Arthur Zanella Lovato, Caroline Lewandowski, Felipe Roque

EXECUÇÃO E ANÁLISE DO MODELO MESTRE-ESCRAVO

1. Introdução

O presente trabalho analisa o funcionamento de uma implementação paralela do modelo mestre escravo utilizando a ferramenta MPI. Para descrever os resultados das execuções solicitadas, com o fim de avaliar a performance obtida da utilização do algoritmo, duas versões foram fornecidas para realizar comparações. Sendo que, uma delas foi desenvolvida de forma sequencial, enquanto a outra utiliza o padrão de comunicação *MPI* (Message Passing Interface) para permitir o paralelismo de sua execução. Ambas foram desenvolvidas na linguagem C e possuem os algoritmos *Quick Sort* e *Bubble Sort* para a ordenação de vetores.

Os pontos a serem descritos abrangem a análise de medições de desempenho de 1000 vetores presentes na *Bag-of-Tasks* (BoT), onde cada uma possuirá 100 mil elementos para o algoritmo *quicksort* e 10 mil para o *bubblesort*, considerando a medição dos tempos de execução, gráfico de *speed up* e cálculo de eficiência ao realizar testes com diferentes processadores, utilização de *Hyper-Threading* para os algoritmos e balanceamento da carga ao serem executados com 16 processadores físicos e 32 processadores *Hyper-Threading*.

1.1 Descrição do Algoritmo

O modelo Mestre-Escravo compreende a classificação de um processo designado para realização da articulação e organização de demais processos. Este processo organizador, denominado mestre, normalmente não realiza processamentos, mas os delega para outros processos classificados como escravos. O processo mestre possui um saco de trabalho, que, na versão fornecida, foi implementado na forma de vetor *m* por *n*. Sendo "*m*" o número de vetores e "*n*" o tamanho do vetor.

Em questão de comunicação, não há troca de mensagens entre escravos, sendo reduzido apenas ao mestre delegando tarefas aos escravos e escravos devolvendo estas tarefas após seu processamento. Tendo a possibilidade de variação entre as iniciativas tomadas entre estas duas entidades, sendo que a implementação analisada possui o padrão de iniciativa tomada pelo mestre: Ele quem delega e recolhe as tarefas, enquanto os escravos permanecem reativos.

1.2 Análise de resultados das execuções

Os testes foram realizados utilizando o Laboratório de Alto Desempenho (LAD), onde alocaram-se 2 nós de forma exclusiva para a compilação e execução das versões a serem analisadas, considerando que desta forma será possível imprimir resultados de tempo de processamento coerentes, sem interferências que uma alocação compartilhada poderia provocar. Cada nó possui 8 processadores, resultando no processamento de 16 threads, dessa forma, deverá totalizar 16 processos e 32 processos com a tecnologia de Hyper-Threading.

Os arquivos executados sob as configurações descritas acima foram nomeados "*ms_seq.c*" e "*ms_mpi.c*". Considera-se que o primeiro arquivo, referente ao paradigma sequencial, será usado como base para apresentar as diferenças em relação ao segundo arquivo, sendo ele respectivamente a versão paralela. É esperado que os algoritmos sequenciais tenham um tempo de execução maior do que os algoritmos paralelos dado a capacidade dos algoritmos paralelos em executar múltiplas tarefas de forma simultânea.

A Tabela 1 apresenta os resultados das execuções paralelas seguido da Tabela 2 com a média de 5 execuções para duas quantidades de vetores da versão sequencial, organizados pelo tamanho da carga de trabalho adotada. A partir disso, foi possível observar a redução no tempo de execução da versão paralela em relação a versão sequencial, porém em questão de eficiência os algoritmos divergem o custo-benefício para as versões paralelas: O ganho é baixo e pouco escalável para o *quicksort* devido a sua complexidade *O(n log n)* que apresenta uma ordenação mais rápida, ou seja, o aumento da quantidade de processadores não causa um grande impacto em sua eficiência. Diferente do *bubblesort*, que por possuir uma complexidade $O(n^2)$, consegue obter maior vantagem do paralelismo. Na sequência, é apresentado na Figura 1 o gráfico de *speed-up* relacionando a quantidade de processos, eficiência e speed-up. Pontua-se que após um certo número de processos houve uma queda de desempenho.

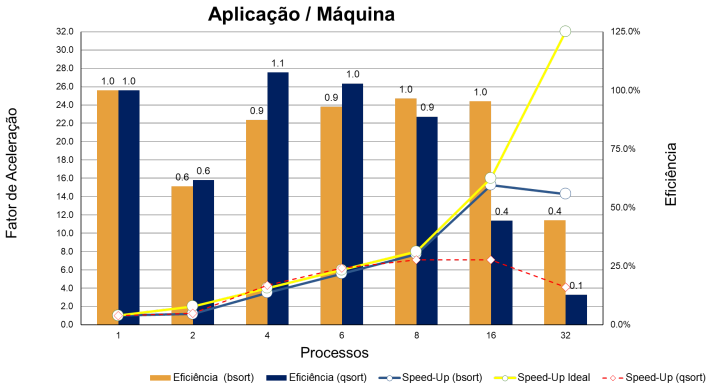
Tabela 1 - Valores de *speed-up* para diferentes n° de processos.

	Bubble Sort			**Quick Sort**		
Processos	***Tempo (s)***	***Speed-Up***	***Eficiência***	***Tempo (s)***	***Speed-Up***	***Eficiência***
1	53,56	1,0	1,0	1,42	1,0	1,0
2	45,36	1,2	0,6	1,15	1,2	0,6
4	15,33	3,5	0,9	0,33	4,3	1,1
6	9,6	5,6	0,9	0,23	6,2	1,0
8	6,94	7,7	1,0	0,20	7,1	0,9
16	3,51	15,3	1,0	0,20	7,1	0,4
32	3,75	14,3	0,4	0,28	4,1	0,1

Tabela 2 - Média dos tempos das execuções sequenciais.

	Bubble Sort		**Quick Sort**	
Qtd. Vetores	***10***	***100***	***10***	***100***
Média (seg)	5,35836	53,56274	0,14378	1,42556

Figura 1 - Gráfico de *speed-up* para diferentes processos.



Quanto à utilização HT, observa-se que há uma redução na eficiência, onde o *speed-up* calculado é inferior aos resultados obtidos com 16 núcleos. O ganho em tempo não é significante para garantir uma boa eficiência pois, em certo ponto, torna-se prejudicial concentrar mais núcleos do que o necessário em uma execução, resulta em ociosidade por acumular muitos processos onde há pouca requisição de processamento. Por fim, o balanceamento também apresentou resultados diferentes para os dois algoritmos executados. Para a estrutura do modelo mestre-escravo com quatro processos, um nó conterá um processo mestre e outro processo escravo, enquanto outro nó conterá dois processos escravos para as tarefas delegadas. O nó que permanece com o processo mestre poderá executar mais tarefas por não precisar compartilhar a CPU, como no caso do segundo nó.

Portanto, esse cenário resultará num alto desbalanceamento de carga para o algoritmo *quicksort*. Devido ao baixo tempo de processamento dada sua complexidade, haverá um impacto na delegação de tarefas e um aumento na competição entre os processos para receber novas tarefas, concedendo então uma vantagem para o processo que compartilha o mesmo nó que o processo mestre, devido ao menor tempo de comunicação, resultando assim num desequilíbrio no balanceamento de carga. Já o algoritmo *bubblesort* não resulta em um alto desbalanceamento neste cenário, devido ao tempo de processamento ser mais lento devido sua complexidade, ou seja, por gerar menos competição por tarefas, a delegação das mesmas não é favorecida pelo tempo de comunicação.

Então, tendo em vista os resultados apresentados, é possível concluir que se faz necessário avaliar as características da tarefa antes da implementação de uma solução paralela ou da distribuição de processos, pois em certos casos é possível obter uma maior eficiência com o paradigma sequencial ou com menos processos do que os disponíveis pelo sistema.