Ordenação distribuída de pares chave-valor utilizando MPI

Rodrigo Morante Blanco[1], **Michel B. Cordeiro**[1], **Wagner M. Nunan Zola**[1]

[1]Departamento de Informática – Universidade Federal do Paraná (UFPR)

rodrigomorante@gmail.com, michel.brasil.c@gmail.com, wagner@inf.ufpr.br

Resumo. *Neste trabalho propõe-se um algoritmo de ordenação distribuída de pares chave-valor utilizando MPI. Os resultados obtidos são comparados com a ordenação sequencial, mostrando que a versão distribuído é capaz de alcançar aceleração de 3.05 em relação à versão sequencial.*

1. Introdução

A ordenação é um componente importante de muitas aplicações paralelas. Por exemplo, o desempenho do método de Barnes-Hut (utilizado para achar soluções aproximadas do problema dos n-corpos) melhora ao calcular as forças consecutivamente sobre corpos próximos entre si, o que é feito ordenando os corpos segundo suas chaves de Morton.

A ordenação distribuída pode ser definida da seguinte forma: Seja p o número de processos, e um conjunto de n pares chave-valor a serem ordenados. Cada processo deve ordenar um subconjunto de n/p pares (ch_i, val_i), onde ch_i representa a chave do i-ésimo par e val_i representa um valor associado a essa chave. O objetivo deste trabalho é propor um algoritmo eficiente de ordenação distribuída utilizando MPI.

2. Descrição do Algoritmo

A ordenação distribuída é realizada nas fases descritas a seguir: Supõe-se que cada processo possui seu subconjunto de n/p elementos da entrada. Se n não é múltiplo de p alguns processos contêm um elemento a mais do que outros. Uma vez ordenados esses subconjuntos locais, os processos intercambiam elementos utilizando `MPI_Alltoall` da seguinte forma: Seja $\Delta = (\max_{\text{global}} - \min_{\text{global}})/p$. Os processos de índices $i = 0, \ldots, p-2$ receberão os elementos de todos os processos cujas chaves estão nos intervalos semi-abertos $[\min_{\text{global}} + i\Delta, \min_{\text{global}} + (i+1)\Delta)$. O último processo receberá os elementos de todos os processos cujas chaves estão no intervalo fechado $[\min_{\text{global}} + (p-1)\Delta, \max_{\text{global}}]$. Cada processo então decide quais elementos serão enviados ao restante dos processos. Esta informação também é compartilhada utilizando `MPI_Alltoall` para poder reservar o espaço necessário. Finalmente os elementos são efetivamente enviados utilizando `MPI_Alltoallv`. Por fim, as listas parciais recebidas por cada processo são combinadas utilizando um *k-way merge*. Desta forma, a lista final de cada processo estará ordenada.

3. Metodologia e Resultados

Foram conduzidos experimentos para avaliar a escalabilidade do algoritmo proposto. Para isso, foi gerado um conjunto de dados contendo 32 milhões de pares chave-valor, com chaves do tipo unsigned long long (64 bits) e valores do tipo unsigned int (32 bits). Também foram realizados experimentos para analisar o tempo de execução de cada etapa do algoritmo, considerando conjuntos com 8 milhões e 32 milhões de elementos. Os testes foram executados 20 vezes, e a vazão média de milhões de pares processados por segundo

(MPPS) foi reportada. Além disso, foram calculados intervalos de confiança de 95%. No entanto, como as variações em relação à média não ultrapassaram 2%, esses valores não foram reportados. Os experimentos foram executados em um *cluster* computacional de 4 nodos, cada um com dois processadores Intel(R) Xeon(R) E5462 @ 2.80GHz, 32 GB de RAM, rodando Linux com distribuição Ubuntu 20.04. A versão de `gcc` utilizada foi 9.4.0 e a versão do Open MPI utilizada foi 4.0.3. A rede que conecta os nodos é Gigabit Ethernet. O método descrito foi implementado em C++, sendo utilizado MPI nas fases que envolvem comunicação entre processos e `std::sort` na ordenação local.

O gráfico à esquerda da Figura 1 apresenta os resultados para escalabilidade forte, isto é, mantendo a mesma quantidade de elementos (32×10^6 pares de chaves-valores) e aumentando o número de processos. Observa-se que o desempenho do algoritmo melhora à medida que o número de processos aumenta, alcançando uma aceleração de 3.05 com a utilização de 8 processos, distribuídos em 4 nodos. Além disso, o método distribuído é mais rápido em todos os casos, embora a aceleração é sempre sub-linear. No gráfico à direita, é possível observar que a etapa de comunicação domina o tempo de execução do algoritmo. Esse comportamento pode ser explicado pela latência e pela vazão da rede Gigabit Ethernet, o que sugere que o desempenho do algoritmo poderia ser melhorado com o uso de uma rede mais rápida.

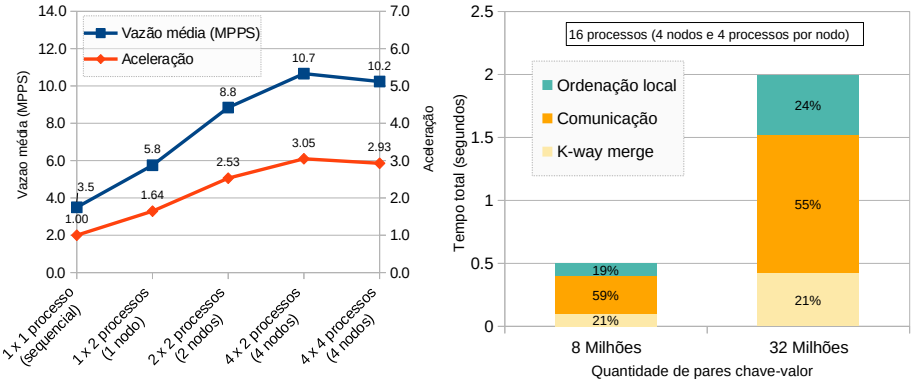

Figura 1. (Esquerda) Escalabilidade forte do método de ordenação distribuída para 32×10^6 pares de chave-valor (chaves de 64 bits e valores de 32 bits). (Direita) Gráfico de tempos de execução para cada etapa do algoritmo.

4. Conclusões

Neste trabalho foi implementada um algoritmo de ordenação distribuída. No entanto, se a distribuição dos elementos não é normal, a divisão do trabalho entre os processos pode ficar desbalanceada. Para melhorar este resultado deve ser utilizada uma forma mais sofisticada de balanceamento de carga dos elementos a serem distribuídos, como a proposta em [Siebert and Wolf 2011].

Agradecimentos

Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) - Programa de Excelência Acadêmica (PROEX).

Referências

Siebert, C. and Wolf, F. G. E. (2011). A scalable parallel sorting algorithm using exact splitting. Technical report, Aachen.