Programação Paralela Trabalho 1

Bruno Brandelli, Rodrigo Pacheco, Usuário pp12804

I. Introdução

Neste primeiro trabalho da disciplina de programação paralela, temos como objetivo desenvolver uma solução para ordenar uma lista de vetores utilizando o algoritmo de ordenação Quick Sort de forma paralela, tendo como modelo a arquitetura mestre/escravo. Na realização deste trabalho serão utilizados mil vetores com cem mil elementos cada, que estarão organizados de forma aleatória, e em um primeiro momento será utilizada uma solução sequencial para ter como base nos cálculos de eficiência e speed-up da solução que utilizará processos paralelos.

II. Modelo Paralelo

Inicialmente a tarefa que foi proposta, necessitava de uma solução sequencial para ser usada como base de comparação. Tal solução consistia de um simples laço iterando pelo vetor de vetores, enquanto os ordenava.

Partindo para a solução paralela era necessário seguir certas especificações, como iniciativa nos escravos, armazenamento do vetor ordenado na mesma posição onde ele encontrava-se originalmente. Neste modelo de arquitetura, o mestre tem a função de organizar o trabalho que tem a ser feito, e enviar o trabalho aos escravos, estes por sua vez realizam o ordenamento do vetor, utilizando o algoritmo Quick Sort.

Apesar da especificação requisitar que a iniciativa fosse dos escravos, foi feita uma modificação em que a primeira carga de trabalho é enviada pelo mestre, sem a solicitação por parte dos escravos. Tal modificação visa aumentar a eficiência desta solução, já que desta forma não haverá uma espera pela iniciativa dos escravos.

III. Análise de Resultados

Após implementar a solução utilizando processos paralelos foram realizados testes com diferentes números de processos, de modo que fosse possível reunir dados para uma melhor análise da eficiência e speed-up. Tais resultados podem ser visualizados na tabela 1 que contém os a relação entre os fatores observados nos testes, e na figura 1, que apresenta em forma de gráfico a comparação entre os resultados obtidos.

Processos	Tempo de Execução(s)	Speed-Up	Speed-Up Ideal	Eficiência
1	4,405	1,00	1	1,00
2	5,418	0,8	2	0,4
3	2,699	1,6	3	0,5
4	1,804	2,4	4	0,5
8	0,809	5,4	8	0,7
16	0,387	7	16	0,9
17	0,378	4,8	17	0,3
30	0,351	2,3	30	0,1
32	0,336	1,9	32	0,1

Table I
Resultados obtidos

Dentre os resultados observados, podemos notar algumas peculiaridades na execução deste algoritmo. Primeiramente é possível notar que ao rodar com dois processos o tempo de execução foi maior do que ao rodar a solução sequencial, tal resultado é decorrência de uma divisão desigual de carga de trabalho entre os papéis de mestre e escravo, onde o mestre mandava trabalho para apenas um escravo, tornando o valor da comunição entre eles mais caro do que o trabalho a ser realizado. Mas após este número de processos ser incrementado os resultado passaram a favorecer o paralelismo, chegando ao seu pico de eficiência com 16 processos. Após isso a troca de mensagem entre o mestre e os escravos torna-se mais lento, pois muitos escravos passavam a ficar mais tempo ociosos do que trabalhando.

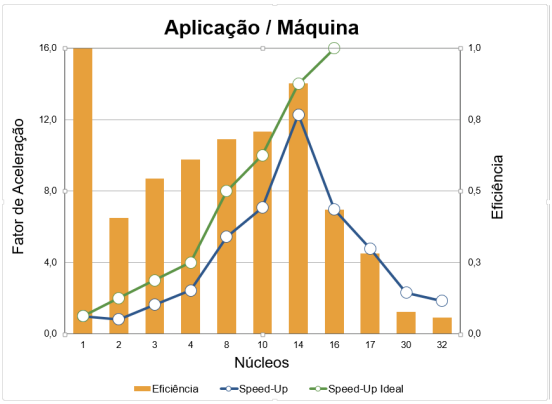

Figure 1. Gráfico de eficiência/speed-up

IV. Dificuldades Encontradas

O grupo não encontrou maiores dificuldades neste primeiro trabalho, devido a familiaridade com a linguagem de programação C, e a quantidade de material disponivel sobre MPI que facilitou seu aprendizado.