Relatório de Entrega de Trabalho
Disciplina de programação Paralela (PP) – Prof. César De Rose
Alunos: Alexandre Yukio Ichida, Lucas Ranzi **Usuário:** pp12801
Exercício: trabalho 1 de MPI (ME) **Entrega:** 29/04/2017

1) Implementação

Durante o desenvolvimento, foram realizados duas implementações. O programa mais simples (Programa 1) o mestre realiza distribuição de tarefas começa o processo de envio apenas quando receber as respostas de todos os escravos. Essa implementação além de ter maior duração de execução, é menos otimizada em relação ao uso do paralelismo pois caso algum processo escravo termine a tarefa previamente, terá que esperar o próximo ciclo de coleta de requisições feitas pelo mestre.
Já a segunda implementação (Programa 2) foi feita tendo o processo mestre realizando a chamada de recebimento para uma fonte qualquer (MPI_ANY_SOURCE) e enviando o próximo vetor disponível para o processo atendido. Logo caso um processo termine mais cedo ele será atendido imediatamente, ou seja, o mestre receberá o primeiro vetor disponível por algum processo escravo. Porém em relação ao balanceamento de carga, como o mestre atende a primeira requisição e envia a nova tarefa para o primeiro escravo atendido, a distribuição de carga entre os escravos não é tão justa comparada com o programa 1, podendo ter casos onde um determinado escravo demore uma determinada tarefa, recebendo menos tarefas comparado com os outros escravos. Foi selecionado o Programa 2 para testes e análise de desempenho devido ao fato de ser mais otimizado em relação ao uso do paralelismo.

2) Dificuldades encontradas

Depuração das instruções executadas pelo MPI, testes e análise das saídas para montagem de justificativa dos resultados.

3) Testes

Os testes foram executados utilizando o cluster Gates do laboratório de alto desempenho, utilizando a máquina Grad. Foi comparado a implementação sequencial(1 processo) com as implementações paralelas utilizando 2, 4, 8, 16 e 32 processos. Como a quantidade máxima de processos por cada nodo é de 16, a execução de 32 processos foi realizada apenas utilizando 2 nodos.

4) Análise do Desempenho

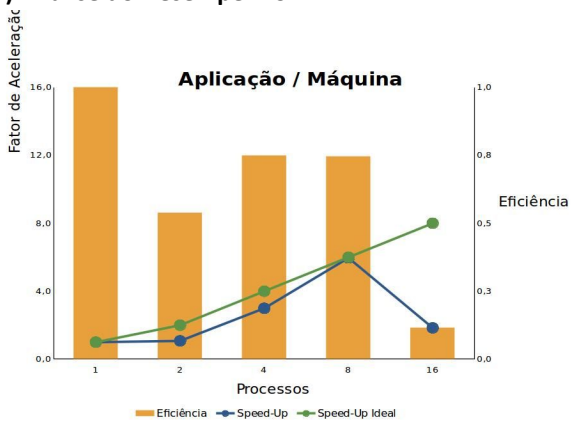

Figura 1 - Resultados utilizando diferentes números de processo rodando em 1 nodo do cluster.

Conforme mostra a Figura 1, o speed up torna-se próximo do ideal utilizando até até 8 processos (1 mestre e 7 escravos), após isso começa a se distanciar do speed up realizado do ideal a medida que é acrescenta processos para a execução paralela.

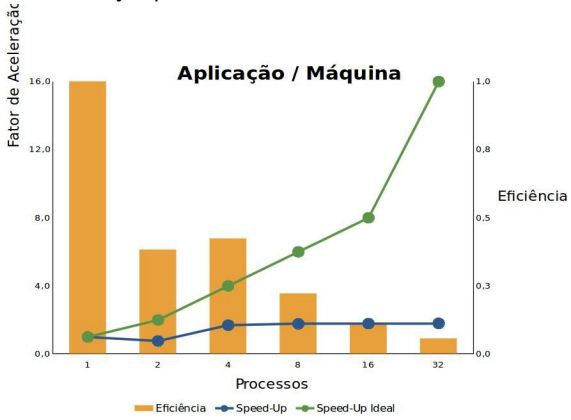

Figura 2 - Resultados utilizando diferentes números de processo rodando em 2 nodo do cluster.

Comparando a figura 2 com a figura 1, nota-se com o aumento de número de nodos a execução se torna menos otimizada em relação ao uso do paralelismo.
Devido ao fato do custo de comunicação entre os nodos, a duração das execuções aumentaram, por mais que a utilização de 2 nodos possibilite a execução utilizando maior número de processos paralelos. A baixa eficiência é devido a alguns processos escravos serem executados fora do nodo onde o mestre está rodando, tendo custo extra de comunicação entre os nodos.

5) Observações Finais

Utilizando os parâmetros contidos no enunciado, nota-se que a execução ao longo dos processos teve perda de eficiência com poucos processos. Também referente aos parâmetros do enunciado (1000 vetores de 100000 posições), o programa mostrou uma eficiência muito abaixo com a utilização de um nodo extra comparado a execução.

Fontes no Github:
Programa 1:
https://github.com/yukioichida/parallel-programming/blob/1.0-RC/mpi/sort_vector_mpi.c
Programa 2:
https://github.com/yukioichida/parallel-programming/blob/master/mpi/sort_vector_mpi.c