Paralelização do Software Weka em Arquitetura *Multicore*: uma Abordagem Inicial

Tiago Augusto Engel[1], **Andrea S. Charão**[2], **Ana Trindade Winck**[2]

[1]Programa de Educação Tutorial – Ciência da Computação
Universidade Federal de Santa Maria – UFSM

[2]Departamento de Eletrônica e Computação
Universidade Federal de Santa Maria – UFSM

{tengel, andrea, ana}@inf.ufsm.br

Resumo. *Neste trabalho, explora-se uma abordagem para a paralelização do software Weka. A partir do profiling do programa, escolheu-se um de seus métodos para ser paralelizado. A implementação paralela, usando threads, foi incorporada ao Weka. Os resultados indicam uma melhora no desempenho em relação ao software original.*

Abstract. *This paper explores a approach to the parallelization of the Weka software. From the profiling of the program, a method was chosen to be parallelized. The parallel implementation, using threads, was merged to the Weka. The result shows improvement in relation to the original software.*

1. Introdução

A descoberta de padrões em grandes conjuntos de dados, utilizando técnicas de mineração de dados, muitas vezes tem alto custo computacional. Existem algoritmos que podem demorar horas para chegar a resultados, dependendo do conjunto de dados, tornando a atividade exaustiva.

O Weka [Witten et al. 1999] é um software mantido pela Universidade de Waikato. Ele contém uma coletânea de algoritmos para filtragem, classificação, regressão, agrupamento, regras de associação e visualização de dados. Esses algoritmos são combinados para formar um dos mais conhecidos pacotes de mineração de dados da atualidade. Ele é escrito totalmente em Java e é licenciado sob a licença GNU (*General Public License*).

Este trabalho visa encontrar meios de melhorar o desempenho do Weka utilizando técnicas de programação paralela em arquitetura *multicore*. O trabalho é dividido em três seções principais: a seção 1 discute o *profiling* do Weka, mostrando como foi escolhido o alvo inicial da paralelização – multiplicação de matrizes; a seção 2 discute o algoritmo que foi implementado e incorporado no Weka; a seção 3 analisa o desempenho tanto do algoritmo de multiplicação de matrizes quanto do Weka com a nova implementação.

2. *Profiling* do Weka

Existem vários *profilers* conhecidos para Java. Cada um deles possui uma maneira de aproximação para a coleta de informações, sendo que os resultados podem diferir entre eles [Mytkowicz et al. 2010]. Por conta disso, no presente trabalho realizou-se o *profiling*

com mais de uma ferramenta. As ferramentas, escolhidas dentre as mais populares, foram: VisualVM, JProfiler, JProbe e o *profiler* embutido do Java.

Para a identificação de pontos do programa que demandam alto custo computacional, escolheu-se um caso utilizando o algoritmo M5P [Wang and Witten 1997] do Weka, que visa à construção de árvores de decisão usadas em tarefas de regressão. O dataset utilizado possui um total de 269 atributos contínuos e 11284 instâncias.

As ferramentas utilizadas mostraram-se intrusivas. O VisualVM[1] apresentou-se excepcionalmente muito intrusivo. O tempo de *profiling* se tornou totalmente inviável para efetuar qualquer avaliação. O JProfiler[2] se comportou melhor, mas ainda com um alto tempo de execução. Pôde-se identificar os métodos que mais custaram a ser processados e as estatísticas dos mesmos. O JProbe[3] comportou-se de maneira menos intrusiva em relação aos outros. Os métodos apontados por ele como mais custosos coincidiram com os do JProfiler. O *profiler* que vem com a distibuição Java se mostrou pouco intrusivo, porém seus resultados foram superficiais, servindo como complemento.

Quando alguns parâmetros do algoritmo foram variados, observou-se uma pequena discordância entre alguns *hotspots*, mas com a análise dos resultados pôde-se convergir para um pequeno conjunto de métodos. Em um primeiro momento deu-se prioridade para a paralelização de métodos que são de uso geral (não usados apenas no M5P). Os principais métodos são mostrados na tabela 1.

Tabela 1. Métodos de uso geral identificados como hotspots no Weka

Método	Uso
weka.core.matrix.LinearRegression.calculate	Cálculo de correlação
weka.core.Instances.quickSort	Ordenação de instâncias
weka.core.matrix.Matrix.times	Multiplicação de matrizes
weka.classifiers.functions.LinearRegression.findBestModel	Busca de modelo de regressão
weka.core.matrix.LUDecomposition.solve	Resolução de sistema linear
weka.classifiers.Evaluation.evaluateModel	Avaliação de modelos

O primeiro método escolhido para ser paralelizado foi o de Multiplicação de Matrizes (*weka.core.matrix.Matrix.times*). A escolha se deu pelo seu alto número de chamadas e o tempo de processamento que consome. Ao estudar as chamadas do método de multiplicação de matrizes do M5P, pôde-se observar que o caso mais rápido estudado teve 1554 chamadas e, no caso mais demorado, observaram-se 53052 chamadas do método. Na seção seguinte, descreve-se a paralelização deste algoritmo e sua implementação feita para o Weka.

3. Multiplicação Paralela de matrizes

A paralelização da multiplicação foi desenvolvida da seguinte forma: sejam A e B duas matrizes, t o número de *threads* e busca-se a matriz $X = A * B$, então dividem-se as colunas da matriz B em partições de tamanho $p = \frac{numerodecolunasdeB}{t}$.

[1]http://visualvm.java.net
[2]http://www.ej-technologies.com/products/jprofiler/overview.html
[3]http://www.quest.com/jprobe/

Deste modo, tem-se as colunas da matriz B divididas em partições de p elementos. Faz-se, então, cada *thread* t_i multiplicar a matriz A pela sub-matriz formada pelas colunas $C(p * i)$ até $C(p * i) + p$, $1 <= i <= t$ de B. Assim, tem-se cada *thread* processando p colunas da matriz B e não existem conflitos para escrita de dados na matriz resultado X. O algoritmo construído é semelhante a outros encontrados na literatura [Barry 2006].

O algoritmo foi implementado utilizando os recursos disponíveis para concorrência na biblioteca do Java. Usou-se o pacote `java.util.concurrent.*`, que provê métodos para criar, agupar e sincronizar threads.

4. Resultados

Primeiramente, avaliou-se o algoritmo paralelo separadamente, para depois incorporá-lo ao Weka. Todos os experimentos foram feitos em um computador com processador Intel(R) Xeon(R), com 8 núcleos de 2.4 GHz cada. Nas subseções seguintes são apresentados os resultados obtidos de cada etapa de avaliação.

4.1. Algoritmo Paralelo de Multiplicação de Matrizes

Para este teste, foram utilizadas matrizes quadradas. Os testes foram feitos com 1, 2, 4, 6 e 8 threads, com matrizes variando de 100x100 até 1000x1000 elementos. O teste de assertividade foi feito comparando-se a matriz resultante da multiplicação paralela e a da multiplicação serial usando a JUnit[4].

O *speedup* obtido da relação entre a multiplicação de matrizes serial e em paralelo foi considerável, como pode-se observar na figura 1. Notou-se, no entanto, que o *speedup* distanciou-se do ideal para casos com 6 e 8 threads, o que precisa ser melhor investigado.

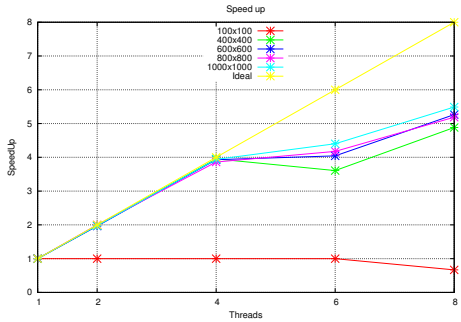

Figura 1. Aceleração (*speedup*) da multiplicação de matrizes

4.2. Weka com o Algoritmo de Multiplicação em Paralelo

A incorporação da multiplicação paralela de matrizes para o Weka levou em consideração as matrizes utilizadas. Constatou-se que o M5P utiliza, predominantemente, matrizes de tamanhos retangulares. Levando isto em consideração, criou-se um mecanismo para que, quando a matriz for pequena, o algoritmo utilize a multiplicação sequencial. Esse mecanismo minimiza o *overhead* causado pela criação e manutenção das threads e permite que

[4]http://www.junit.org

os algoritmos que utilizam a multiplicação de matrizes façam-na paralelamente quando a matriz for grande.

A implementação conseguiu melhorar consideravelmente o desempenho do M5P, como pode-se observar na figura 2. Para este teste, foi fixado número de threads em 4 e variou-se a quantidade de dados de processamento. Quanto mais alta a árvore, mais processamento o algoritmo demanda. Então, quando tem-se a maior altura possível da árvore, a melhoria de desempenho é mais significativa. Por outro lado, com a altura da árvore diminuída, observa-se uma melhoria pouco considerável de desempenho.

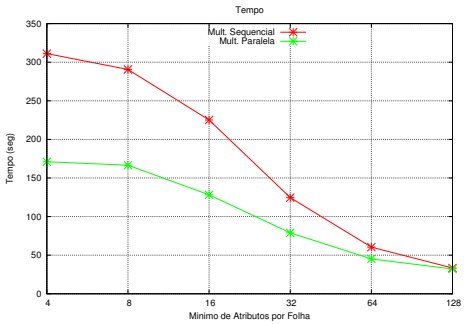

Figura 2. Desempenho do Weka com o Algoritmo Paralelo

5. Conclusão e Trabalhos Futuros

A multiplicação de matrizes em paralelo mostrou-se útil na melhoria do desempenho do Weka, devido ao amplo uso desta operação e ao *speedup* obtido. Acredita-se que o potencial de melhoria do sofware em arquitetura multicore seja grande e possa ser explorado para outros algoritmos.

Esse artigo serve como base para o estudo e implementação paralela do Weka. Os métodos citados como *hotspots* servem como norteadores para a continuação da paralelização do software. Estudos para avaliar outros métodos são necessários para a descoberta de novos pontos onde é possível a melhoria de desempenho.

Referências

Barry, W. (2006). *Parallel Programming: Techniques And Applications Using Networked Workstations And Parallel Computers, 2/E*. Pearson Education.

Mytkowicz, T., Diwan, A., Hauswirth, M., and Sweeney, P. F. (2010). Evaluating the accuracy of Java profilers. In *Proceedings of the 2010 ACM SIGPLAN conference on Programming language design and implementation*, PLDI '10, pages 187–197, New York, NY, USA. ACM.

Wang, Y. and Witten, I. H. (1997). Induction of model trees for predicting continuous classes. In *Poster papers of the 9th European Conference on Machine Learning*. Springer.

Witten, I. H., Frank, E., Trigg, L., Hall, M., Holmes, G., and Cunningham, S. J. (1999). Weka: Practical machine learning tools and techniques with Java implementations.