesma otimização pode ser feita na Linguagem MATLAB utilizando a função "decomposition" e a opção "auto". Dessa forma, a linguagem escolhe a fatoração mais adequada a ser utilizada.

As Tabelas 1 e 2 apresentam os tempos de execução anotados para diferentes tamanhos de malha e números de Reynolds. O passo de integração temporal em todos os casos é $\delta_t = 0.001$ e o tempo de execução é dado em segundos.

Tabela 1: Tempos de execução (Diferenças finitas de segunda ordem).

n	Re	Original (MATLAB)	Otimizado (Julia)	Otimizado (MATLAB)
64	100	36.975384	3.305687	4.063123
64	1000	158.725786	13.060023	15.068767
128	100	161.043162	14.224780	18.008023
128	1000	663.722948	58.782705	72.816757

Tabela 2: Tempos de execução (Diferenças finitas de quarta ordem)

n	Re	Original (MATLAB)	Otimizado (Julia)	Otimizado (MATLAB)
64	100	26.834905	2.060433	5.497554
64	1000	105.583558	6.820514	21.445892
128	100	110.740466	9.820449	24.601200
128	1000	446.860264	39.783770	89.705491

Comparando os tempos de execução anotados, é patente a melhora obtida a partir das otimizações citadas anteriormente. O método original em MATLAB também apresentou uma melhora significativa a partir do armazenamento das fatorações, no entanto, apresentou um tempo de execução pior quando comparado à adaptações feitas utilizando a linguagem de programação Julia, devido às micro-otimizações realizadas em porções específicas do código, separadas em funções.

Agradecimentos

Agradecemos ao Programa Institucional de Bolsas de Iniciação Cientifica (PIBIC) pelo auxílio financeiro no desenvolvimento deste trabalho.

Referências

[1] U. Ghia, K. N. Ghia e C. T. Shin. "High-Re solutions for incompressible flow using the Navier-Stokes equations and a multigrid method". Em: **Journal of Computational Physics** 48.3 (1982), pp. 387–411. DOI: 10.1016/0021-9991(82)90058-4.

[2] R. L. Sterza, B. L. Carreira e A. C. Brandi. "Solução numérica da equação de Poisson no problema da cavidade com tampa móvel". Em: **CQD - Revista Eletrônica Paulista de Matemática** 17 (fev. de 2020), pp. 227–238. DOI: 10.21167/cqdvol17ermac202023169664rlsblcacb227238.

[3] Jeff Bezanson, Alan Edelman, Stefan Karpinski e Viral B Shah. "Julia: A fresh approach to numerical computing". Em: **SIAM review** 59.1 (2017), pp. 65–98. URL: https://doi.org/10.1137/141000671.