Programação Paralela
Trabalho III

Giovanni Cupertino, Matthias Nunes, *Usuário pp12820*

I. Introdução

O objetivo do trabalho é desenvolver uma solução que ordene diversos vetores utilizando o algoritmo Quick Sort. Os vetores contém cem mil elementos e são, no total, dez mil vetores que estão na ordem inversa de valores indo de noventa e nove mil novecentos e noventa e nove na primeira posição até zero na ultima posição. Para a abordagem paralela do trabalho, utilizou-se uma implementação híbrida utilizando as ferramentas MPI e OpenMP. Para esta abordagem temos um dos processos como mestre, que é responsável por mandar mensagens com os vetores a serem ordenados pelos nós do cluster e reorganizar os vetores nas suas posições originais na estrutura. Os outros processos (escravos) recebem os vetores, os ordenam e devolvem ao mestre utilizando o OpenMP para a exploração do paralelismo com memória compartilhada dentro do nó.

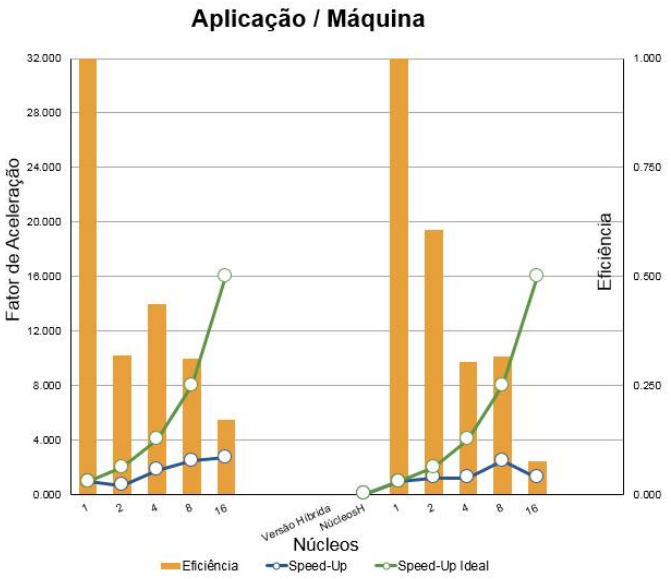

Figura 1: Gráfico gerado a partir da tabela

Núcleos	Tempo de Execução (s)	Speed-Up	Speed-Up Ideal	Eficiência
1	6.59	1.000	1	1.000
2	10.316653	0.639	2	0.319
4	3.775602	1.745	4	0.436
8	2.633243	2.503	8	0.313
16	2.380381	2.768	16	0.173
Versão Híbrida				
NúcleosH	Tempo de ExecuçãoH (s)	Speed-UpH	Speed-Up IdealH	EficiênciaH
1	37.968962	1	1	1
2	31.269688	1.21424179	2	0.6071209
4	31.071716	1.22197828	4	0.30549457
8	30.996441	2.53126413	8	0.31640802
16	31.068571	1.22210198	16	0.07638137

II. Análise dos Resultados Obtidos

É possível observar que o tempo de resposta para dois processos é maior que quando executado sequencialmente, isso ocorre pelo fato de que só temos um processo realizando a ordenação e o outro como mestre, havendo um tempo extra principalmente pela troca de mensagens entre os dois, como pode ser visto na tabela **??**.

Essas pequenas diferenças, como a troca de mensagens, a reorganização dos vetores nas suas posições inicias e a comunicação entre os 2 nós utilizados também reduzem a eficiência e criam diferenças entre o speed-up ideal e o real. Devido a velocidade do algoritmo de ordenação o tempo para realizar a tarefa depois de cinco processos permaneceu bastante semelhante, já que os escravos terminam a sua tarefa antes mesmo do mestre conseguir enviar uma nova para todos os processos que estão solicitando. Algumas alternativas para maior utilização dos núcleos, para os casos com mais de cinco, seriam possuir mais de um mestre permitindo maior

distribuição de vetores aos escravos(cada um controlando escravos diferentes e passando os vetores quando lhes forem requeridos) e a utilização de um algoritmo de ordenação mais lento que proporcionaria mais tempo ao mestre para distribuir vetores aos escravos, ou seja, antes que tivesse novas requisições.

O fato de se estar utilizando dois nós e a biblioteca MPI permite um balanceamento da carga igualitário dos processos nos núcleos físicos (8 em cada uma das máquinas com capacidade de simular 16 threads). Mesmo utilizando capacidades acima de dezesseis processos (*hyper-threading*), não foi possível observar melhoras significativas nos tempos de execução e por consequência no speed-up e na eficiência.

III. Dificuldades Encontradas

Não foram encontradas dificuldades na implementação desse trabalho, nem na utilização da biblioteca MPI.