Paralelização em um Ambiente de Memória Distribuída de um Simulador da Formação de Edemas no Coração

Lara Turetta Pompei[1], **Ruy Freitas Reis**[1], **Rodrigo Weber dos Santos**[1] **e Marcelo Lobosco**[1]

[1]FISIOCOMP
Laboratório de Fisiologia Computacional e Computação de Alto Desempenho,
Universidade Federal de Juiz de Fora (UFJF)

pompei.lara@engenharia.ufjf.br,{ruyfreitas,marcelo.lobosco}@ice.ufjf.br

Abstract. *The purposes of this work are the paralelization of an algorithm that simulates an edema formation in the heart, as well as to analyse its performance. The results show that the parallel version led to a reduction in the execution time up to 14 times compared to its sequential version.*

Resumo. *O objetivo deste trabalho é paralelizar um algoritmo que simula a formação de um edema no coração, usando para isso MPI, bem como analisar o seu desempenho em um cluster de computadores. Os resultados mostram que a versão paralela conseguiu reduzir o tempo de execução sequencial em até 14 vezes.*

1. Introdução

Segundo a OMS, cerca de 17,5 milhões de pessoas morrem vítimas de doenças cardiovasculares a cada ano no mundo, o que corresponde a aproximadamente 0,55 mortes por segundo, números que fazem destas doenças as principais causas de morte no mundo [OMS 2014]. O grande número de mortes decorrentes de doenças cardiovasculares impulsiona muita pesquisa sobre o tema, mas ainda assim inúmeros de seus aspectos permanecem desconhecidos.

Uma ferramenta que pode ajudar os cientistas a melhor compreender as causas das doenças cardiovasculares e possíveis estratégias para seus tratamentos é a modelagem computacional. Os modelos matemático-computacionais podem ser criados para representar aspectos chave do funcionamento do coração, do surgimento de uma determinada doença ou mesmo da ação de uma droga para o seu tratamento. A partir destes modelos, simulações podem ser feitas para melhor compreender os fenômenos modelados.

Uma simulação pode, no entanto, demandar um grande poder computacional para sua execução, levando horas, dias ou mesmo meses para gerar resultados, o que pode na prática inviabilizar seu uso. Neste cenário, o desenvolvimento de uma versão paralela do código é essencial para atender sua demanda computacional, reduzindo seu tempo de execução e tornando assim seu uso viável. Tal versão paralela faz uso simultâneo de várias unidades de processamento para reduzir seu tempo de execução. Por exemplo, uma das estratégias empregadas para o desenvolvimento de uma aplicação paralela é o chamado paralelismo de dados, em que um mesmo conjunto de instruções deve ser executado para um grande conjunto de dados [Pacheco 2011]. Os dados podem então ser divididos em

conjuntos menores, que por sua vez são operados simultaneamente por distintas unidades de processamento, reduzindo assim o tempo de execução total.

Neste trabalho o paralelismo de dados é empregado para reduzir o tempo de execução de um simulador da formação de edemas em um tecido cardíaco [Reis 2018]. A versão paralela do simulador é implementada com o uso da biblioteca MPI [Pacheco 1996], sendo executada em um ambiente de memória distribuída. Os tempos de execução são então coletados e analisados. Os resultados mostram que a versão paralela conseguiu reduzir o tempo de execução da versão sequencial do simulador em até 14 vezes.

Este artigo está organizado da seguinte forma. A Seção 2 apresenta o referencial teórico necessário para o entendimento deste trabalho, destacando aspectos biológicos da formação de uma edema e a implementação serial do simulador; o método aplicado para a paralelização do simulador é apresentado na Seção 3. Os resultados são apresentados e discutidos na Seção 4. Por fim, a última seção apresenta as conclusões e perspectivas de trabalhos futuros.

2. Referencial Teórico

Esta seção apresenta brevemente a teoria necessária para a compreensão de uma das contribuições deste trabalho, a paralelização de um simulador da formação de um edema no coração. Para tanto, serão apresentados os aspectos biológicos relativos à formação de um edema no tecido, bem como o modelo matemático que foi implementado na versão sequencial do código que foi paralelizado.

2.1. Aspectos Biológicos da Formação de um Edema

A inflamação é uma reação importante do organismo a qualquer evento que cause uma lesão em suas células ou tecidos. É também uma resposta imunológica, podendo envolver diferentes células dependendo do local da lesão.

Um aspecto essencial das células imunológicas é sua mobilidade intersticial, que consiste na habilidade de se deslocar facilmente através das células do tecido até o local infectado. Quando patógenos entram no organismo encontram células e moléculas do sistema imunológico, como proteínas do sistema complemento e macrófagos, que desenvolvem uma resposta imunológica imediata a presença dos mesmos [Sompayrac 2008]. Os macrófagos são responsáveis por fagocitar os patógenos e produzir proteínas chamadas de citocinas, responsáveis por sinalizar às outras células imunes a presença de uma anomalia [Sompayrac 2008].

As citocinas podem desencadear uma série de eventos que levam a aumento da permeabilidade do endotélio, facilitando outras células do sistema imunológico a saírem da corrente sanguínea e chegarem até o local infectado. No entanto, o aumento da permeabilidade do endotélio permite, também, a passagem de plasma até o tecido e o excesso destes fluidos no interstício causa um edema extracelular. Portanto um edema ocorre quando um volume excessivo de fluido acumula no tecido, levando ao inchaço na região [Guyton *et al.* 2006]. Edemas podem ocorrer em vários tipos de tecidos, como o cardíaco. A formação de um edema no coração pode levar à morte.

2.2. Modelagem Matemático-Computacional da Formação de um Edema no Coração

A versão sequencial do simulador da formação de edemas no tecido cardíaco utiliza um conjunto de três Equações Diferenciais Parciais (EDPs) que descrever tanto a dinâmica de um agente patogênico (bactéria) invasor, quanto do sistema de defesa do organismo (neutrófilos) [Reis 2018, Reis *et al.* 2016a]. Para que os neutrófilos possam deixar a corrente sanguínea e entrar no tecido, é necessário o aumento da permeabilidade do endotélio. Por sua vez o aumento de permeabilidade dos capilares leva ao também aumento do fluxo de plasma que sai dos vasos sanguíneos para o tecido, aumentando seu volume e consequentemente também aumentando a pressão do fluido dentro do tecido. Parte deste plasma é coletado por vasos linfáticos. Assim, ocorre o aumento do fluxo linfático, que por sua vez leva a um $feedback$ negativo do fluxo para o tecido, em uma tentativa de reequilibrar o organismo [Scallan *et al.* 2010]. Matematicamente, o fluxo plasmático é modelado como um fluido único em um meio poroso. Bactéria e neutrófilos são considerados misturados homogeneamente no fluido intersticial. Considera-se ainda no modelo que as bactérias podem proliferar e difundir no tecido, que o recrutamento de neutrófilos é influenciado pela quimiotaxia, e que bactérias podem ser mortas por neutrófilos. As descrições completas do modelo matemático utilizados na simulação podem ser encontradas nas referências [Reis *et al.* 2016b].

O crescimento da população de bactérias no tecido foi descrita através da seguinte equação:

$$\begin{cases} \dfrac{\partial \phi C_b}{\partial t} = \nabla \cdot (D_b \nabla C_b) - r_b + q_b & \text{em } \Omega \times I, \\ D_b \nabla C_b \cdot \mathbf{n} = f_b & \text{em } \partial\Omega \times I, \\ C_b\left(\cdot, 0\right) = C_{b0} & \text{em } \Omega, \end{cases} \tag{1}$$

onde $\Omega \subset \mathbb{R}^2$ e $I = (0, t_f] \subset \mathbb{R}^+$ é o intervalo de tempo, $C_b : \Omega \times I \to \mathbb{R}^+$ é a concentração de bactérias no interstício, ϕ representa a porosidade do meio, D_b representa o coeficiente de difusão da bactéria no interstício, q_b é o termo fonte, o qual representa a dinâmica de crescimento da população de bactérias, e r_b representa o termo de sumidouro, ou seja, a dinâmica de morte das bactérias causadas pela interação neutrófilo-bactéria. A dinâmica de crescimento da bactéria é dada por $q_b = c_b C_b$, onde c_b é a taxa de crescimento da bactéria no interstício. Finalmente, a dinâmica de morte da bactéria r_b é dada por: $r_b = \lambda_{nb} C_n C_b$, onde λ_{nb} é a taxa de morte de bactéria ao ser fagocitada por neutrófilos e C_n é a concentração de neutrófilos no fluido intersticial.

A dinâmica do neutrófilo no tecido é descrita pela seguinte equação:

$$\begin{cases} \dfrac{\partial \phi C_n}{\partial t} = \nabla \cdot (D_n \nabla C_n - \chi_{nb} C_n \nabla C_b) - r_n + q_n & \text{em } \Omega \times I, \\ (D_n \nabla (C_n - \chi_{nb} C_n \nabla C_n) \cdot \mathbf{n} = f_n & \text{em } \partial\Omega \times I \\ C_n\left(\cdot, 0\right) = C_{n0} & \text{em } \Omega, \end{cases} \tag{2}$$

onde $\Omega \subset \mathbb{R}^2$ e $I = (0, t_f] \subset \mathbb{R}^+$ definem o intervalo de tempo, $C_n : \Omega \times I \to \mathbb{R}^+$ representa a concentração do neutrófilo no interstício, D_n é o coeficiente de difusão do neutrófilo no interstício, χ_{nb} representa a influência da quimiotaxia no tecido, q_n é o termo fonte que representa o transporte de neutrófilos através da parede dos capilares, e por fim r_n é o termo de sumidouro, que representa a dinâmica de morte dos neutrófilos causada

pela interação neutrófilo-bactéria e pela diminuição natural da população. A dinâmica de crescimento do neutrófilo é modelada como um transporte através dos vasos sanguíneos, ou seja, $q_n = \gamma_n C_b(C_{n,max} - C_n)$, onde $C_{n,max}$ é a concentração de neutrófilos na corrente sanguínea e γ_n é a permeabilidade do capilar ao neutrófilo. Finalmente, a dinâmica de morte dos neutrófilos é descrita pela equação: $r_n = \lambda_{bn} C_n C_b + \mu_n C_n$, onde λ_{bn} é a taxa de morte de neutrófilo devido a sua interação com as bactérias e μ_n representa o decaimento natural de neutrófilos.

O último conjunto de equações representa a pressão intersticial do fluido no tecido, considerando este isotrópico e homogêneo. Matematicamente a pressão P é dada pela equação:

$$\begin{cases} \nabla \cdot \dfrac{K}{\mu} \nabla P = q_c + q_l & \text{em } \Omega, \\ \alpha P + \beta \nabla P \cdot \mathbf{n} = f_p & \text{em } \partial\Omega, \end{cases} \tag{3}$$

onde $\Omega \subset \mathbb{R}^n$ e $P : \Omega \to \mathbb{R}$, K é a permeabilidade do interstício, e μ a viscosidade da fase fluida. Os termos q_c e q_l modelam a influência dos capilares sanguíneos e linfáticos, respectivamente. A influência dos capilares sanguíneos é dada por:

$$q_c(P) = c_f(P_c - P - \sigma(\pi_c - \pi_i)), \tag{4}$$

onde $c_f = L_p(S/V)$ é o coeficiente de filtragem, $L_p = L_{p0}(1 + c_{bp} C_b)$ é a permeabilidade hidráulica da parede microvascular, e (S/V) é área superficial do vaso por unidade de volume; P e P_c são a pressão do fluido intersticial e a pressão capilar, respectivamente; π_c e π_i são a pressão oncótica do capilar e interstício, respectivamente; $\sigma = \frac{\sigma_0}{(1+c_{br}C_b)}$ é o coeficiente de reflexão para as proteínas do plasma.

A influência dos terminais linfáticos é modelada pela seguinte relação:

$$\int_V q_l(P) dV = -\left(q_0 \left(1 + \frac{V_{max}(P - P_0)^n}{K_m^n + (P - P_0)^n} \right) \right), \tag{5}$$

onde q_0 é o fluxo linfático normal, P_0 é a pressão não infeccionada do fluido intersticial. Além disso, V_{max} é o fluxo linfático máximo, K_m é a meia vida, *i.e.* o valor de P no qual o fluxo linfático corresponde a $\frac{V_{max}}{2}$, é n é o coeficiente de Hill [Keener and Sneyd 1998].

Os pontos discretizados do tecido são representados computacionalmente por uma malha bidimensional de valores em ponto-flutuante de precisão dupla. O método numérico usado para a implementação computacional do modelo matemático é o Método dos Volumes Finitos. O método de Jacobi foi utilizado para resolver os sistemas lineares resultantes, e o método de Euler explícito é utilizado para a discretização do tempo. A computação do termo convectivo é uma parte complexa na resolução das EDPs. A implementação utiliza o esquema conhecido como *First-Order Upwind* para garantir a solução estável do termo convectivo [Reis 2018].

3. Método

A paralelização do código do simulador foi feita para uma arquitetura de memória distribuída, sendo por esse motivo escolhida a biblioteca MPI. Após identificada a parte do

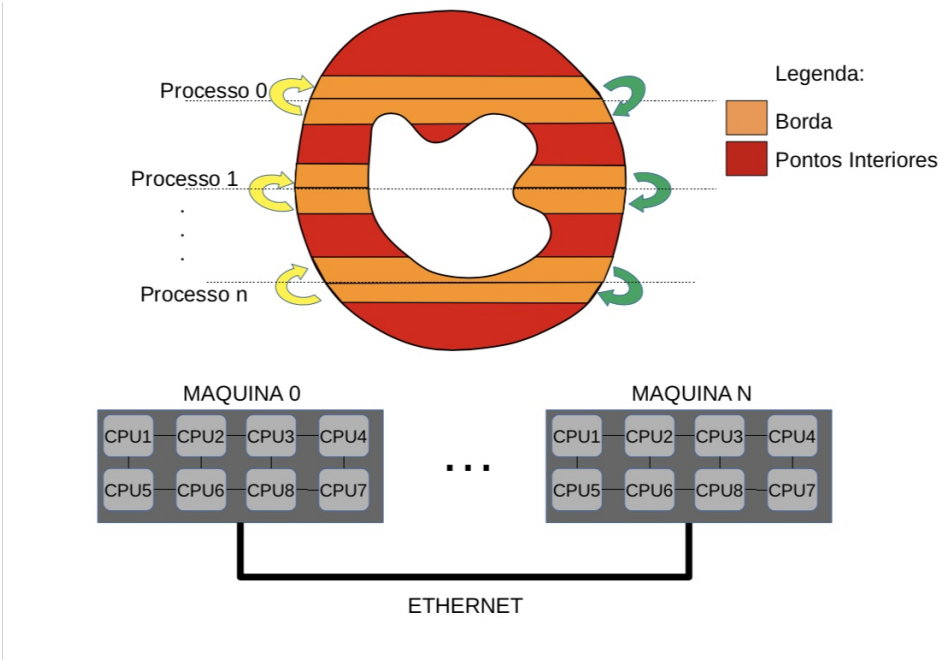

Figura 1. Divisão de trabalho entre os processos. O domínio representa um corte do eixo curto do coração. Os pontos que compõe o domínio são divididos entre os processos.

código que possui o maior tempo de execução, responsável pelo cálculo da dinâmica dos neutrófilos e das bactérias, verificou-se a viabilidade de empregar paralelismo de dados para reduzir o seu tempo de computação, uma vez que, a cada passo de tempo, uma mesma sequência de operações é calculada para cada ponto do domínio.

Dessa forma, deve-se dividir a malha que representa todo o tecido entre todos os processadores disponíveis, conforme ilustrado na Fig. 1. Por ser uma aplicação com características *CPU-bound*, foi criado um processo MPI para cada processador disponível.

Ao dividir a malha entre os processadores introduz-se, contudo, um problema que não existia na versão sequencial: para calcular as novas populações de neutrófilos e bactérias para um ponto do domínio em um dado intervalo de tempo t, o método numérico exige o acesso aos valores das populações contidas em seus pontos vizinhos no intervalo $t - 1$. Com a divisão dos dados entre os processadores, os pontos vizinhos agora podem estar localizados em outras máquinas. O conjunto dos pontos vizinhos necessários para os cálculos, chamados de bordas, devem ser trocados entre máquinas a cada passo de tempo para que a computação possa ser realizada corretamente. Introduz-se, assim, um custo de comunicação a cada passo de tempo, conforme ilustrado pelo Algoritmo 1 e indicado na Fig. 1 através das setas.

O Algoritmo 1 ilustra uma versão simplificada do código real, identificando apenas as trocas de mensagens. A variável k recebe o total de linhas a ser computada por cada processo, caso a divisão de linhas por processos disponíveis não seja exata, utilizamos a variável resto para armazenar as linhas restantes, as quais são atribuídas para um dos processos, para que todas as linhas sejam computadas.

Pode-se fazer a divisão da malha entre os processos de três formas distintas: a)

Algoritmo 1: Pseudocódigo da versão paralela do código.

```
1  MPI_Init (NULL, NULL);
2  MPI_Comm_rank (MPI_COMM_WORLD, &r);
3  MPI_Comm_size(MPI_COMM_WORLD,&n);
4  ...
5  k = nx/n; resto = nx%n;
6  if rank == 0 then
7  |  xi = 0; xf = k + resto;
8  end if
9  else
10 |  xi = (k*rank)+resto; xf = (k*rank)+resto;
11 end if
12 for t = 0; t < nt; t++ do
13 |  for x = ki; x < xf; x++ do
14 |  |  for y = 0; y < ny; y++ do
15 |  |  |  Computa bactérias e neutrófilos para o ponto x,y
16 |  |  end for
17 |  end for
18 |  MPI_Sendrecv para trocar minhas bordas com meus vizinhos
19 end for
```

dividir as colunas entre os processos, b) dividir as linhas entre os processos ou c) dividir a matriz em submatrizes a serem computadas simultaneamente. Inicialmente optou-se por dividir a malha em linhas (Fig. 1) a fim de se reduzir o número de mensagem que precisam ser trocada entre os processos, se comparado com a divisão em submatrizes. Quando a divisão por linhas é comparada com a divisão por colunas, o número de mensagens trocadas é igual pelo fato das matrizes utilizadas nos testes ser quadrada. Porém pode existir uma pequena diferença no tempo de comunicação devido a forma como os dados são armazenados na memória. Tendo em vista que o código sequencial foi originalmente implementado em C, e que nesta linguagem matrizes são armazenadas na memória por linhas, os acessos aos dados para montagem de um pacote de comunicação por parte do MPI tendem a ser mais rápidos pelo fato dos dados estarem localizados em endereços contíguos de memória.

4. Resultados

Nesta seção, discutem-se os resultados obtidos para execução da versão paralela do simulador usando diferentes quantidades de computadores, malhas e de núcleos.

4.1. Ambiente Experimental

Os experimentos foram executados em *cluster* composto por oito computadores, cujo acesso exclusivo é feito através de uma fila de submissão de *jobs* (OGE). Cada computador possui dois processadores Intel Xeon E5620 de 2.40 GHz, cada um com quatro núcleos, totalizando assim 8 núcleos de processamento por computador. Cada núcleo possui 32KB de *cache* L1 de dados, 32 KB de *cache* L1 de instruções, e 256KB de *cache* L2, esta compartilhada por instruções e dados. Cada processador possui ainda uma *cache* L3

de 12MB, também usada para armazenar dados e instruções, e compartilhada entre os quatro núcleos do processador. Apesar deste processador possuir suporte para a tecnologia *hyper-threading*, esta foi desabilitada na BIOS. Os computadores executam o SO Linux com *kernel* na versão $3.10.0$. O compilador gcc na sua versão $4.8.5$ foi usado para compilar os programas e a biblioteca *mpich* versão 3.2 foi usada para estabelecer a comunicação entre os processos. O tempo foi computado através do aplicativo *time*, disponível no SO. Para fins de cálculo do tempo de execução, foi considerada a média aritmética simples de 8 execuções, sendo o maior desvio-padrão observado igual a $0,8\%$. A simulação representa um tecido de dimensões 1cm $\times$ 1cm, sendo discretizado, primeiro, com 400×400 pontos e, em seguida, com 800×800 para testar sua escalabilidade. São usados 100.000 passos de tempo na simulação com a malha discretizada em 800×800 e 10.000 passos para a malha discretizada em 400×400. As condições iniciais, parâmetros e condições de contorno utilizadas são as mesmas para ambas as malhas e idênticas aos valores usados em um artigo anterior[Reis *et al.* 2016a]. O tempo de execução sequencial da aplicação é de 5.201s (cerca de 1h27) para a malha de 800×800, enquanto para a malha de 400×400 é de 297s (4min57s).

4.2. Resultados Numéricos

As figuras 2, 3 e 4 apresentam o resultado da simulação na metade dos passos de simulação, ilustrando respectivamente a concentração de bactérias e de neutrófilos, bem como a pressão do fluido intersticial.

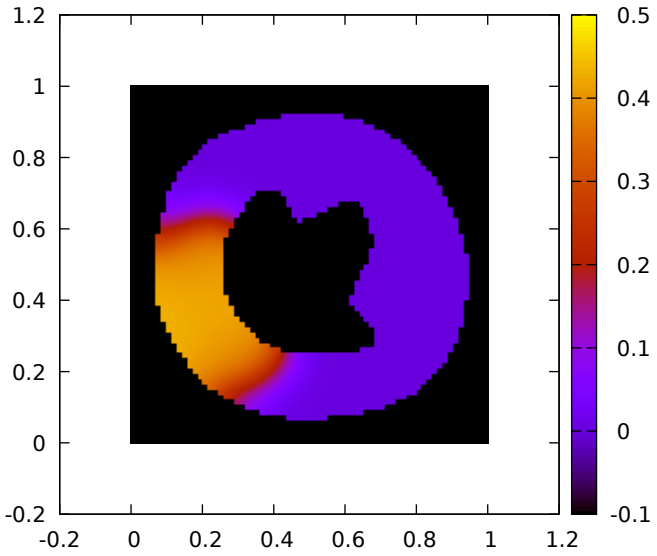

Figura 2. Concentração de bactérias (escala em $células/cm^2$) na metade dos passos de simulação.

4.3. Resultados Computacionais

A Figura 5 apresenta o *speedup* da aplicação, quando executada com 2, 4, 8, 16, 32 e 64 núcleos. Considera-se sempre a execução em uma configuração com o menor número de máquinas possível. Como estão disponíveis apenas 8 núcleos por máquina, as configuração com 2, 4 e 8 núcleos foram executadas em uma mesma máquina, 16

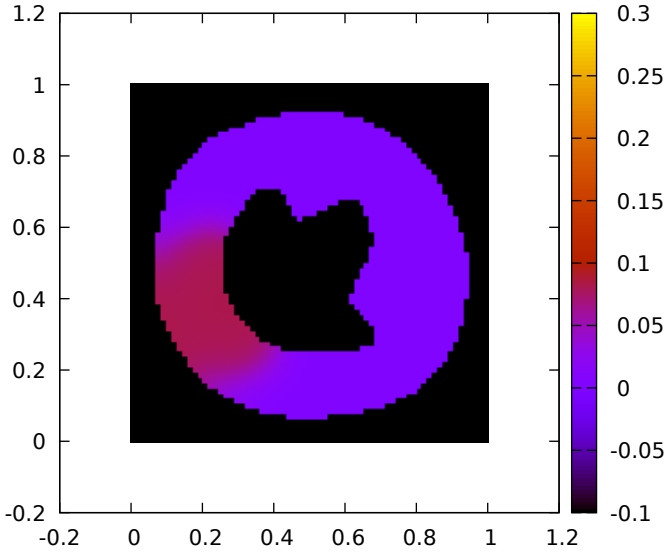

Figura 3. Concentração de neutrófilos (escala em $células/cm^2$) na metade dos passos de simulação.

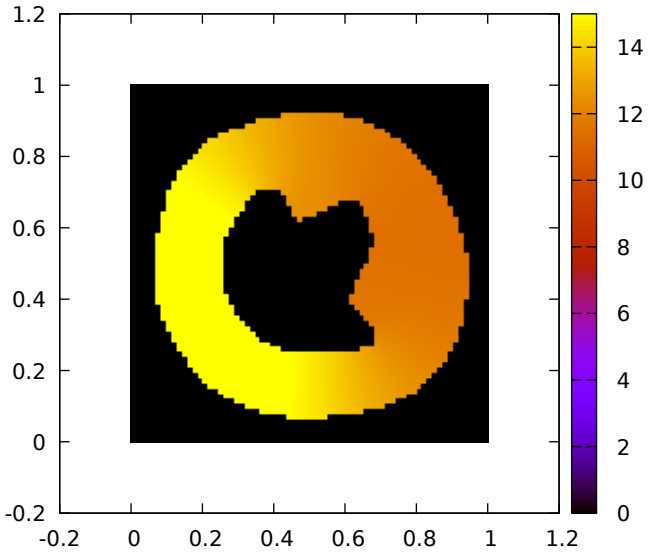

Figura 4. Pressão do fluido intersticial (escala em mmHg) ao longo do tecido na metade dos passos de simulação.

núcleos foram executadas com 2 máquinas, 32 núcleos com 4 máquinas e 64 núcleos com 8 máquinas.

Pode-se observar que a eficiência foi baixa: na malha de 400×400, iniciou-se com 70% para dois núcleos e terminou com 5% para 64 núcleos. Já na malha de 800×800, a eficiência do programa começa em 92% para dois núcleos, e chegou a apenas 17% para 64 núcleos, como apresentados na Tabela 1. Adicionalmente os resultados apontam que o programa é fracamente escalável, uma vez que a eficiência mantêm-se aproximadamente constante ao aumentar o número de processos na mesma proporção que o aumento no tamanho da malha (400×400, com 160.000 elementos, e 800×800, com

640.000 elementos, ou seja, 4 vezes). Por exemplo, ao comparar a eficiência na malha de 400×400 com 8 núcleos com a eficiência na malha de 800×800 com 32 núcleos, verifica-se que os valores são aproximadamente iguais (0,28 e 0,27, respectivamente).

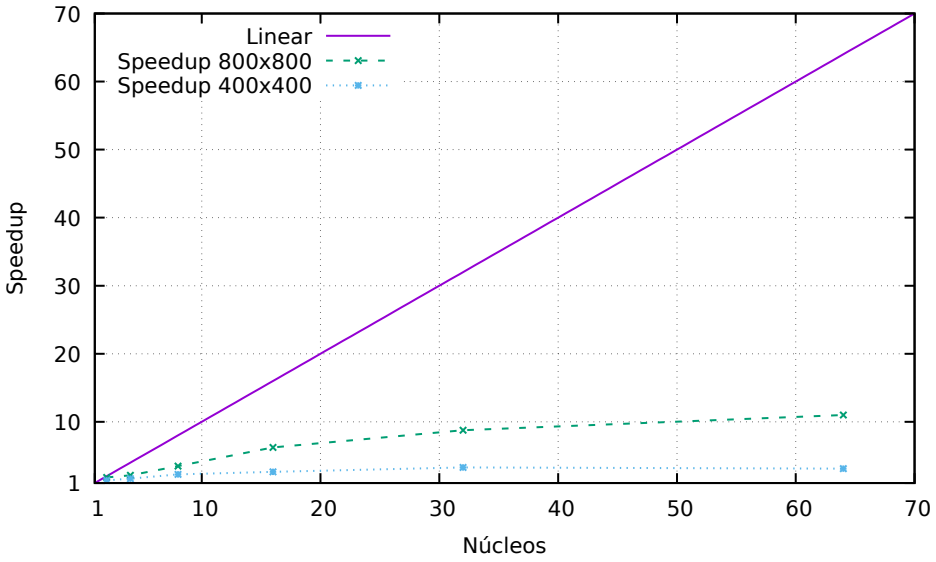

Figura 5. *Speedup*, considerando sempre a execução em uma configuração com o menor número de máquinas.

Tabela 1. Aceleração e eficiência calculados para as malhas de 400×400 e 800×800. Considerou-se para os cálculos do *speedup* e da eficiência a configuração com o menor número de máquinas.

Quantidade de Núcleos	400×400		800×800	
	Speedup	Eficiência	*Speedup*	Eficiência
2	1,4	0,70	1,8	0,92
4	1,7	0,41	2,1	0,54
8	2,3	0,28	3,5	0,44
16	2,7	0,17	6,2	0,39
32	3,3	0,10	8,8	0,27
64	3,1	0,05	11,0	0,17

O melhor tempo de execução paralelo para a malha de 400×400 foi obtido em uma configuração que utiliza 32 núcleos: 62s, representando um *speedup* de 4,8 vezes, ou seja, uma redução de cerca de 80% do tempo de execução sequencial. Na malha de 800×800, a melhor configuração foi obtida com 43 núcleos: 366s, o que representa um *speedup* de 14 vezes, ou seja, uma melhoria de 93% do tempo de execução sequencial.

Um experimento final foi realizado: forçou-se a alocação dos processos MPI em um número maior de máquinas, através de comandos específicos para a fila de execução, para que se pudesse investigar se haveria impacto no tempo total de execução. A princípio se esperaria um aumento no tempo de execução em virtude de um possível aumento no tempo de comunicação. Isso ocorreria em virtude da comunicação se dar por padrão, quando os processos estão localizados na mesma máquina, por memória compartilhada

Tabela 2. Tempo de execução (em s) na malha discretizada em 400×400 pontos para 2, 4, 8, 16, 32 e 64 processos em 1, 2, 4, 8 e 16 máquinas. Os menores tempos para cada configuração estão destacados em negrito.

Número de Máquinas	2	4	8	16	32	64
1	**213**	180	131			
2	215	142	**96**	112		
4		**125**	128	102	90	
8			139	**90**	**62**	**95**
16				178	76	133

Tabela 3. Tempo de execução (em s) na malha discretizada em 800×800 pontos para 2, 4, 8, 16, 32 e 64 processos em 1, 2, 4, 8 e 16 máquinas. Os menores tempos para cada configuração estão destacados em negrito.

Número de Máquinas	2	4	8	16	32	64
1	**2.828**	2.434	1.493			
2	3.859	2.432	1416	835		
4		**2.431**	**1412**	809	593	
8			1.573	**801**	**509**	**473**
16				931	555	561

(usa-se o padrão *ch3:nemesis*). Por exemplo, no caso da Fig. 1, os processos localizados em uma mesma máquina se comunicariam por memória compartilhada. Contudo, de modo surpreendente, os resultados mostram que o emprego de um número maior de máquinas pode, eventualmente, reduzir o tempo de execução, ao invés de aumentá-lo. Essa redução do tempo pode ocorrer devido à disponibilidade de maior quantidade de memória cache em cada uma das configurações. No entanto seria necessário o uso de ferramentas de *profile* para verificarmos a veracidade dessa hipótese.

A Tabela 2 apresenta os tempos de execução de 2, 4, 8, 16, 32 e 64 processos em distintas configurações em uma malha de 400×400 pontos, a Tabela 3 apresenta os tempos de execução nas mesmas configurações para uma malha de 800×800.

5. Conclusões e trabalhos futuros

Este trabalho apresentou a paralelização, usando MPI, de um simulador da formação de edemas em um tecido cardíaco. Na sequência, a versão paralela do código foi executada em um *cluster* de 8 máquinas, sendo cada máquina composta por 8 núcleos, totalizando assim 64 núcleos. A versão paralela obteve ganhos de desempenho de até 14 vezes em relação a sua versão sequencial, reduzindo o tempo de computação de cerca de 5.201s para 366s. Também foram feitos experimentos forçando que os processos MPI fossem criados em diferentes máquinas, de modo que se pudesse avaliar seu impacto no tempo de comunicação. Surpreendentemente, apesar do suporte para memória compartilhada oferecido pelo *mpich*, que deveria reduzir os tempos de comunicação quando uma mensagem fosse enviada para um destinatário localizado na mesma máquina, os experimentos mostraram que para certas configurações reduz-se o tempo de execução ao alocar os processos MPI em um número maior de máquinas. Suas causas, no entanto, devem ser melhor investigadas em trabalhos futuros.

Adicionalmente, como trabalhos futuros, pretende-se melhorar a implementação atual, dividindo a computação de cada núcleo em dois pedaços: bordas e pontos interiores. A ideia é computar inicialmente as bordas e, enquanto os pontos interiores são computados, realizar em paralelo a troca das bordas. Também deseja-se a) utilizar aceleradores, como GPUs, na computação, b) investigar se de fato a divisão das malhas atual é a que leva ao melhor desempenho, e c) coletar eventos do processador para averiguar os impactos da *cache* nas variações de tempo de execução de cada configuração.

Agradecimentos

Os autores agradecem o apoio do CNPq, FAPEMIG e UFJF para a elaboração desse trabalho.

Referências

Guyton, A. C., Hall, J. E., and Guyton, A. C. (2006). *Tratado de fisiologia médica*. Elsevier Brasil.

Keener, J. P. and Sneyd, J. (1998). *Mathematical physiology*, volume 8. Springer.

OMS (2014). Organização Mundial da Saúde. Publicação Eletrônica - http://www.who.int/. Último acesso em 21 de setembro de 2016.

Pacheco, P. (2011). *An Introduction to Parallel Programming*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1st edition.

Pacheco, P. S. (1996). *Parallel Programing with MPI*. Morgan Kaufmann Publishers.

Reis, R. F. (2018). *Modelagem Mecânica da Formação de Edemas*. PhD thesis, Universidade Federal de Juiz de Fora.

Reis, R. F., dos Santos, R. W., de Oliveira Campos, J., and Lobosco, M. (2016a). Interstitial pressure dynamics due to bacterial infection. *Mecánica Computacional. Bioengineering And Biomechanics (B)*, 34(1):1181–1194.

Reis, R. F., dos Santos, R. W., and Lobosco, M. (2016b). A plasma flow model in the interstitial tissue due to bacterial infection. *Lecture Notes in Computer Science*, pages 335–345.

Scallan, J., Huxley, V. H., and Korthuis, R. J. (2010). *Capillary fluid exchange: regulation, functions, and pathology*, volume 2. Morgan & Claypool Life Sciences.

Sompayrac, L. M. (2008). *How the Immune System Works*. Wiley, John & Sons, Incorporated.