Paralelização do algoritmo DIANA em OpenMP

Hethini Ribeiro[1], Roberta Spolon[1], Aleardo Manacero Jr.[2], Renata S. Lobato[2]

[1]Departamento de Computação
Universidade Estadual Paulista "Júlio de Mesquita Filho" (UNESP) – Bauru,SP – Brazil

[2]Departamento de Ciências da Computação e Estatística
Universidade Estadual Paulista "Júlio de Mesquita Filho" (UNESP)
São José do Rio Preto, SP – Brazil

hethini.ribeiro@outlook.com, roberta@fc.unesp.br

Abstract. *Global data production has increased by approximately 40% per year at the beginning of the last decade. These large datasets, also called Big Data, are posing unavoidable challenges in many areas and in particular in the Machine Learning (ML) field. ML algorithms are able to extract useful information from large data repositories, but these applications are computationally expensive, such as hierarchical algorithms AGNES and DIANA, which have O (n) and O (2n) complexity, respectively. So the big challenge is to process large amounts of data in a realistic time frame. In this context, it is proposed the parallelization of the DIANA OpenMP algorithm.*

Resumo. *A produção global de dados aumentou aproximadamente 40% ao ano no início da década passada. Esses grandes conjuntos de dados, também chamados de Big Data, estão colocando desafios inevitáveis em muitas áreas e, em particular, no campo de Machine Learning (ML). Algoritmos de ML são capazes de extrair informações úteis de grandes repositórios de dados, porém essas aplicações são dispendiosas computacionalmente, como por exemplo os algoritmos hierárquicos AGNES e DIANA, que por sua vez, possuem complexidade O (n) e O (2n) respectivamente. Sendo assim, o grande desafio consiste em processar grandes quantidades de dados em um período de tempo realista. Nesse contexto, propõe-se a paralelização do algoritmo DIANA OpenMP.*

1. Introdução

Devido à diversidade e grande quantidade de dados produzida no mundo, a analise destes fica praticamente impossível de se realizar dentro de um tempo razoável para determinadas aplicações. Existe uma grande pressão para entender, sintetizar e tomar decisões em praticamente todos os tipos de dados [BELL 2015]. Com a produção de dados crescendo surge a necessidade de máquinas capazes de analisar e entender o conjunto de dados. Neste contexto os algoritmos de aprendizado de máquina, ou *Machine Learning* (ML), desempenham um papel central na análise. A produção sem precedentes de grandes quantidades de dados de alta dimensão aumenta consideravelmente a complexidade das tarefas de análise desses algoritmos. Outro ponto a ser levado em consideração é a complexidade computacional das metodologias de ML. Isto é um fator limitante que pode tornar a aplicação de muitos algoritmos impraticáveis quando estes trabalham com grandes conjuntos de dados [LOPES and RIBEIRO 2015]. Sendo assim, a paralelização

destes amplia a aplicabilidade dos algoritmos existentes à conjuntos de dados maiores e mais complexos, uma vez que estes se adequam a volume de dados grande devido ao processamento veloz. Este documento tem como proposta otimizar o algoritmo DIANA de ML com classificação hierárquica através do uso de paralelismo com OpenMP.

2. Trabalhos Relacionados

Já existem estudos relacionando a paralelização com algoritmos de classificação e agrupamento, como por exemplo, o trabalho feito em [DANALIS et al. 2012] no qual os autores paralelizaram o algoritmo *Quality Threshold Clustering* com um modelo híbrido de MPI e OpenMP e tiveram um *speedup* de 35x. Em [BHIMANI et al. 2015] os autores otimizaram o algoritmo K-means usando CUDA, MPI e OpenMP. Vários testes foram realizados com dois dados de entrada: uma imagem de 300x300 pixels e outra 1164x1200 pixels. No experimento o OpenMP teve melhor desempenho com imagens pequenas e médias enquanto CUDA teve melhor performance com as maiores. Em todos os casos o processamento paralelo teve melhor desempenho que o sequencial. A paralelização de algoritmos de classificação e clusterização vem se mostrado uma área válida de exploração. No entanto, nem todos os algoritmos foram otimizados por esta técnica ou foram pouco estudados.

3. Paralelização do algoritmo DIANA

O algoritmo DIANA consiste em analisar dados e, através de padrões encontrados usando distância euclidiana, são criadas subclassificações para os objetos. Essa categorização ocorre de maneira a dividir grupos maiores em menores. Nesse contexto, a ideia do algoritmo é construir uma matriz de distâncias que contém o intervalo de todos os pontos para todos os outros, conforme a Figura 1 [JOHNSON 1967].

d	1	2	3	4	5	6
1	0	0.31	0.23	0.31	0.23	0.23
2	0.31	0	0.31	0.23	0.31	0.31
3	0.23	0.31	0	0.31	0.4	0.7
4	0.31	0.23	0.31	0	0.31	0.31
5	0.23	0.31	0.4	0.31	0	0.7
6	0.23	0.31	0.7	0.31	0.7	0

Figura 1. Um exemplo de uma matriz de distância.

Como pode ser observado na Figura 1, a matriz formada é do tipo simétrica, ou seja, os elementos opostos em relação a linha diagonal principal são idênticos. Com essa característica parte do trabalho de análise pode ser reduzido, pois apenas um dos lados – acima ou abaixo da linha diagonal – pode ser utilizado para o cálculo. Para a distribuição das distancias de um dos lados da matriz é necessário determinar quantas distancias serão analisadas. Para isto utiliza-se a seguinte formula:

$$Total = \frac{(QtdPontos^2) - QtdPontos}{2}$$

Com a quantidade de dados determinada, o total, por sua vez, é divido entre o número de threads. Ou seja, para a tabela apresentada na Figura 1 o total de amostras a

serem analisadas é 15. Em um ambiente com duas threads, cada uma analisará metade do conjunto. A distribuição é feita através de uma diretiva conforme o exemplo apresentado pela Figura 2, na qual a variável "posA" é utilizada para caminhar na matriz de distâncias e as iterações do laço são distribuídas em relação ao número da variável "chunk" e MAX é a constante com o número total de pontos, ou seja, para este exemplo seu conteúdo é 5000. O valor de "chunk" é definido através da formula 1, apresentada anteriormente, dividida pelo número de threads no utilizados naquele ambiente (Figura 3).

```
#pragma omp for schedule (static, chunk)
for(posA=0; posA< MAX; posA++)
```

Figura 2. Exemplo do uso da diretiva de paralelização de laços em OpenMP.

```
chunk = (((MAX*MAX)-MAX)/2)/nthreads;
```

Figura 3. Cálculo da variável "chunk" utilizada na distribuição das iterações do laço de repetição entre os threads.

É interessante que os threads não acessem os dados que os outros possuam por limitar ao máximo a comunicação entre eles, gerando maior desempenho. No entanto, quando um ponto é classificado, a atualização da informação, embora possa ser feita por todos os threads, ocorre de maneira sequencial. Isso se dá por ser uma região crítica, onde pode vir a acontecer de todos os processos acessarem a mesma região de memória ao mesmo tempo. Por esse motivo, um deve esperar que o outro termine sua operação para poder iniciar a sua. Essa região do programa é um limitador do paralelismo.

4. Resultados e Conclusões

Até o momento da finalização deste documento, foi desenvolvido uma versão em OpenMP do algoritmo DIANA utilizando o método *Single Linkage* – utilizando a distância Euclidiana para encontrar padrões entre as amostras – e testado em uma máquina com processador AMD FX-8320, memória Ram de 8 Gb e sistema operacional Ubuntu 16.04.2 x64. Para o programa desenvolvido neste trabalho, mediu-se o tempo de execução do trecho sequencial (19,65%) e paralelo (80,35%) em um único *thread*. Com essa informação determinou-se a porcentagem das porções sequenciais e paralelas do código para uma base de dados com 5000 pontos, permitindo deste modo o cálculo do ganho de velocidade esperado para o trabalho desenvolvido. O primeiro cálculo é referente a lei de Amdahl, na qual estuda-se o desempenho para problemas de tamanho fixo. Esta foi determinada pela Equação 2, onde N é o número de processadores:

$$Speedup = \frac{1}{0,1965 + (\frac{0,8035}{N})}$$

A segunda lei estudada foi Gustafson (Equação 3). Esta, assim como a lei de Amdahl, também prevê o desempenho do algoritmo, porém esta leva em consideração a variação do tamanho do problema.

$$Speedup = N - 0,1965.(N - 1)$$

O programa desenvolvido para este estudo foi testado três vezes para a mesma base de dados com 5000 pontos. O tempo para a análise e classificação dos elementos é descrito na Figura 4.

Tempo em segundos para 5000 elementos			
Nº de *threads*	Tempo	Tempo	Tempo
1	1412	1406	1415
2	423	429	426
4	314	310	311
8	299	299	299
16	299	300	299
32	298	299	297
64	297	296	296
128	296	297	295
256	291	292	293
512	283	285	283
1024	266	270	270

Figura 4. Tempo em segundos para classificar 5000 pontos para o aumento da quantidade de threads.

Observa-se um grande ganho de tempo ao comparar qualquer resultado paralelo com o monoprocessado, chegando a um *speedup* médio de 5,2521. Outro parecer sobre a Figura 4 é a estabilização do ganho após 8 *threads*, ou seja, para uma base de dados com 5000 elementos, a adição de mais recursos computacionais a partir de 8 *threads* deixa de ser vantajosa. Isso se dá pela limitação do trecho sequencial no código – atualização dos pontos, efetivando a relação de pertinência destes com os clusters. Pela Lei de Amdahl, esperava-se um ganho de 5,0694, ou seja, o programa desenvolvido teve um desempenho melhor do que o previsto. Já para a Lei de Gustafson calculava-se 823,004. Como esta prevê maiores ganhos para problemas maiores, acredita-se que para bases de dados com maior volume de dados o ganho esperado seja alcançado. Para tal, uma base disponibilizada pelo departamento de computação da Universidade da Finlândia Oriental foi escolhida. A base possui 13.467 localizações de usuários do aplicativo MOPSI na Finlândia [FRÄNTI et al. 2014]. Ainda para propõe-se o desenvolvimento de uma versão em CUDA para este algoritmo para comparação de desempenho.

Referências

BELL, J. (2015). *Machine Learning: Hands-On for Developers and Technical Professionals*. John Wiley Sons.

BHIMANI, J., LEESER, M., and MI, N. (2015). *Accelerating K-Means Clustering with Parallel Implementations and GPU computing.* In: High Performance Extreme Computing Conference (HPEC).

DANALIS, A., MCCURDY, C., and VETTER, J. S. (2012). Efficient quality threshold clustering for parallel architectures.

FRÄNTI, P., REZAEI, M., and ZHAO, Q. (2014). Centroid index: cluster level similarity measure. pattern recognition.

JOHNSON, S. (1967). Hierarchical clustering schemes. Psychometrika.

LOPES, N. and RIBEIRO, B. (2015). Machine learning for adaptive many-core machines: A practical approach. Springer.