Quick Sample Sort

Caroline Freitas Alvernaz

1 Introdução

Foi implementado o algoritmo QuickSampleSort utilizando 4 processadores para simular a execução paralela proposta por Joseph Jaja (2000) em "A Perspective on Quicksort". O método consiste em dividir o vetor de entrada em quatro partes, selecionar pivôs globais com base em amostragem aleatória, redistribuir os elementos conforme esses pivôs e ordenar localmente cada subvetor com o QuickSort convencional. O trabalho também compara o melhor caso, em que o vetor é dividido em quatro subvetores de mesmo tamanho, ao caso médio, obtido pela estratégia probabilística de amostragem descrita no artigo.

2 Implementação

O algoritmo foi desenvolvido na linguagem Java, simulando a execução paralela do QuickSampleSort em 4 processadores. Foram definidas as constantes NUM_THREADS = 4, SAMPLE_FACTOR = 4 e MAX_ELEMENTS = 100, que determinam respectivamente o número de threads, o fator de amostragem utilizado e o tamanho máximo do vetor de entrada.

O vetor de inteiros é gerado de forma aleatória e sem repetição pelo método generateDistinctRandomArray, garantindo variedade e unicidade dos elementos. No melhor caso, o vetor é inicialmente ordenado e dividido em quatro subvetores de mesmo tamanho, sendo escolhidos pivôs determinísticos nas posições correspondentes aos quartis do conjunto. Já no caso médio, cada thread realiza uma ordenação local de seu segmento, extrai amostras uniformemente distribuídas, e essas amostras são reunidas, ordenadas globalmente e utilizadas para definir os pivôs por meio de amostragem aleatória, conforme o modelo descrito por Jaja (2000).

Após a definição dos pivôs, o vetor é redistribuído em quatro buckets (subvetores), de acordo com os intervalos delimitados pelos pivôs. Em seguida, cada subvetor pode ser ordenado localmente com o QuickSort convencional. Ao final, os subvetores são combinados para formar o vetor final ordenado, permitindo comparar visualmente a divisão dos dados no melhor e no caso médio.

3 Pseudocódigo do Quick Sample Sort

```
Algoritmo QuickSampleSort
Entrada: Vetor A[1..n]
Saída: Vetor A ordenado

Início
    Definir NUM_THREADS = 4, SAMPLE_FACTOR = 4, MAX_ELEMENTS = 100
    Gerar vetor A com elementos distintos e aleatórios

    // Melhor caso
    Ordenar A
    Selecionar pivôs determinísticos nos quartis do vetor
    Embaralhar A
    Distribuir elementos em 4 subvetores conforme os pivôs
    Exibir tamanho e conteúdo de cada subvetor

    // Caso médio
    Dividir A em 4 partes
    Cada thread ordena sua parte e seleciona amostras locais
    Reunir e ordenar todas as amostras
    Escolher 3 pivôs globais igualmente espaçados
    Redistribuir elementos conforme os pivôs
    Exibir tamanho e conteúdo de cada subvetor

    Ordenar A completamente para exibição final
Fim
```

4 Conclusão

No melhor caso, o algoritmo QuickSampleSort apresentou partições perfeitamente balanceadas, com os 100 elementos divididos em quatro subvetores de 25 elementos cada. Essa divisão uniforme ocorre porque os pivôs foram escolhidos exatamente nas posições correspondentes aos quartis do vetor, garantindo que cada processador receba a mesma quantidade de dados para ordenar. Com o trabalho igualmente distribuído entre as threads, o algoritmo reduz o número total de comparações e trocas, alcançando um desempenho ideal. Nessa condição, a complexidade de tempo é proporcional a $(n/p) \cdot \log(n/p)$, que para quatro processadores resulta em um comportamento equivalente a $O(n \log n)$, mas com melhor aproveitamento do paralelismo. Esse cenário confirma a eficiência descrita por Joseph Jaja (2000), onde o balanceamento perfeito das partições representa o limite ótimo de desempenho do QuickSampleSort.

5 Referência

Jaja, J. (2000). A Perspective on Quicksort. *Computing in Science and Engineering*, 2(1), 43–49.