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 este trabalho 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 utilizando o algoritmo do quick sort e devolvem ao mestre utilizando o OpenMP para a exploração do paralelismo com memória compartilhada dentro do nó.

II. Análise dos Resultados Obtidos

É possível observar que o tempo de resposta não tem grande melhoria com o aumento do número de threads e que o tempo de execução do algoritmo mesmo utilizando mais nós do que quando realizado no primeiro trabalho não apresentou benefícios com as configurações disponíveis. Os tempos de execução utilizando a implementação hibrida não obteve muito sucesso quanto ao tempo de resposta e seus tempos com duas ou mais threads se manteve muito proxima degradando a eficiência. 1.

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

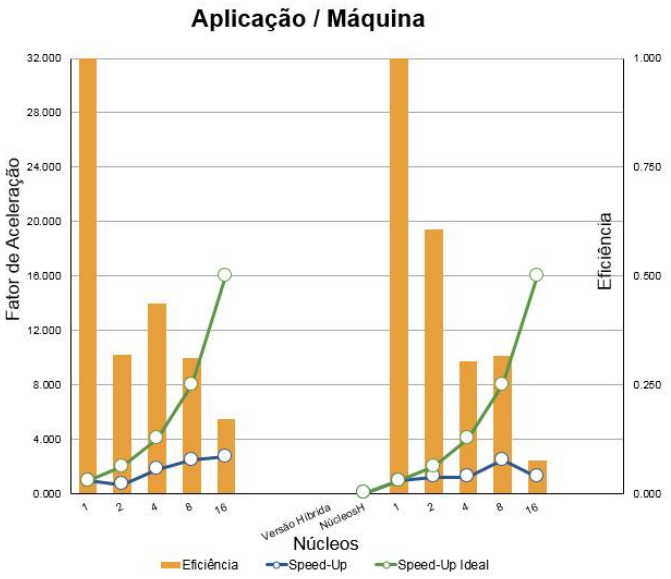

Figura 1: Gráfico gerado a partir da tabela

A diferença entre a troca de mensagens pequenas para mensagens maiores podem ser um dos motivos que impactou no tempo de resposta, já que os processos passam a receber mensagens maiores e tem que processalas em paralelo com as threads disponíveis e tira o trabalho do mestre em dividir em trabalhos menores para serem realizados. Outra questão é que número de núcleos que os nós possuem, para esse tipo de implementação, pode ser ainda muito baixo para que se obtenha resultados mais satisfatórios.

O fato de se estar utilizando quatro nós e a biblioteca MPI junto a OpenMP permite uma menor dependência no mestre e que os pacotes de trabalho a serem ordenados sejam realizados pelos nós((8 núcleos em cada uma das máquinas com capacidade de simular 16 threads)) que conseguem explorar um paralilismo com memória compartilhadarealizando mais de um pedaço da tarefa ao mesmo tempo para cada processo. Quanto a distribuição de carga que fica a cargo do MPI o algoritmo consegue distribuir ela igualmente parar os processos que estão ativos nos diferentes nós.

III. Dificuldades Encontradas

Foram encontradas dificuldades quanto ao modo de trabalhar do OpenMP que diminuiram após algumas pesquisas de como ele funciona, outras dificuldades encontradas foram relativas ao incremento do tempo de execução quando comparado aos do trabalho um e quais são as razoões para esse tempo ter aumentado.