UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

RELATÓRIO DE QUALIDADE DE *SOFTWARE*

Sadi Júnior Domingos Jacinto

Professor orientador: Odorico Machado Mendizabal

Florianópolis
2019

Sadi Júnior Domingos Jacinto

RELATÓRIO DE ANÁLISE DE DESEMPENHO USANDO PARALELISMO

Análise de desempenho de algoritmos de ordenação usando paralelismo requerido pelo professor da disciplina Programação Paralela e Distribuída, Odorico Machado Mendizabal, necessário para obtenção de nota.

Professor orientador: Odorico Machado Mendizabal

Florianópolis

2019

Conteúdo

1 PROBLEMÁTICA

O presente trabalho refere-se à implementação e análise de dois algoritmos de ordenação usando processamento paralelos e usando as linguagens de programação *C* e *Java*. Tal estudo objetiva descobrir as diferenças de desempenho entre cada um dos algoritmos de ordenação, além de também analisar a diferença de desempenho entre essas duas linguagens.

2 ALGORITMOS

Os algoritmos de ordenação usados neste trabalho foram:

2.1 BUBBLE SORT

O Bubble Sort é um algoritmo simples que é usado para ordenar um dado conjunto de n elementos fornecidos na forma de um *array* com n número de elementos. O Bubble Sort compara todos os elementos, um por um, e classifica-os com base em seus valores.

Se a matriz determinada tiver que ser classificada em ordem crescente, então a ordenação em bolha começará comparando o primeiro elemento da matriz com o segundo, se o primeiro elemento for maior que o segundo, ele trocará os elementos e, em seguida, siga em frente para comparar o segundo e o terceiro elemento, e assim por diante.

Se tivermos n elementos totais, precisamos repetir esse processo por *n-1* vezes.

É conhecido como Bubble sort, porque a cada iteração completa é o maior elemento na matriz dada, borbulha em direção ao último lugar ou ao índice mais alto, assim como uma bolha d'água sobe até a superfície da água.

A classificação ocorre percorrendo todos os elementos um por um e comparando-o com o elemento adjacente e trocando-os, se necessário.

No melhor caso, o algoritmo executa n operações relevantes, onde n representa o número de elementos do vetor. No pior caso, são feitas n^2 operações. A complexidade desse algoritmo é de ordem quadrática. Por isso, ele não é recomendado para programas que precisem de velocidade e operem com quantidade elevada de dados.

2.2 MERGE SORT

A ideia básica do Merge Sort é criar uma sequência ordenada a partir de duas outras também ordenadas. Para isso, o algoritmo Merge Sort divide a sequência original em pares de dados, agrupa estes pares na ordem desejada; depois as agrupa as sequências de pares já ordenados, formando uma nova sequência ordenada de quatro elementos, e assim por diante, até ter toda a sequência ordenada.
O algoritmo segue três passos comuns aos algoritmos dividir-para-conquistar:

1. Dividir:
   Dividir os dados em subsequências pequenas. Este passo é realizado recursivamente, iniciando com a divisão do vetor de n elementos em duas metades, cada uma das metades é novamente dividida em duas novas metades e assim por diante, até que não seja mais possível a divisão (ou seja, sobrem n vetores com um elemento cada).

2. Conquistar:
   Classificar as duas metades recursivamente aplicando o algoritmo do Merge Sort;

3. Combinar:
   Juntar as duas metades em um único conjunto já classificado. Para completar a ordenação do vetor original de n elementos, faz-se o Merge ou a fusão dos sub-vetores já ordenados.

A desvantagem do Merge Sort é que requer o dobro de memória, ou seja, precisa de um vetor com as mesmas dimensões do vetor que está sendo classificado.

A vantagem (ou desvantagem, dependendo do observador), é que a complexidade do Merge Sort (O(n log n)) é a mesma para o melhor, médio e pior caso, uma vez que, independente da situação dos dados no vetor, o algoritmo irá sempre dividir e intercalar os dados.

3 AMBIENTE DE TESTE

Os testes de desempenho foram executados em um computador com as seguintes configurações:

Além disso, foram adicionados *logs* dos testes que se encontram no arquivo anexado junto com este relatório.

4 METODOLOGIA

Para tornar o processo paralelo possível, ambas as aplicações (em C e em Java), utilizam o *input* do usuário para definir o número de linhas e colunas

5 CONCLUSÕES