Mestre-Escravo Usando MPI

Trabalho 1 – Programação Paralela e Distribuída

Maiki Buffet
Estudante de Engenharia de Computação
Pontifícia Universidade Católica do Rio Grande do Sul
Porto Alegre, Brasil
maiki.buffet@acad.pucrs.br

***Resumo* — Este estudo tem como finalidade apresentar testes com algoritmos paralelos utilizando MPI e o paradigma Mestre-Escravo, fazendo uso de métodos como Ranksort e Mergesort para ordenação de elementos, a fim de equiparar o *Speed Up* e a eficiência em relação ao algoritmo sequencial.**

Palavras-chave* — *Programação Paralela; MPI; Mestre-Escravo; Mergesort; Ranksort; Speed Up; Eficiência.

I. INTRODUÇÃO

Não é fácil a criação de algoritmos paralelos por um número de razões. Uma delas é que a maioria das pessoas tendem a pensar de maneira sequencial; outra razão é o fato do tratamento de áreas críticas e troca de mensagens entre processos paralelos não ser trivial. O Message Passing Interface (MPI) [1] é um sistema de transmissão de mensagens padronizado que ajuda o programador a realizar estas tarefas de forma facilitada, uma vez que cria e administra a interface na criação de algoritmos paralelos.

II. FUNCIONAMENTO

A. *Algoritmo Sequencial*

O algoritmo sequencial cria um vetor e o preenche com números inteiros positivos e não repetidos, através da leitura de um arquivo. Após isto começa a contagem do tempo do processamento do método Ranksort [3] – responsável pela ordenação crescente dos valores; até o ponto em que o método é finalizado e a contagem encerrada, computando o tempo gasto no processamento; além da entrega dos valores já ordenados.

B. *Algoritmo Mestre-Escravo*

O algoritmo Mestre-Escravo executa os mesmos passos que o algoritmo sequencial até, inclusive, o momento do início da contagem do processamento do método Ranksort. A diferença entre os algoritmos consiste em que ao invés de mandar todo o vetor desordenado para ser ordenado por um único processo, o mestre divide o vetor em tarefas menores para serem enviadas aos escravos. O tamanho de cada tarefa foi definido como:

$$T = N / (M * 4)$$

Onde N é o número de elementos a serem ordenados e M é o número de processos destinados a execução do algoritmo. Os escravos então ordenam através do Ranksort a parte que receberam do mestre e enviam de volta ao mesmo o vetor já ordenado. O mestre, por sua vez, ao receber as tarefas dos escravos executa o método Mergesort [3] – que consiste em pegar os vetores recebidos e combiná-los. Ao fim da execução o mestre para a contagem do processamento e envia um sinal para cada escravo a fim de finalizá-lo. Novamente, repetindo o algoritmo sequencial, entrega os valores ordenados e o tempo gasto no processamento total desta tarefa de ordenação.

III. EXECUÇÃO

A. *Execução*

São necessários o envio de três parâmetros para a execução de cada algoritmo – ambos integrados no mesmo arquivo fonte:

- O número de processos a serem executados;
- O número de elementos a serem ordenados;
- O nome do arquivo com os elementos a serem lidos.

IV. SAÍDA

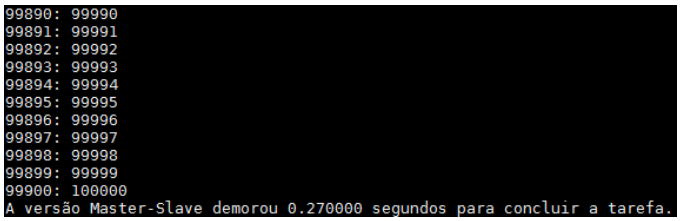


Fig. 1. Saída padrão com elementos ordenados e tempo gasto neste processamento. Na figura foram utilizados 9 processos (1 mestre e 8 escravos) na ordenação de 99900 valores.

V. RESULTADOS

Os resultados foram computados por uma única máquina do *cluster* Cerrado – composto por 2 *enclosures* Dell PowerEdge M1000e com 16 *blades* Dell PowerEdge M610 e 15 *blades* Dell PowerEdge M620, sendo que cada máquina possui dois processadores Intel Xeon Six-Core E5645 2.4GHz Hyper-Threading e 24GB de memória, totalizando 12 núcleos (24 threads) por nó e 372 núcleos (744 threads) no cluster. Estas máquinas que estão presentes no Laboratório de Alto Desempenho (LAD) [2], executaram cada teste três vezes e uma média entre eles fora feita.

Fig. 2. Imagem frontal do cluster Cerrado.

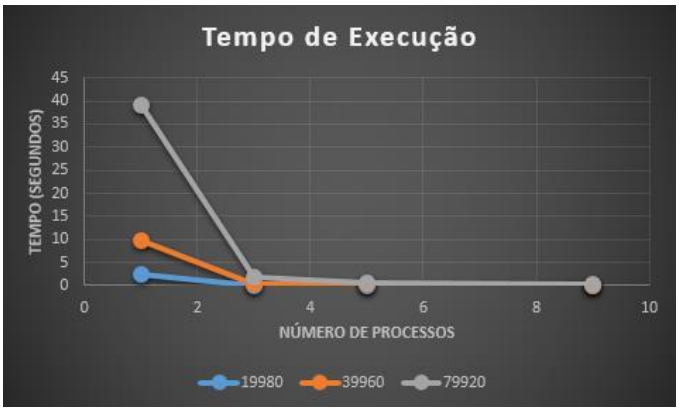
A. *Tempo de Execução*

Os tempos de execução foram retratados na tabela 1.

Casos de Teste	Sequencial	2 Escravos	4 Escravos	8 Escravos
19980	2.51	0.1	0.03	0.01
39960	9.74	0.43	0.13	0.05
79920	39.01	1.67	0.53	0.17

Tab. 1. Tempo de Execução em segundos.

O tempo de execução está ilustrado no gráfico 1.



Gráf. 1. Gráfico do tempo de execução em função da quantidade de processos.

B. *Speed Up*

O *Speed Up* para um determinado número de processos é calculado pela fórmula:

$$Spn = Ts / Tn$$

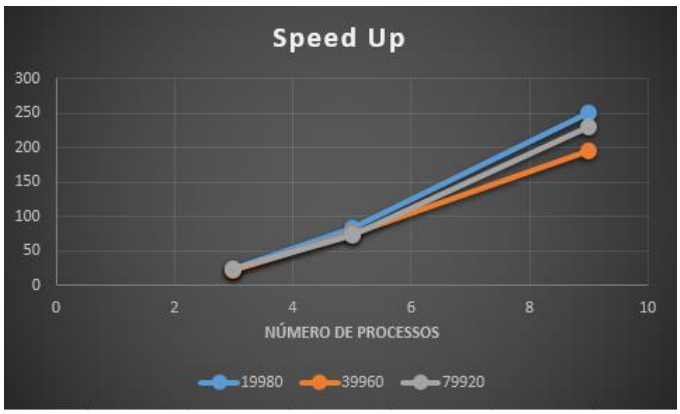
Onde *Ts* é o tempo sequencial e *Tn* é o tempo com *n* processos.

Os *Speed Up's* foram retratados na tabela 2.

Casos de Teste	2 Escravos	4 Escravos	8 Escravos
19980	25.1	83.67	251
39960	22.65	74.92	194.8
79920	23.36	73.6	229.47

Tab. 2. Speed Up

O *Speed Up* está ilustrado no gráfico 2.



Gráf. 2. Gráfico do *Speed Up* em função da quantidade de processos.

C. *Eficiência*

A eficiência para um determinado número de processos é calculada pela fórmula:

$$Efn = Spn / n$$

Onde *Spn* é o *Speed Up* com um determinado número de processos e *n* é o número de processos.

A eficiência foi retratada na tabela 3.

Casos de Teste	2 Escravos	4 Escravos	8 Escravos
19980	8.37	16.74	27.89
39960	7.55	14.98	21.64
79920	7.79	14.72	25.5

Tab. 3. Eficiência.

A eficiência está ilustrada no gráfico 3.

Gráf. 3. Gráfico da eficiência em função da quantidade de processos.

VI. CONCLUSÃO

É fácil determinar que este é um algoritmo *Superescalar* ao fato que o *Speed Up* e a eficiência continuam a aumentar, mesmo que sejam adicionados maiores valores de comunicação entre o mestre e os escravos. Isto pode ser explicado examinando a complexidade do algoritmo de ordenamento:

$$O(n^2)$$

Isto significa que dividindo o vetor de elementos por quatro, você está atualmente dividindo a complexidade do resultado por dezesseis.

$$(n / 4)^2 = n^2 / 16$$

Além disto, deve-se destacar os seguintes pontos:

- O tempo de execução foi bastante reduzido ao se aumentar o número de processos. Entretanto, aumentar demasiadamente esta quantidade pode fazer com que a aplicação gaste mais tempo realizando a comunicação entre os processos do que propriamente realizando a ordenação dos elementos;
- Devido aos resultados encontrados, verifica-se que a eficiência vai muito além de 1 (valor ideal). Deve-se entender que isto ocorre devido ao fato de que o algoritmo sequencial possuía – em termos de performance – um desempenho ruim.

REFERÊNCIAS

[1] *Message Passing Interface (MPI)*, Blaise Barney, Lawrence Livermore National Laboratory, Livermore, CA, EUA. Disponível em: computing.llnl.gov/tutorials/mpi/

[2] *Laboratória de Alto Desempenho*, Faculdade de Informática, PUCRS, Porto Alegre, RS, Brasil. Disponível em: www3.pucrs.br/portal/page/portal/ideia/Capa/LAD/LADInfraestrutura/LADInfraestruturaHardware

[3] *Parallel Programming: Techniques and Applications using Networked Workstations and Parallel Computers - Chapter 9 Sorting Algorithms*, Barry Wilksinson and Michael Allen, Prentice Hall, Upper Saddle River, NJ, EUA. Disponivel em: grid.cs.gsu.edu/~cscyip/csc4310/Slides9.pdf