Um estudo do algoritmo Agglomerative Clustering em diferentes ferramentas de paralelização

Juan Rios[1], **Edevaldo Santos**[1], **Gerson Geraldo H. Cavalheiro**[1]

[1] Universidade Federal de Pelotas – (UFPEL)
Pelotas – RS – Brasil

Resumo. *Este trabalho documenta um experimento de programação paralela com ferramentas para programação multithread. É realizada a implementação de um algoritmo paralelo, o Agglomerative Clustering, em OpenMP, Cilk Plus, Threading Building Blocks e xKaapi. Diferenças entre as implementações e desempenhos obtidos documentam os resultados do estudo realizado.*

1. Introdução

Com o avanço e a demanda cada vez maior em processamento de dados, a área de processamento de alto desempenho tem se tornado uma solução viável para suprir essa necessidade. Então passou-se a estudar e investir em técnicas onde exploram o hardware para extrair o máximo proveito dos recursos disponíveis. Multithreading é um exemplo dessas técnicas, onde ela consiste na exploração do hardware voltada ao uso de múltiplas unidades de processamento, ou cores. Diversas ferramentas oferecem recursos para programação multithreading, como por exemplo: OpenMP, Threading Building Blocks, Cilk Plus e xKaapi. Em comum, estas ferramentas exploram os recursos de processamento disponível por meios de estratégias de escalonamento, visando obter melhores índices de desempenho. A seguir será demonstrado uma breve introdução destas ferramentas:

***OpenMP*:** Uma das mais conhecidas e referenciada na área de programação paralela, desenvolvida em 1997 vem sendo atualizada até os dias atuais. Dispõe de diversas funções, bem com diretivas de compilação que delimitam o bloco a ser paralelizado pela aplicação, também sendo capaz de modificar o número de threads a ser lançadas, com o uso de variáveis ambientes, antes e durante o processo de execução da aplicação. Sua estrutura de escalonamento é baseada no estilo fork-join aninhado.

***Threading Building Blocks* (TBB):** Desenvolvida pela Intel com suporte a C++, é uma das ferramentas que possui uma alta escalabilidade e um desempenho satisfatório em determinadas aplicações. Possuindo diferentes metódos de tratamento para regiões críticas ou até mesmo loops.

***Cilk Plus* :** Baseada em C (Cilk) e C++, é uma ferramenta simples de paralelização onde conta com construtores para expressar loops em paralelo. Seu escalonamento também é baseado em fork-join aninhado.

***xKaapi* :** Desenvolvida pelo time MOAIS (INRIA Rhône-Alpes) utiliza da libKOMP, um runtime OpenMP baseado na biblioteca xKaapi, desenvolvida em C. Possuindo uma grande gama de funções para desenvolvimento de aplicações paralelas ou até mesmo clusters.

O uso de APIs na construção de um sistema é extremamente comum nos dias atuais, pois com ela o programador pode usar apenas as funcionalidades de uma determinada aplicação sem entrar em maiores detalhes de implementação.

O objetivo desse trabalho é apresentar uma comparação entre as ferramentas estudas nesse trabalho utilizando do benchmark *Agglomerative Clustering*, para testar seus desempenhos em diferentes situações.

2. Desenvolvimento

Como um dos algoritmos de data mining, o Agglomerative Clustering consiste em uma entrada com um dataset de pontos contidos em um espaço n-dimensional e uma função de similaridade entre os itens do dataset. Esta função é utilizada para medir a distância entre os pontos, quanto mais similar dois elementos são mais próximos eles estão. A saída do algoritmo é uma árvore binária, chamada dendograma, representando um grupamento hierárquico de pares de elementos do dataset. Sua implementação é dada de forma que se dois pontos concordarem que são vizinhos, seus país próximos são agrupados na árvore final. Então com o estudo do algoritmo, desenvolveu-se as seguintes versões:

- **OpenMP**: Em OpenMP foi utilizado da diretiva `omp parallel` para criação da região paralela do código. Então é chamado uma função que busca o ponto mais próximo para verificar se é possível agrupá-lo com o ponto escolhido anteriormente e então caso a afirmação seja correta, junta-se os pontos para que formem um novo cluster. Para o laço de repetição das funções de busca do ponto mais próximo e da junção deles, foi utilizado do `parallel for shared` e compartilhando a variável auxiliar `clusters_aux` para que os demais processadores tenham acesso a essa variável para o uso em suas zonas críticas, então tratando elas com a diretiva `pragma omp critical`. Para a remoção de possíveis repetições foi utilizado a diretiva `pragma omp barrier` para que não seja executado outra thread fora da barreira e então executará uma função para remoção dessas repetições.
- **TBB**: Em TBB foi utilizado de uma estrutura previamente criada chamada `clusterize`, onde ele consiste de uma função `operator()` para manipulação da estrutura (achar o ponto mais próximo, uni-los e etc) essa função é chamada dentro do `parallel_for`. Para o laço de repetição foi utilizado o `tbb::parallel_for`, onde o template recebe como seu primeiro argumento o range de valores que ele irá percorrer, cuidando assim de possíveis regiões críticas assim como sua função/classe a ser executada em paralelo. E então, por fim é chamada uma função de remoção de repetições, caso necessário.
- **xKaapi**: Em xKaapi, foi utilizado a API para conexão entre o código e as bibliotecas do xKaapi para essa interação é utilizado um shell script, onde sua entrada é um código objeto desenvolvido em OpenMP. Dessa forma ele faz uma chamada as biblotecas do xKaapi, a libKOMP, para que possa ser entendido pelo runtime da ferramenta, por meio da utilização das shared objects da libKOMP ao iniciar o programa e então executa-lás.
- **Cilk Plus**: Em Cilkplus foi utilizado o metódo `cilk_for` para a criação da região paralela no código. Similar a versão do TBB, porém um pouco mais simples devido não precisar de uma estrutura para manipulação. Com isso, havendo a necessidade de cuidar de possíveis regiões críticas. Ele chama também uma função

para achar o ponto mais próximo e então criar um novo cluster. Para as regiões críticas do código foi utilizado do mutex, logo após é chamado uma função que remove repetições.

3. Resultados

Para o desenvolvimento dos experimentos no algoritmo foi utilizado o computador com as seguintes configurações: processador 4x Intel Core i5 4690 @3.50GHZ, memória de 7,7GB de RAM e sistema operacional Ubuntu 16.04 LTS. Para a compilação, foi utilizado o compilador G++ com as bibliotecas de OpenMP, TBB, Cilk Plus e xKaapi previamente instaladas, e as seguintes flags de compilação: -fopenmp, -fcilkplus e -ltbb.

Foram realizados testes, onde cada teste consistia na execução do algoritmo com uma determinada quantia de pontos, variando também o número de threads a ser utilizado. Começando então com 10 pontos, depois 100 e por fim 1000. A variação de threads foi de 1, 2, 4, 8 e 16 threads em todas as versões, e para cada uma dessas variações o algoritmo foi executado 10 vezes. Então com a realização dos experimentos foi construído um gráfico, afim de uma análise do algoritmo tomando como base de tempo a execução do algoritmo em sua versão sequêncial. O gráfico da Figura 1 de tempo de execução do sequencial por tempo de execução em paralelo.

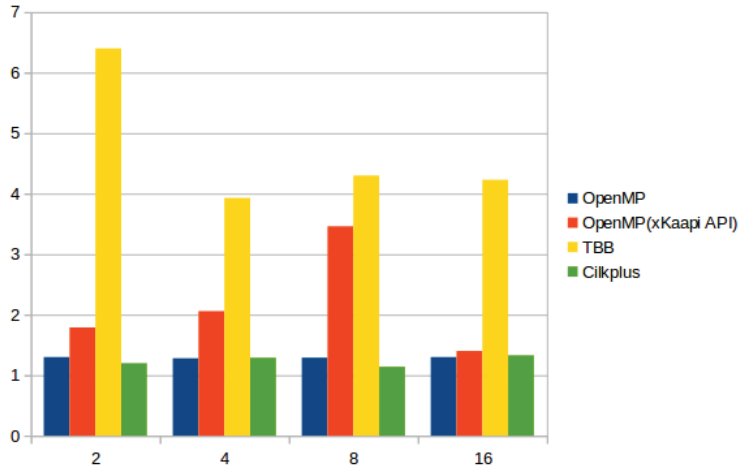

Figura 1. Speedup entre as diferentes versões

Como pode ser observado, com a variação do número de threads utilizadas ao longo da execução do programa ele demonstra aumento de ganho relativo ao tempo sequencial de execução do mesmo. Conforme o número limite de threads é alcançado, ao tentar aumentar ainda mais esse número o algoritmo começa a perder seu desempenho, devido a ter atingindo a capacidade máxima do computador. A versão em TBB teve um ganho melhor se comparado as demais versões, ainda que a versão do xKaapi tenha mostrado um bom desempenho quando utilizado oito threads. Esse ganho de TBB é explicado em função de sua região paralela não quebrar exaustivamente todos os chunks possíveis, ao contrário do que ocorre nas demais versões. Em TBB, a técnica de divisão e conquista de quebra de laço só gera novas tarefas quando somente um processador fica sem trabalho, assim não gera tarefas excessivas para um processador, ou até mesmo não deixando um processador ocioso.

4. Conclusão e Continuidade

Existem diversas ferramentas de programação multithread, possuindo interfaces distintas e promovendo diferentes índices de desempenho dependendo da aplicação. Concluí-se que após a execução dos experimentos, mesmo que haja uma capacidade grande de processamento por parte do hardware disponível, ele pode ser limitado devido a maneira como é utilizado a técnica que foi implementada. Embora TBB seja uma ferramenta que demonstrou ter um desempenho superior as demais, é uma ferramenta que exige um melhor conhecimento sobre ela, não sendo de maneira trivial o seu entendimento. Ferramentas como OpenMP, Cilk Plus possuem uma maneira mais simples de transformar uma aplicação sequencial em paralela, contando com funções mais simples, porém, dando um poder de abstração um pouco menor. O problema na resolução do algoritmo torna-se muito grande quando é aumentado o número de pontos a ser inseridos na árvore, uma vez que o algoritmo tem de percorrer um número maior de ramos para então encontrar seus vizinhos, gastando um bom tempo até que a árvore final seja construída. Como continuidade do trabalho é previsto a implementação de um maior número de aplicações para entendimento da adequação das diferentes ferramentas, a diferentes classes de problemas, em função do desempenho e das facilidades de programação oferecidas pela API.

Referências

Gautier, T. (2015) ”The xKaapi Runtime”, http://www.gipsa-lab.grenoble-inp.fr/thematic- school/gpu2015/presentations/GIPSA-Lab-GPU2015-T-Gautier.pdf.

Gautier, T. Lima, J. Maillard, N. e Ran, B. (2013) “xKaapi: A Run time System for Data-Flow Task Programming on Heterogeneous Architectures”.

SAAVEDRA, R. H. e SMITH, A. J. (2008) ”Analysis of benchmark characteristics and benchmark performance prediction”. Computer Science Technical Report UCB/CSD 92/715, UC Berkeley. p. 344-384.

Broquedis, F. Gautier, T. Danjean, V. (2012) ”an Efficient OpenMP Runtime System for Both Fork-Join and Data Flow Paradigms”. 8th International Workshop on OpenMP, Rome, Italy, p. 114-118.

PAS, R. V. (2005) ”An Introduction to OpenMP”, http://www.nic.uoregon.edu/iwomp2005/iwomp2005_tutorial_openmp_rvdp.pdf.

Intel. ”Intel TBB Tutorial”, https://www.threadingbuildingblocks.org/intel-tbb-tutorial.

Intel. ”Cilk Plus Tutorial”, https://www.cilkplus.org/cilk-plus-tutorial.