Multiplicação Paralela de Matrizes

Thiago R. P. M. Rúbio[1], **Cristina D. Murta**[1]

[1]CEFET-MG
Departamento de Computação
Belo Horizonte, MG

thiagoreismunhoz@gmail.com, cristina@decom.cefetmg.br

Resumo. *Este artigo apresenta uma avaliação experimental de cinco implementações paralelas da operação de multiplicação de matrizes. As implementações são feitas nos ambientes de programação paralela POSIX Threads e Hadoop. Os resultados indicam larga vantagem para o Hadoop em termos de tempo de execução. O trabalho contribui também para apresentar e discutir as várias possibilidades de implementação da operação nos ambientes paralelos.*

Abstract. *This paper presents an experimental evaluation of five parallel implementations of the matrix multiplication operation. The implementations are made in POSIX Threads and Hadoop parallel programming environments. The results show that Hadoop provides the best execution times. The work also contributes to present and discuss many possible implementations of that operation in parallel environments.*

1. Introdução

Nas últimas décadas, a frequência dos processadores aumentou continuamente de unidades de MHz para GHz. Após enorme progresso, um limite físico foi alcançado para frequências elevadas: o calor gerado pelos circuitos densamente empacotados é maior do que a capacidade de dissipação desse calor. Assim, a indústria passou a oferecer *chips* com múltiplos núcleos de processamento e as novas máquinas tornaram-se paralelas. Esta mudança está sendo considerada um marco para o futuro da computação [Sutter and Larus 2005, Hennessy and Patterson 2007, Oskin 2008, Asanovic et al. 2009].

Nesse novo contexto de arquitetura paralela, a melhoria no tempo de execução de um programa é obtida quando o código pode ser executado paralelamente nos núcleos de processamento disponíveis [Galon and Levy 2008, Manferdelli et al. 2008]. Há um grande esforço na direção de programação paralela. Esse artigo visa contribuir nessa direção, apresentando uma avaliação experimental de cinco implementações paralelas da operação de multiplicação de matrizes. Estas implementações são executadas em máquinas *multicore*. O objetivo é avaliar e comparar os custos das implementações. A multiplicação de matrizes é uma operação essencial em várias áreas de processamento científico, e é uma operação computacionalmente cara. Assim, a execução paralela do algoritmo provê ganhos significativos que podem beneficiar um grande número de aplicações.

As implementações testadas foram feitas utilizando dois ambientes importantes de programação paralela: o ambiente de programação POSIX Threads, também conhecido como Pthreads, e o ambiente Hadoop. O Pthreads é um padrão para programação concorrente baseado em threads com implementações para vários sistemas operacionais [Nichols et al. 1996]. O Hadoop é uma implementação de código aberto do modelo MapReduce para programação paralela [Dean and Ghemawat 2008]. Os resultados indicam que as implementações em Hadoop são amplamente favoráveis quando comparadas com a implementação em Pthreads.

Este trabalho foi realizado por Thiago R. P. Munhoz Rúbio, que é aluno do curso de graduação em Engenharia de Computação do CEFET/MG. Para a realização desse trabalho, o aluno recebeu bolsa PIBIC/FAPEMIG e foi orientado pela professora Cristina Duarte Murta.

2. Trabalhos Relacionados

A recente mudança na arquitetura de computadores – de máquinas de um processador para máquinas com múltiplos processadores – está sendo vista como um caminho definitivo para a programação paralela [Asanovic et al. 2009]. Neste sentido, ganham importância os ambientes que dão suporte a implementações paralelas de programas.

As bibliotecas que implementam threads são um dos mais tradicionais ambientes para programação concorrente e paralela. Dentre estas, está o Pthreads, um padrão IEEE, que é um dos ambientes de programação paralela escolhidos para esse trabalho. A implementação de threads no Pthreads é explícita, cabendo ao programador controlar todos os aspectos das threads, incluindo sua criação, associação com as funções, sincronização e controle das interações entre as threads e os recursos compartilhados [Nichols et al. 1996].

O segundo ambiente de programação paralela utilizado nesse trabalho é o MapReduce, um modelo de programação associado a um ambiente de execução desenvolvido no Google [Dean and Ghemawat 2008, Dean and Ghemawat 2010]. A abstração é baseada nas primitivas *map* e *reduce* encontradas em linguagens funcionais, por exemplo, Lisp. A computação consiste em aplicar a operação *map* a cada registro de entrada, gerando um conjunto de valores intermediários que são associados a cada chave dos registros. Posteriormente a operação *reduce* agrupa os registros que compartilham a mesma chave. As computações executadas são sempre controladas por meio de um par chave-valor. O modelo pode ser usado genericamente em programação paralela. No entanto, não é adequado para todos os tipos de problema [Ranger et al. 2007, Dean and Ghemawat 2010].

O modelo MapReduce foi planejado para executar computação paralela e distribuída em grandes conjuntos de máquinas. O modelo inclui recursos que permitem o tratamento de falhas de maneira transparente, além de escalonar a comunicação entre as máquinas e o uso dos recursos rede e disco de forma eficiente. Portanto, o ambiente gerencia e executa várias tarefas que são deixadas para o programador no modelo Pthreads.

Para o desenvolvimento deste trabalho utilizamos o *Apache Hadoop Core* [White 2009], que é uma implementação de código aberto do MapReduce. Para permitir a distribuição de dados e gerenciar as falhas, o Hadoop inclui um sistema de dados distribuído integrado, o *Hadoop Distributed File System (HDFS)*. O Hadoop é

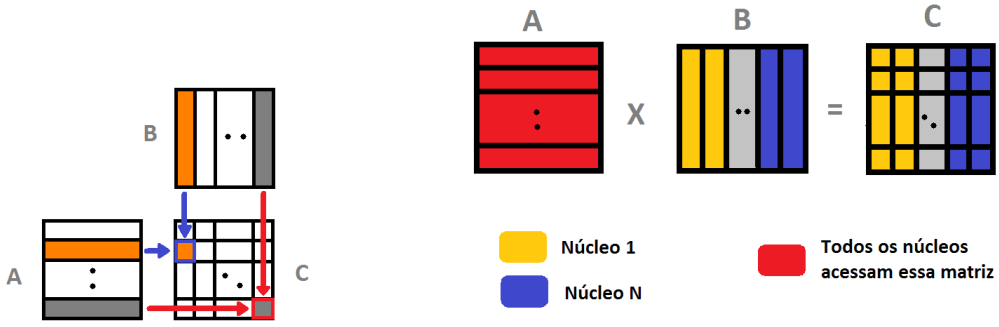
uma ferramenta muito promissora para aplicações com volumes significativos de dados que necessitam de alto desempenho, e vem sendo utilizado por dezenas de empresas, inclusive de grande porte, tais como Facebook e Yahoo!, em ambientes com milhares de máquinas [Dean and Ghemawat 2010, Hadoop 2011].

O trabalho mais próximo desse artigo é a proposta de implementação de quatro versões do algoritmo de multiplicação de matrizes usando o MapReduce [Norstad 2009]. O presente artigo apresenta uma implementação dos algoritmos propostos, bem como uma avaliação experimental e uma comparação com uma implementação da mesma operação no ambiente Pthreads.

3. Estratégias para a Multiplicação Paralela de Matrizes

O objetivo deste artigo é comparar experimentalmente o custo de implementações paralelas da operação de multiplicação de matrizes. Para isso, implementamos uma versão do algoritmo de multiplicação de matrizes usando Pthreads [Nichols et al. 1996] e quatro versões em Hadoop, descritas em [Norstad 2009]. Todos os algoritmos foram implementados e testados, e sua correção foi avaliada antes do processamento dos testes de medição de tempo.

A multiplicação de matrizes é uma operação binária, que gera uma matriz produto, dado um par de matrizes como operandos. O algoritmo de multiplicação de matrizes pode ser esquematizado conforme a Figura 1a, que representa a operação descrita pela Fórmula 1.



(a) Multiplicação (b) Estratégia em Pthreads

Figura 1. Representações da operação de multiplicação de matrizes

A entrada é composta por duas matrizes, A e B, obtendo como saída a matriz C, resultado do produto $A \times B$, calculado conforme a Fórmula 1.

$$C[i,j] = \sum_{k=i}^{N} A[i,k] * B[k,j] \quad (1)$$

A multiplicação de matrizes pode, genericamente, ser realizada em três etapas:

1. seleção dos elementos a_i e b_j de A e B, respectivamente, a serem multiplicados;
2. multiplicação dos elementos selecionados no item 1;
3. soma dos elementos dos produtos de linhas de A por colunas de B (resultado).

A multiplicação de matrizes é um problema que permite alto grau de paralelismo. A estratégia escolhida para a implementação em Pthreads é apresentada na Figura 1b e consiste em dividir o processamento das matrizes entre as threads, definindo blocos de linhas e colunas das matrizes a serem multiplicadas. Cada thread varre todas as linhas da primeira matriz e multiplica apenas algumas colunas da segunda matriz. Dessa forma, cada núcleo processa apenas parte dos elementos da matriz resultante. As matrizes A e B são apenas lidas, a escrita ocorre somente em C. A escolha do número de threads é fundamental para que se obtenha um bom ganho com o paralelismo, sem sobrecarregar o sistema com o gerenciamento das mesmas.

As quatro estratégias de implementação da multiplicação paralela de matrizes implementadas no ambiente Hadoop são descritas a seguir. A paralelização dos algoritmos é feita em função dos *jobs*, que constituem a unidade de divisão das tarefas para cada núcleo de processamento da rede e garantem o paralelismo no modelo MapReduce. Tendo em vista as etapas do processo de multiplicação, enunciadas acima, foram definidos dois *jobs* principais: um para realizar a multiplicação dos elementos e o segundo para somar os resultados intermediários. Cada *job* é composto por *mappers* e *reducers*, sendo que os primeiros tratam do mapeamento dos elementos que devem ser operados juntos (a_{ij} com b_{kl}) e os últimos (*reducers*) executam as operações a serem realizadas sobre os dados.

A diferença entre as quatro estratégias refere-se à maneira de compor esses *jobs*. Assim, cada estratégia aborda uma forma diferente de mapear os elementos, o que influencia diretamente no desempenho dos algoritmos, com vantagens e desvantagens. Veremos a seguir o funcionamento de cada estratégia.

A estratégia 1, ilustrada no diagrama da Figura 2 é a mais simples. Consiste em definir um *mapper* para mapear cada elemento da matriz A com seu multiplicador de B, para cada conjunto de linhas de A e colunas de B. Dessa forma, é possível atribuir uma multiplicação a cada *reducer* do primeiro *job*. O segundo *job* realiza, portanto, as somas dos elementos intermediários por meio de seus *reducers*. Esta estratégia usa muitos *reducers*, fazendo bom uso do paralelismo, mas gera grande quantidade de tráfego de dados.

A estratégia 2 é ilustrada na Figura 3. Essa estratégia consiste em mapear cada elemento de A com todos os elementos de B pelos quais será multiplicado. No primeiro *job* temos um único *reducer* que realiza as multiplicações desse elemento de A, resultando em elementos intermediários que estão dispersos pela rede. Os *mappers* do segundo *job* ficam responsáveis por mapear os elementos, e um *reducer* realiza as somas e finaliza a operação.

Esta é uma melhoria considerável em termos de tráfego de dados, ao custo de se ter menos *reducers*. No entanto, cada um deles deve trabalhar mais, resultando em um menor grau de paralelismo. Além disso, esta estratégia pode ser muito interessante se utilizada com matrizes que não sejam quadradas, uma vez que o número de *mappers* e *reducers* será diferente.

A estratégia 3, ilustrada na Figura 4, é uma imagem espelhada da estratégia 2. No primeiro *job* mapeamos um elemento de B com seus multiplicadores da matriz A e usamos um redutor único para realizar as multiplicações em que este elemento está envolvido. O segundo *job* realiza as mesmas operações realizadas pelo segundo *job* da

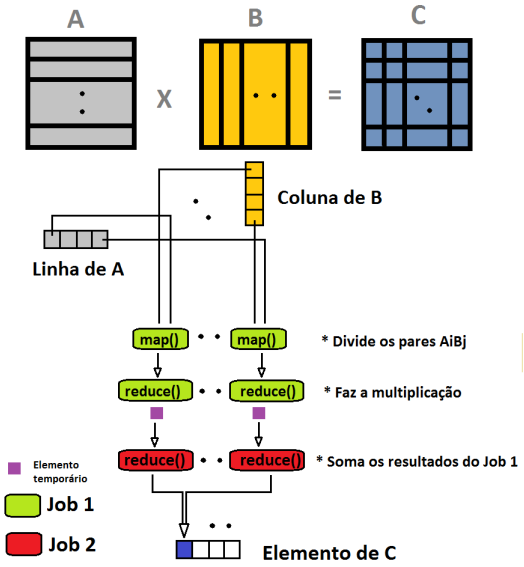


Figura 2. Estratégia 1 para multiplicação de matrizes com MapReduce

Estratégia 2. Esta abordagem apresenta tráfego de dados similar ao da Estratégia 2. No entanto, esta estratégia poderá será melhor que a Estratégia 2 caso as matrizes não sejam quadradas e o número de linhas de A seja menor que o número de linhas de B, visto que o número de *reducers* será menor.

Nas três primeiras estratégias, cada *reducer* do primeiro *job* emite um ou mais elementos da matriz C e precisamos definir um segundo *job* para somar os resultados intermediários e, finalmente, produzir os elementos da matriz C. A quarta estratégia consiste em usar um único *reducer* ainda no *job* 1 para calcular o elemento final de C. Isso é feito associando-se todos os *maps* necessários e acumulando o resultado de cada multiplicação. Dessa maneira não há necessidade de um segundo *job*, o que reduz bastante o custo de manipulação dos dados e o tráfego na rede, embora essa estratégia seja mais difícil de ser programada e controlada. A quarta e última estratégia é apresentada na Figura 5.

A utilização dessas estratégias é vantajosa pelo fato de que, no modelo MapReduce e na sua implementação Hadoop, o programador não precisa preocupar-se com tarefas de mais baixo nível, como o controle de acesso aos dados, o que não ocorre com a programação em Pthreads ou MPI (*Message Passing Interface*), por exemplo. A seguir são descritos os testes executados com estas implementações.

4. Descrição dos Experimentos e do Ambiente de Teste

O objetivo do trabalho é avaliar experimentalmente as versões apresentadas na seção anterior para a operação de multiplicação de matrizes. O primeiro experimento tem como objetivo ajustar a implementação Pthreads, obtendo a configuração do número de threads que produz o menor tempo de execução. A partir desse resultado, a melhor configuração em Pthreads é comparada com as versões em Hadoop. A seguir, comparamos as versões do Hadoop entre si. O último teste compara uma implementação do Hadoop executada em uma máquina com oito núcleos de processamento, com a execução em um conjunto de quatro máquinas com dois núcleos cada uma. Finalmente discutimos a função de com-

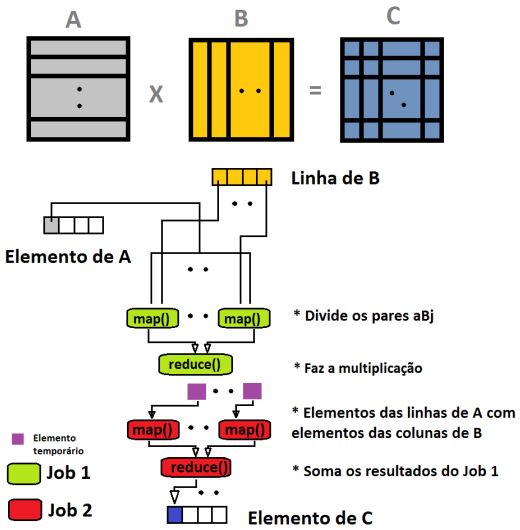


Figura 3. Estratégia 2 para multiplicação de matrizes com MapReduce

plexidade do código.

A implementação das quatro versões em Hadoop foi feita em linguagem Java, utilizando os recursos da biblioteca *Apache Hadoop Core*. Utilizamos também o *plugin* Karmasphere Studio, integrado ao IDE NetBeans, que permite a execução dos projetos do Hadoop a partir da própria IDE. A implementação com Pthreads foi feita em linguagem C, no ambiente IDE NetBeans.

Em todos os testes, as matrizes A e B foram geradas de maneira aleatória, apenas com números inteiros. Somente matrizes quadradas $N \times N$ foram testadas. Os testes consistiram em medir tempos de execução das multiplicações para matrizes de diversos tamanhos (valores distintos de N). O tempo de execução é o tempo decorrido apenas no processo de multiplicação. Os tempos gastos para a criação, alocação e preenchimento das matrizes não são contabilizados. A função de registro do tempo é executada imediatamente antes e após à chamada de função que realiza a multiplicação. Para cada tamanho de matriz, foram feitas dez execuções de cada programa e foi calculado o tempo médio das execuções.

Os testes foram executados em dois tipos de ambientes de hardware. O primeiro tipo é composto por máquinas multiprocessadas contendo processadores Intel Core2Quad (total de quatro ou oito cores) operando a 2.6 Ghz cada, 4 GB de memória RAM e sistema operacional Ubuntu Linux 10.04. O segundo ambiente é um *cluster*, composto por quatro máquinas com 4 GB de memória RAM e processador Intel core2Duo (total de 8 cores) operando a 2.6 GHz cada uma. As máquinas executam o sistema operacional Ubuntu Linux 10.04 e foram interligadas em rede com um Switch 3Com Gigabit. As plataformas de programação são equivalentes e incluem o Hadoop Core em sua versão distribuída, bem como o sistema de arquivos HDFS configurado [White 2009].

5. Resultados

O primeiro teste tem como objetivo avaliar o número ideal de threads a ser definido na implementação com Pthreads. Para isso, foram feitas várias execuções da operação, em

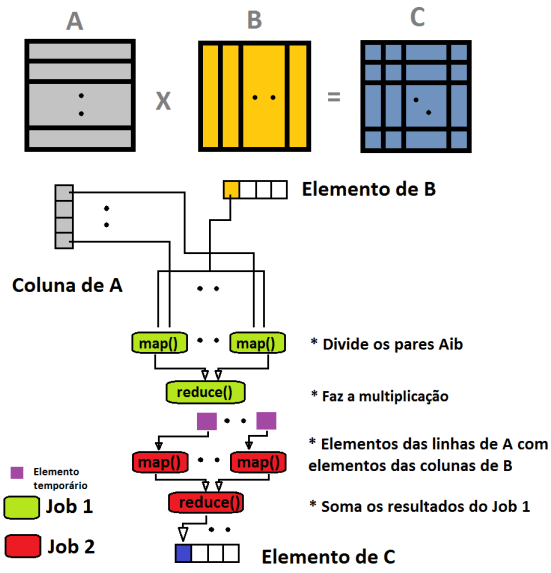


Figura 4. Estratégia 3 para multiplicação de matrizes com MapReduce

matrizes de mesma ordem N, e variando-se o número de threads. O gráfico da Figura 6 apresenta resultados do tempo médio de execução para matrizes de vários tamanhos e número de threads variando entre 1 e 8 para cada tamanho de matriz. A execução ocorreu em uma máquina com quatro cores. Os resultados indicam que o menor tempo médio de execução ocorre quando há uma thread por núcleo de processamento. Um número de threads menor não aproveita bem os recursos da máquina, e um número de threads maior gera custos de escalonamento para tratar mais de uma thread no mesmo núcleo de processamento. Assim, nos demais experimentos, definimos uma thread por núcleo de processamento.

A seguir, comparamos os resultados obtidos com todas as cinco implementações, sendo quatro versões do Hadoop e uma versão em Pthreads. O gráfico à esquerda da Figura 7 apresenta o tempo médio de execução, em segundos, para matrizes quadradas $N \times N$, de ordem N, em que N varia de mil a oito mil elementos.

Observa-se no gráfico que a implementação com Pthreads obtém um tempo de execução bastante superior às implementações do Hadoop. Por exemplo, para $N = 6000$, o tempo de execução da versão em Pthreads é de 5.724 segundos, enquanto a versão mais lenta do Hadoop, a de número 3, executa em 1.326 segundos. Para $N = 8000$, Os tempos médios das duas versões são, respectivamente, 15.659 e 6.245 segundos. A diferença entre os tempos é decrescente: no caso de $N = 6000$, o tempo de execução do Hadoop-03 é 23% do tempo de execução da versão Pthreads. Para $N = 8000$, a proporção cresce para 39%.

A seguir comparamos somente as versões implementadas no Hadoop. O gráfico à direita da Figura 7 apresenta os tempos médios de execução, em segundos, para cada estratégia descrita na Seção 3. Observamos que as estratégias apresentam resultados bastante semelhantes, mas que começam a se diferenciar para matrizes maiores. A partir de $N = 6000$, temos uma ordem consistente de tempos de execução, tendo, do melhor para o pior resultado, as estratégias 4, 2, 1 e 3. Assim, concluímos que a última estratégia

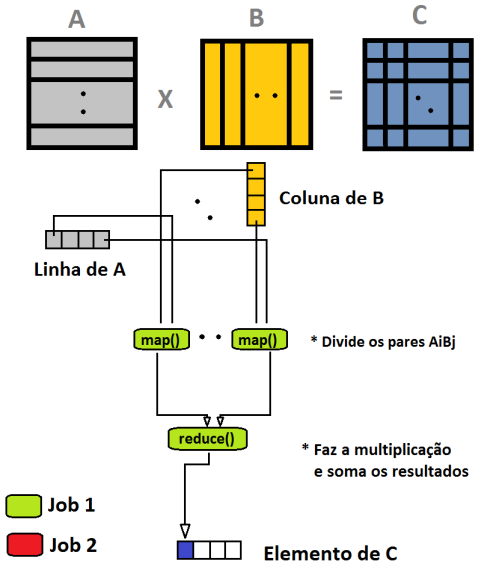


Figura 5. Estratégia 4 para multiplicação de matrizes com MapReduce

proposta, que utiliza somente um *job* obtém o melhor resultado.

A seguir avaliamos a execução dos quatro algoritmos implementados no Hadoop, para matrizes de ordem $N = 5000$ em dois ambientes de hardware, a máquina com oito núcleos (pseudo-distribuído) e o ambiente distribuído (*cluster*), tal como descrito na Seção 4. O resultado é apresentado no gráfico da Figura 8. O ambiente distribuído de execução (*cluster*) apresentou os melhores resultados, isto é, o menor tempo médio de execução para todas as estratégias. A melhoria em relação à execução no ambiente pseudo-distribuído fica em torno de 30% a 40%. Esse resultado indica a superioridade do Hadoop no ambiente distribuído e pode ser explicado pela contenção por acesso à memória na arquitetura do sistema *multicore*, que tem memória compartilhada.

A ordem de complexidade para a operação de multiplicação de matrizes é $\mathcal{O}(n^3)$. Devido ao pequeno número de processadores em relação à ordem das matrizes testadas, não há alteração na ordem de complexidade, embora o paralelismo tenha reduzido de forma significativa o tempo de execução final da multiplicação.

6. Conclusão

A multiplicação de matrizes é uma operação essencial em várias aplicações. O estudo de versões paralelas desta operação pode contribuir de forma genérica para melhorar os tempos de execução em vários contextos, especialmente considerando a ampla disponibilidade de ambientes de computação paralelos. Este artigo apresentou um estudo experimental de cinco implementações de programas paralelos para multiplicação de matrizes. Os resultados indicam que as implementações no ambiente Hadoop são marcadamente mais eficientes do que as implementações em Pthreads.

Além disso, a estratégia de descrição dos *jobs* no Hadoop influencia no resultado. A diferença dos tempos de execução entre as implementações Hadoop pode chegar a 20%. Um aspecto essencial é o ambiente de execução. A execução em *cluster* é mais eficiente do que a execução em uma máquina com vários núcleos.

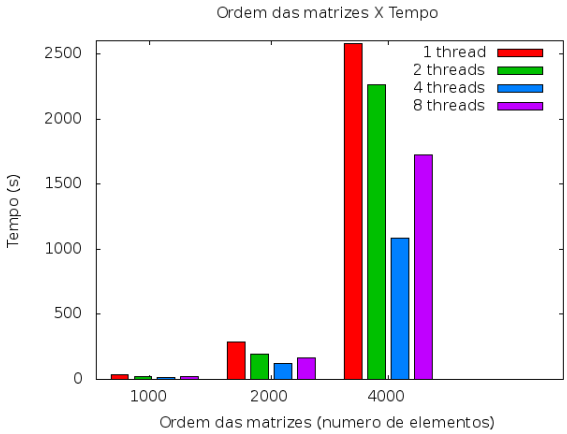


Figura 6. Tempos de execução em função dos tamanhos das matrizes e do número de threads para a implementação Pthreads

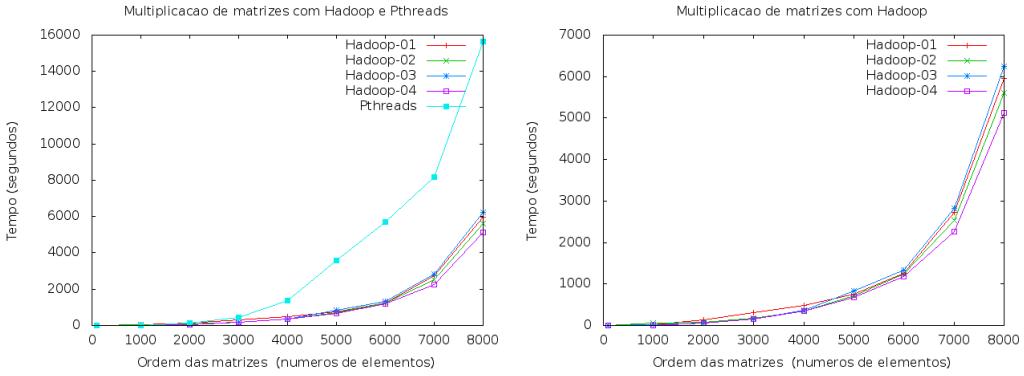


Figura 7. Tempos de execução em segundos em função dos tamanhos das matrizes nos ambientes Hadoop e Pthreads

Há várias direções para trabalhos futuros. Várias outras estratégias podem ser propostas. O estudo deve ser repetido para matrizes não quadradas, uma vez que observamos que algumas estratégias de implementação do Hadoop podem se beneficiar nestes casos. É necessário avaliar a escalabilidade das estratégias para matrizes maiores, em *clusters maiores*. Considerando que o estudo, a implementação e a análise de programas paralelos é de grande interesse na computação atual, sugerimos também o estudo de versões paralelas de outras classes de algoritmos, tais como algoritmos de ordenação, de busca e algoritmos em grafos.

Agradecimentos

Este trabalho foi parcialmente financiado pela FAPEMIG e pelo CNPq.

Referências

Asanovic, K., Bodik, R., Demmel, J., Keaveny, T., Keutzer, K., Kubiatowicz, J., Morgan, N., Patterson, D., Sen, K., Wawrzynek, J., Wessel, D., and Yelick, K. (2009). A View of the Parallel Computing Landscape. *Communications of the ACM*, 52(10):56–67.

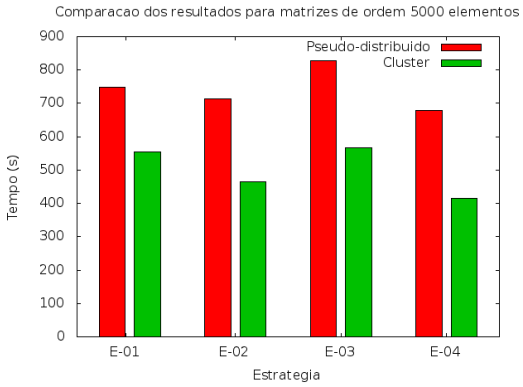


Figura 8. Tempos de execução em segundos para as estratégias Hadoop em dois ambienets de hardware

Dean, J. and Ghemawat, S. (2008). MapReduce: Simplified Data Processing on Large Clusters. *Communications of the ACM*, 51:107–113.

Dean, J. and Ghemawat, S. (2010). MapReduce:A Flexible Data Processing Tool. *Communications of the ACM*, 53(1):72–77.

Galon, S. and Levy, M. (2008). Measuring Multicore Performance. *IEEE Computer*, pages 99–102.

Hadoop (2011). Hadoop Users List. http://wiki.apache.org/hadoop/PoweredBy. Acesso em 2011.

Hennessy, J. and Patterson, D. (2007). *Computer Architecture - A Quantitative Approach (4th. Edition)*. Morgan Kaufmann Publishers.

Manferdelli, J. L., Govindaraju, N. K., and Crall, C. (2008). Challenges and Opportunities in Many-Core Computing. *Proceedings of the IEEE*, 96(5):808–815.

Nichols, B., Buttlar, D., and Farrell, J. P. (1996). *PThreads Programming - A POSIX Standard for Better Multiprocessing*. O'Reilly.

Norstad, J. (2009). A MapReduce Algorithm for Matrix Multiplication. `http://homepage.mac.com/j.norstad/matrix-multiply/index.html`. Acesso em julho de 2010.

Oskin, M. (2008). The Revolution Inside the Box. *Communic. of the ACM*, 51:70–78.

Ranger, C., Raghuraman, R., Penmetsa, A., Bradski, G., and Kozyrakis, C. (2007). Evaluating MapReduce for Multi-core and Multiprocessor Systems. In *Proceedings of the 2007 IEEE 13th International Symposium on High Performance Computer Architecture*, pages 13–24, Washington, DC, USA. IEEE Computer Society.

Sutter, H. and Larus, J. (2005). Software and the Concurrency Revolution. *ACM Queue*, 3:54–62.

White, T. (2009). *Hadoop: The Definitive Guide*. O' Reilly.