Simulação Computacional Paralela Baseada em Autômatos Celulares: Estudo de Caso em Simulação da Dinâmica de Nuvens

Alisson Rodrigo da Silva, Carlos Augusto da Silva Paiva Martins, Maury Meirelles Gouvêa Júnior

Programa de Pós-Graduação em Engenharia Elétrica
Pontifícia Universidade Católica de Minas Gerais (PUC Minas)
Belo Horizonte, MG – Brasil

alisson.sistemas@gmail.com, capsm@pucminas.br, maury@pucminas.br

Abstract. *Simulation methods to solve complex problems are usually run on supercomputers or distributed computing platforms using parallel computing resources. This work proposes a parallel implementation of a cellular automaton using as a case study of a real application of dynamic simulation of clouds. OpenMP API was used as a tool to parallelize the code. A validation of the impact by using parallelism with respect to the performance showed that the processing time decreased as the number of threads increased until the later does not exceed the number of physical cores.*

Resumo. *Métodos de simulação para resolver problemas complexos são geralmente executados em supercomputadores ou plataformas de computação distribuída utilizando os recursos de computação paralela. Esse trabalho apresenta uma proposta de implementação paralela de um autômato celular utilizando como estudo de caso uma aplicação real de simulação da dinâmica de nuvens. Foi utilizado a API OpenMP como meio de paralelizar o código. Foi realizada uma avaliação do impacto do uso paralelismo em relação ao desempenho mostraram que o tempo de processamento diminuiu à medida que o número de threads foi aumentada enquanto estas não ultrapassaram o número de núcleos físicos.*

1. Introdução

A simulação computacional proporciona diversos benefícios, como econômico, temporal e segurança. Existem diversos métodos de simulação que são utilizados para a construção de um modelo real, como Gerador de Números Aleatórios [L'ecuyer 1994] para distribuição, recorrência e correlação; Simulação de Monte Carlo [Fishman 1996] para cálculo de integral, área e volume; caminho aleatório [Gould *et al.*1988] aplicados em ciência da computação, física e economia; e Autômato Celular [Wolfram 2002] para simulação de sistemas, como comportamento de bactérias, vírus e gases, espalhamento de incêndios, desenvolvimento de populações, sistemas econômicos.

Dentre os métodos de simulação supracitados, os autômatos celulares (ACs) têm ganhado destaque nos últimos anos, pois existem algumas vantagens de utilizá-lo em simulações de sistemas complexos, como a possibilidade de representá-los com um conjunto de regras de transição de estado. Um AC é composto por uma grade, ou *grid*, n-dimensional, onde são dispostas células com regras de transição de estado, que interagem com suas células vizinhas produzindo um comportamento local que permite emergir a um global. As regras de transição refletem a dinâmica do sistema que pode variar de linear até caótico. Outra característica importante dos ACs é que sua natureza

favorece sua implementação utilizando o paradigma da computação paralela, em razão da sua estrutura celular e grande demanda de cálculos. A computação paralela pode, assim, reduzir o tempo de processamento e melhorar a precisão dos resultados [Kindratenko *et al.* 2011].

Neste trabalho utilizou-se um estudo de caso relacionado à área de climatologia, mais especificamente à dinâmica de nuvens. O sistema de simulação de dinâmica de nuvens foi modelado utilizando o autômato celular. Em problemas envolvendo sistemas atmosféricos, é desejável que os resultados sejam não só de previsões para curto ou médio prazo, mas também em tempo real, por exemplo, para analisar a evolução de tempestades e furacões. Entretanto, com a diversidade de arquiteturas paralelas existentes, não é trivial a escolha de uma arquitetura para ser utilizada, pois de acordo com as características da aplicação pode-se obter um ganho de desempenho bom ou ruim. Neste trabalho, implementa-se um código paralelo a partir do código sequencial de um modelo de simulação de dinâmica de nuvens com autômatos celulares. Buscou-se avaliar o impacto no tempo de resposta da simulação, causados pela utilização de computação paralela em computador com memória compartilhada.

2. Modelo para Simulação de Dinâmica de Nuvens

Um autômato celular (AC) [Wolfram 2002] é um sistema discreto composto por um conjunto de células cada uma podendo assumir um número finito de estados. As células, além de poderem ter uma dinâmica própria, interagem com suas células vizinhas. A função ou regras de transição é uma função determinística que fornece o estado $s_i(k+1)$ da i-ésima célula no instante $k+1$ como uma função dos estados das células pertencentes à vizinhança N_i. Uma vizinhança é constituída por uma célula central e outras ao seu redor. Em princípio, as regras de transição podem ser representadas por uma tabela de transição, que fornece o estado futuro de uma célula a partir do estado atual da própria célula e de suas vizinhas.

Para simular a dinâmica de nuvens em duas dimensões, como uma função da temperatura ambiente, é necessário no mínimo seis equações diferenciais parciais [Vianello 2000] que definem o comportamento das velocidades latitudinal e longitudinal do ar, da temperatura, do vapor de água e da água da nuvem.

No modelo proposto, cada célula pode assumir dois estados: presença ou ausência de parte de uma nuvem. As regras de transição são baseadas em princípios termodinâmicos que compõem a física de nuvens. O vapor de água está presente em uma microrregião devido a condensação de vapores de água na atmosfera. Assim, uma nuvem surge quando a temperatura declina até que o vapor de água condense formando micropartículas de nuvem. O limite usado para condensação do vapor de água é a temperatura do ponto de orvalho. Esse é o limite no qual o vapor de água acumulado na atmosfera está na sua máxima concentração [Vianello 2000].

O modelo proposto simula nuvens em um *grid* de ACs com vizinhança de Von Neumann, onde cada célula que representa uma microrregião na atmosfera possui três atributos: temperatura e as componentes vertical e horizontal do vetor vento. Cada célula possui uma coordenada (x, y), representando os eixos longitudinal e latitudinal, respectivamente. Assim, (x_l, y_l), (x_r, y_r), (x_a, y_a) e (x_b, y_b) são as coordenadas das células vizinhas à i-ésima célula, Δx e Δy são, respectivamente, os comprimentos horizontal e vertical de cada célula.

A temperatura do ponto de orvalho, T_d, é definida como a temperatura na qual a quantidade de vapor de água presente na atmosfera estaria em sua máxima

concentração, isto é, trata-se de uma temperatura crítica entre o estado de vapor e a condensação da água na atmosfera. Com uma temperatura atmosférica acima de T_d, a água mantém-se em forma de vapor e abaixo de T_d na forma líquida. A temperatura do ponto de orvalho, T_d, pode ser calculada pela expressão

$$T_d = \frac{186.4905 - 237.3 \log e}{\log e - 8.2859} \qquad (2)$$

sendo,

$$e = e_s - A\, p\, (T - T_u) \qquad (3)$$

a pressão de vapor real, e_s a pressão de saturação do vapor de água, A a constante psicométrica, p a pressão atmosférica local e $(T - T_u)$ a depressão psicométrica. Uma nuvem surge na *grid* quando a temperatura atmosférica pertencer ao intervalo $[T_d - \delta, T_d + \delta, T_d + \delta]]$, sendo δ uma constante.

O fluxo de vento é descrito por um campo vetorial, sendo cada vetor implementado em uma célula e descrito por componentes vertical e horizontal, (x_i, y_i). As Equações que definem a dinâmica do fluxo de vento na forma discreta, para a i-ésima célula em uma dada iteração, são aproximadas, respectivamente, por

$$\frac{\partial u_x}{\partial x} \approx \frac{\Delta u_{x_i}}{\Delta x_i} \qquad (4)$$

e

$$\frac{\partial u_y}{\partial y} \approx \frac{\Delta u_{y_i}}{\Delta y_i}, \qquad (5)$$

sendo u_x e u_y as velocidades latitudinal e longitudinal do vento. As diferenças Δx e Δy são definidas pela dimensão que uma célula da *grid* representa no espaço 2D. Assim, as equações diferenciais do modelo atmosférico são definidas como equações de diferenças, a saber

$$u_{x_i}(k+1) = u_{x_i}(k) + \Delta u_{x_i}(k), \qquad (6)$$

para a velocidade latitudinal do ar,

$$u_{y_i}(k+1) = u_{y_i}(k) + \Delta u_{y_i}(k), \qquad (7)$$

para a velocidade longitudinal do ar, e

$$T(k+1) = T(k) + \Delta t\left(-u_{x_i}\frac{\Delta T}{\Delta x_i} - u_{y_i}\frac{\Delta T}{\Delta y_i} + F_T + \phi_T\right), \qquad (8)$$

para a temperatura, sendo

$$\Delta u_{x_i}(k) = \Delta t\left(-u_{x_i}\frac{\Delta u_{x_i}}{\Delta x_i} - u_{y_i}\frac{\Delta u_{x_i}}{\Delta y_i} - \frac{1}{\rho_0}\frac{\Delta p_i}{\Delta x_i} + F u_{x_i}\right), \qquad (9)$$

e

$$\Delta u_{y_i}(k) = \Delta t\left(-u_{x_i}\frac{\Delta u_{y_i}}{\Delta x_i} - u_{y_i}\frac{\Delta u_{y_i}}{\Delta y_i} - \frac{1}{\rho_0}\frac{\Delta p_i}{\Delta y_i} + F_{u_{y_i}} \right), \quad (10)$$

sendo F_i e ϕ_T a força de buoyant e a entropia, para $i = \{ u_x, u_y, T \}$, respectivamente.

3. Proposta de Implementação Paralela

Na última década, as arquiteturas de sistemas de computação vêm se desenvolvendo em relação à capacidade de processamento. Hoje existem processadores *CPU (Central Processing Unit)* com múltiplos núcleos em um mesmo chip chamados de *multicore* e *manycore*. Estes possuem memória compartilhada, ou seja, os núcleos de processamento compartilham o mesmo recurso de memória.

Atualmente, para construir códigos paralelos para essa arquitetura, existe uma API (Application Programming Interface) chamada *OpenMP (Open Multi-Processing)*. A utilização do padrão *OpenMP* tem crescido bastante nos últimos anos, uma vez que as funcionalidades do mesmo facilitam o desenvolvimento de aplicações em memória compartilhada [Sena *et al.* 2008].

Para o problema abordado nessa pesquisa, a proposta de solução é utilizar a computação paralela de memória compartilhada. Após a leitura dos dados com os valores de entrada do sistema, a *grid*, é inicializada. Essa *grid* é uma matriz instanciada na memória do computador e esta não é uma matriz de tipo primitivo, isto é, cada posição da matriz aponta para um objeto do tipo *cell* que contém variáveis e métodos. Em seguida, insere-se uma nuvem na *grid* e, assim, o processamento da dinâmica de nuvens é iniciado. O laço de iteração só termina quando alcança o número máximo de iterações previamente escolhidas pelo usuário do sistema. Neste trabalho, optou-se por paralelizar somente o processamento das células.

A arquitetura escolhida para os testes foi a arquitetura *multicore*. Escolheu-se um computador que contém 4 núcleos físicos, mas que emula 8 núcleos lógicos. Escolheu-se o padrão *OpenMP* como meio de paralelizar o código.

4. Estudo Experimental

Para realização dos testes, foi utilizado um micro-computador Intel® Core™ i7-3632QM (2.2GHz até 3.2GHz com Intel® Turbo Boost 2.0, 8 *Threads*, 6Mb *Cache*), memória de 8GB, Dual Channel DDR3, 1600MHz e sistema operacional Windows 8 de 64 bits. Para a realização dos testes escolheu-se quatro matrizes com tamanhos diferentes, definidos em: 100x100, 200x200, 400x400, 800x800. As diferentes resoluções de *grid* representam o aumento da resolução de uma mesma área simulada. Assim, mais células terão que ser manipuladas e o volume de cálculo será ser maior. Foram realizados experimentos com diferentes números de iterações, a saber, 100, 200, 400, 800, para que a estabilidade numérica fosse analisada, além de avaliar a correlação entre o número de iterações e a convergência dos resultados. O tamanho da nuvem aumenta proporcionalmente ao aumento da *grid*. Os demais parâmetros do sistema foram fixados com os seguintes valores: altitude igual a 5 mil metros, temperatura atmosférica de -3 °C, ponto de orvalho, T_d, igual a 2°C, pressão atmosférica de 700 hPa e vento constante em direção, sentido e intensidade.

Quanto ao paralelismo, variou-se o número de *threads* em 2,4,8,16 para analisar o ganho de *speedUp*. A escolha desses valores está correlacionada ao número de núcleos do processador utilizado nos experimentos (quatro núcleos físicos que emulam 8 núcleos lógicos). Busca-se verificar o comportamento dos tempos com paralelismo

puro, quando o número de *threads* é igual ao número de processadores, e do paralelismo com concorrência, quando o número de *threads* é maior que o número de processadores.

Cada experimento foi executado 30 vezes. Para cada execução, o *software* foi iniciado e fechado, para que o lixo de memória não interferisse no tempo de resposta. O computador foi desconectado da rede *wireless* e da rede *ethernet*, alguns serviços foram desativados, ficando ativos somente os serviços essenciais do sistema operacional.

4.1. Resultados

As Figuras 1, 3, 5, 7 mostram gráficos com os tempos de execução. Para cada tamanho de matriz apresentada, o rótulo (a) se refere ao tempo de execução com 100 iterações e o rótulo (b) o tempo com 800 iterações. Os tempos mínimo, médio e máximo, foram medidos entre as amostras das 30 execuções realizadas. O tempo mínimo representa o melhor desempenho do tempo de reposta alcançado pelo processador. Acredita-se que o tempo mínimo foi alcançado quando o programa foi executado sem nenhuma interrupção do sistema operacional ou foi executado com poucas interrupções. O tempo médio, calculado com a média aritmética, mostra o tempo de execução que na maioria das vezes será alcançado quando o programa for executado. O tempo máximo representa o tempo que provavelmente o sistema operacional gerenciou recursos para um maior número de processos, afetando o desempenho da aplicação.

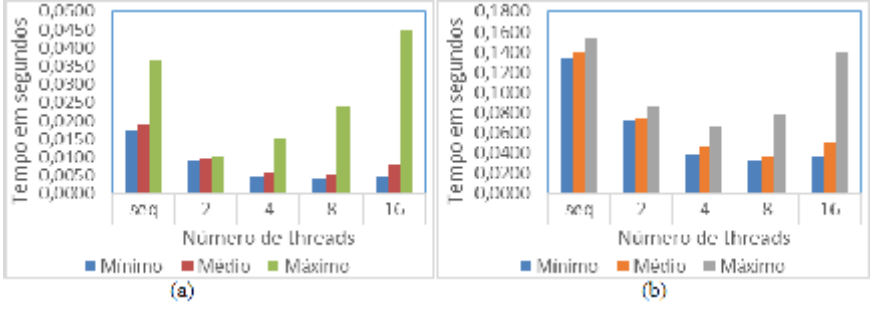

Figura 14. Tempo execução com matriz 100x100: (a) 100 e (b) 800 iterações

A Figura 1 mostra que o tempo de resposta diminuiu à medida aumentou-se o número de *threads*, até chegar em 8. O tempo com 16 *threads* obteve pior desempenho como indicado no *speedUp* da Figura 2. Supõe-se que, a perda de desempenho foi devido ao paralelismo com concorrência, já que o número de *threads* foi o dobro do número de núcleos. A menor diferença entre os tempos mínimo, médio e máximo foi com 100 iterações e 2 *threads*, o que indica que houve menor variância.

Na avaliação geral, o melhor desempenho considerando o tempo médio foi com 8 *threads*, Figura 2. Supõe-se que tal desempenho se deve ao paralelismo puro com os 8 núcleos lógicos. Observa-se o mesmo comportamento no tempo mínimo. Já pelo tempo máximo, o melhor desempenho foi com 2 *threads*, acredita-se que durante a execução desse experimento, não houve muita disputa por recursos como nos outros experimentos. O melhor desempenho global foi com o tempo mínimo e com 8 *threads*. Tal fato, sugere que os 8 núcleos lógicos foram utilizados da melhor forma pelo sistema operacional do que com 4 núcleos físicos. O pior desempenho global foi com o tempo máximo e 16 *threads*, Figura 2(a), o que pode ser justificado pelo paralelismo com concorrência. Considerando o tempo máximo, o melhor desempenho foi com 4 *threads*, o que supõe que no momento dessa execução houve pouca concorrência por recursos em relação aos outros experimentos.

As Figuras. 2, 4, 6, e 8 mostram gráficos com o desempenho (*SpeedUp*) alcançado com código paralelo em relação ao sequencial. Esses resultados de desempenho refletem os tempos de execução dos experimentos apresentados nos gráficos de tempo, portanto seguem os mesmos critérios dos rótulos das figuras, de tamanho de matriz, números de iteração e valores mínimo, médio e máximo. O *SpeedUp* mínimo foi calculado utilizando o tempo sequencial mínimo em relação aos tempos paralelos mínimos com 2, 4, 8 e 16 *threads*. O *speedUp* médio e máximo seguiram a mesma lógica.

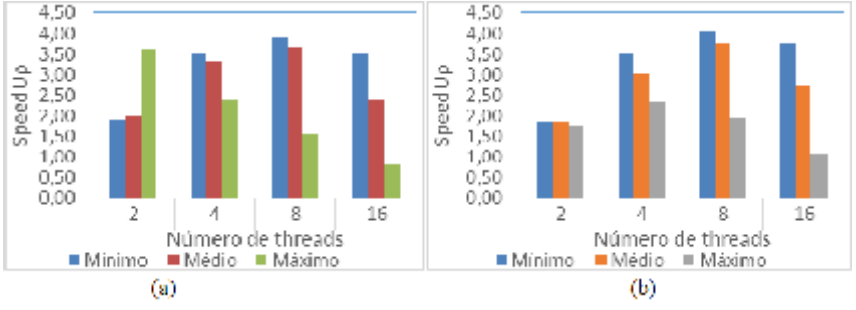

Figura 15. *SpeedUp* para matriz 100x100 em (a) 100 e (b) 800 iterações

Os valores de *SpeedUp* apresentados na Figura 2 corroboram com a análise de desempenho feita na Figura 1. Observa-se que na Figura 2(b), o maior *SpeedUp* global foi de 4,05 vezes com 8 *threads*.

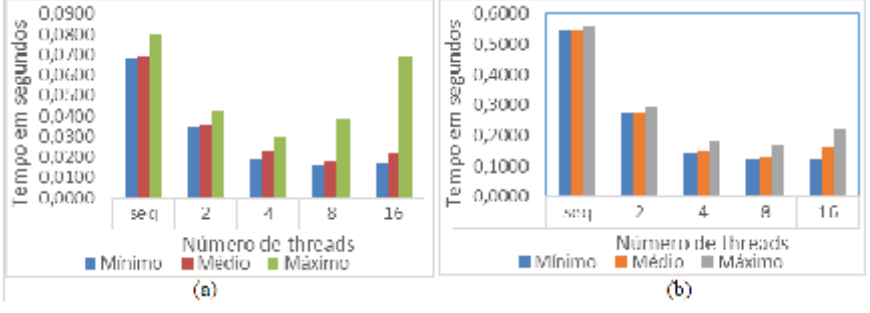

Figura 16. Tempo execução para matriz 200x200 em (a) 100 e (b) iterações

A Figura 3 mostra que o uso do paralelismo e o aumento do número de *threads* seguiu a mesma tendência da matriz 100x100. O mesmo aconteceu com o melhor desempenho, o desempenho global e o pior desempenho. A menor diferença entre os tempos mínimo, médio e máximo foi com 800 iterações e 2 *threads* com variância menor que 6,7%, Figura 3(b). Já pelo tempo máximo, o melhor desempenho foi com 100 iterações e 4 *threads*, Figura 4(a) e com 800 iterações foi com 8 *threads*, Figura 4(b). Os tempos médios com 8 *threads* da matriz 200x200, Figura 3, foram menores que os tempos sequenciais da matriz 100x100, Figura 2, isto significa que houve ganho no desempenho e na qualidade dos resultados, pois aumentou a precisão da área simulada.

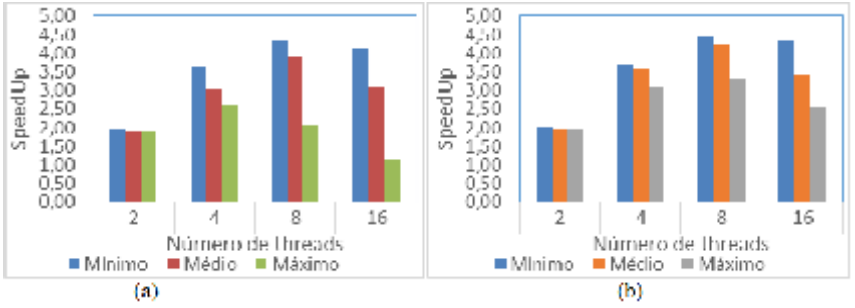

Figura 17. *SpeedUp* para matriz 200x200 em (a) 100 e (b) 800 iterações

Os valores de *SpeedUp* apresentados nas Figura 4 corroboram com a análise de desempenho feita na Figura 3. O maior *SpeedUp* global foi de 4,47 com 8 *threads* Figura 4(b).

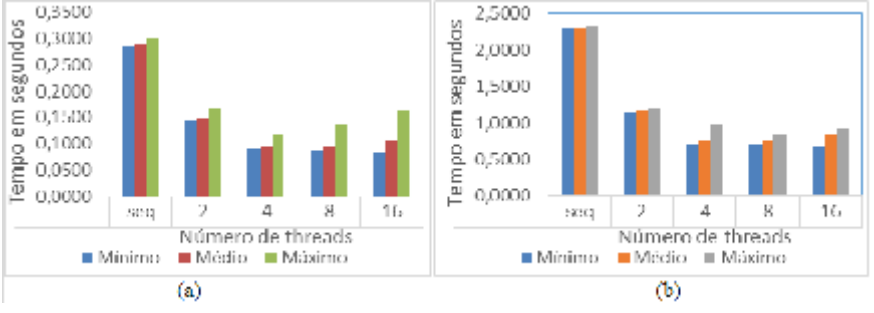

Figura 18. Tempo de execução para matriz 400x400 em (a) 100 e (b) 800 iterações

A Figura 5 mostra que o uso do paralelismo e o aumento do número de *threads* seguiu a mesma tendência da matriz 100x100 e 200x200. Na avaliação geral, o melhor desempenho considerando o tempo médio foi com 4 *threads*. Para o tempo mínimo foi com 16 *threads*, Figura 6. Já pelo tempo máximo, o melhor desempenho foi com 4 *threads* para 100 iterações, Figura 6(a) e 8 *threads* para 800 iterações, Figura 6(b). O melhor desempenho global foi com o tempo mínimo e 16 *threads*, Figura. 6(b). O pior desempenho global foi com o tempo máximo e 2 *threads*, Figura 6(a). A menor diferença entre os tempos mínimo, médio e máximo foi com 800 iterações e 2 *threads*, Figura 5(b), com variância menor que 2,66%.

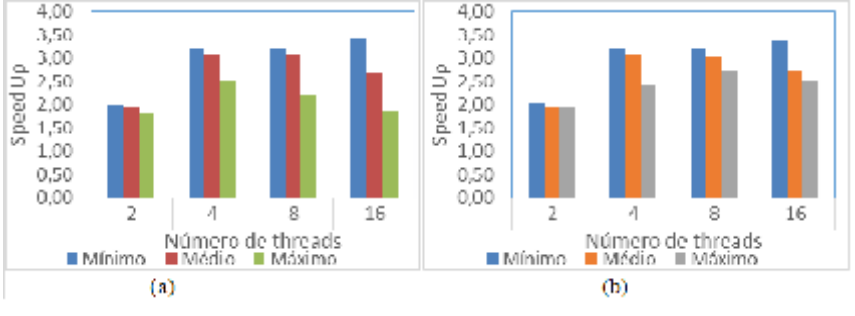

Figura 19. *SpeedUp* para matriz 400x400 em (a) 100 e (b) 800 iterações

Os valores de *speedUp* apresentados na Figura 6 corroboram com a análise de desempenho feita na Figura 5. O maior *SpeedUp* global Figura 4(b) foi de 3,43 e foi menor que o *SpeedUp* das matrizes 100x100 e 200x200. Acredita-se que esse comportamento pode estar relacionado com o aumento da matriz, já que nesse caso mais dados foram alocados na memória e consequentemente aumentou o número de falha de acesso à memória cache quando os dados foram acessados, afetando o desempenho.

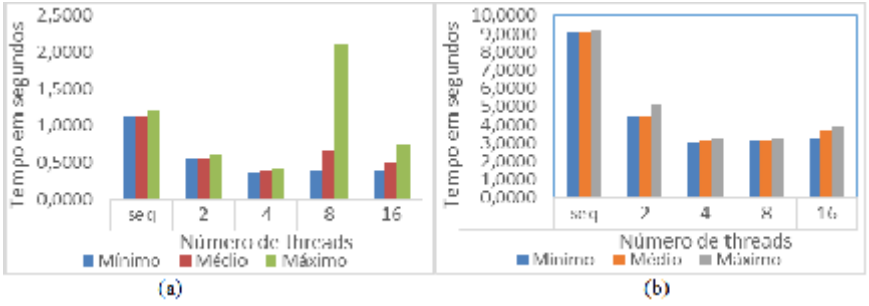

Figura 20. (a) tempo execução para matriz 800x800 em (a) 100 e (b) 800 iterações

A Figura. 7 mostra que o uso do paralelismo o tempo de resposta diminuiu à medida que foi aumentada a quantidade de *threads*, mas até chegar em 4. Observa-se que o tempo com 8 e 16 *threads* foi pior que com 4. O tamanho dessa matriz é 16 vezes maior que a matriz 100x100, com isso, supõe-se que para esse volume de dados, os 8 núcleos emulados não obteve um bom desempenho devido às falhas de acesso à memória cache. Na avaliação geral, o melhor desempenho considerando o tempo mínimo foi com 4 *threads* e analisando pelo tempo máximo, o melhor desempenho para 100 iterações, Figura 7(a) foi com 4 *threads* e para 800 iterações, Figura 7(b) foi com 8 *threads*. O melhor desempenho global foi com o tempo mínimo e 4 *threads*, Figura 7. O pior desempenho global foi com o tempo máximo e com 8 *threads*, Figura 7(a).

A menor diferença entre os tempos mínimo, médio e máximo foi com 100 iterações e 4 *threads*, Figura7(a), com variância menor que 4,56%.

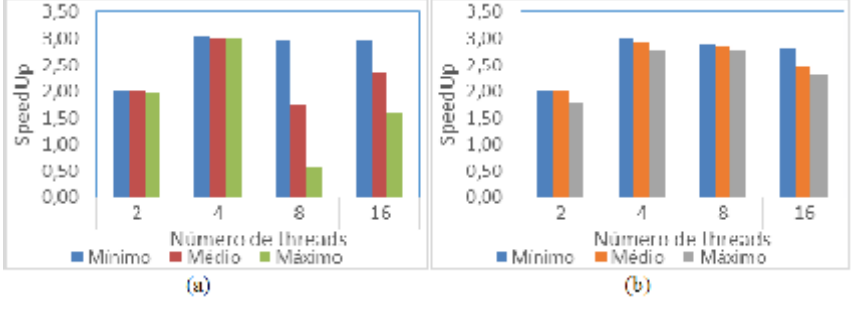

Figura 21. *SpeedUp* para matriz 800x800 em (a) 100 e (b) 800 iterações

Os valores de *speedUp* apresentados na Figura 8corroboram com a análise de desempenho feita na Figura 7. O maior *SpeedUp* global na Figura 8(a) foi de 3,04 vezes maior. Observa-se que esse foi menor quando comparado com todas as matrizes anteriormente analisadas. Isso supõe ser falha de acesso à memória cache por causa do aumento do número de dados.

4.2 Análise dos Resultados

Foi observado que nas *grids* com tamanho 100x100 e 200x200 o melhor *speedUp* considerando o tempo médio foi com 8 *threads*, acredita-se que os 8 núcleos lógicos foram utilizados no máximo desempenho pelo sistema operacional, já para as *grids* 400x400 e 800x800 o melhor *speedUp* foi alcançado com 4 *threads* e supõe que com o volume de dados maior os 4 núcleos físicos foram utilizado no máximo desempenho.

O valor do melhor *speedUp* global para as matrizes 100x100, 200x200, 400x400 e 800x800 foi respectivamente, 4.05, 4.47, 3.43 e 3.04. Esses valores indicam que com o aumento do tamanho da *grid* o *speedUp* diminuiu. Isso supõe que que houve aumento de falha de acesso à memória cache, o que prejudicou diretamente o desempenho.

Analisando a correlação dos tempos de execução com o número de iterações e o tamanho das matrizes, a complexidade do algoritmo apresentado pode ser descrita como $O(m^2 \times n_i)$, sendo m^2 a dimensão da matriz e n_i o número de iterações.

5. Conclusão

Esse trabalho teve por objetivo avaliar os ganhos do uso do paralelismo em relação ao desempenho de um modelo de simulação da dinâmica de nuvens com implementação série. Para isso utilizou-se uma arquitetura *multicore*, com a API *OpenMP* de memória compartilhada. Observou-se que houve ganho de desempenho com o paralelismo. O melhor desempenho foi de 4,47 vezes maior em relação ao tempo do código sequencial, sendo este superior à quantidade de núcleos físicos, mas praticamente a metade da quantidade de núcleos lógicos, pois a arquitetura física é que de fato realiza o processamento. Foi observado que com o aumento do tamanho das matrizes o *speedUp* foi diminuindo. Acredita-se que esse comportamento se deve ao fato de que com um volume de dados maior, gera-se um número maior de falha de acesso à memória cache, comprometendo diretamente o desempenho.

Referências

L'ecuyer, P. (1994) "Uniform random number generation". Annals of Operations Research, vol. 53, p. 77-120.

Fishman, G. S. (1996) "Monte Carlo: Concepts, Algorithms, and Applications". Springer Series in Operations Research. New York: Springer-Verlag.

Gould, H. and Tobochnik, J. (1988) "An Introduction to Computer Simulation Methods: Applications to Physical Systems" -- Part 2. Addison-Wesley.

Wolfram, S. (2002) "A New Kind of Science". Champaign: Wolfram Media.

Kindratenko, V. and Trancoso, P. (2011) "Trends in High-Performance Computing". Computing in Science & Engineering, vol.13, n.3, p.92-95, mai-jun.

Vianello, R. L. (2000) "Meteorologia Básica e Aplicações", Editora UFV.

Sena, M. C. R. and Caldas, J. A. C. (2008) "Tutorial OpenMP C/C++ Programa Campus Ambassador HPC, Boulic, R. and Renault, O. "3D Hierarchies for Animation", In: New Trends in Animation and Visualization, Edited by Nadia Magnenat-Thalmann and Daniel Thalmann, John Wiley & Sons ltd., England.