Relatório de Entrega de Trabalho
Disciplina de programação Paralela (PP) – Prof. César De Rose
Aluno: Fábio Delazeri Riffel; Lásaro Curtinaz Dumer **Usuário:** pp12806; pp12818
Exercício: Trabalho 1 de MPI (Quick Sort) **Entrega:** 29/09/2015

1) Implementação

Conforme o enunciado, utilizamos o modelo mestre escravo para executar o algoritmo Quicksort. Como discutido em aula, a primeira tarefa dos escravos é atribuída pelo mestre, de forma a agilizar a distribuição de trabalho. Para controle dos processos, foram utilizadas diferentes tags que servem para várias ações, como: pedir trabalho, devolver trabalho, suicidio dos processos, etc.
Para geração da carga de trabalho, utilizamos uma matriz que é alimentada com valores decrescentes, com base na quantidade e tamanho dos vetores definidos, tendo assim valores únicos por toda a matriz, e não vários vetores iguais.
Como era necessário que os vetores, ao serem ordenados, voltassem para a mesma posição que estavam antes da ordenação, adicionamos uma mensagem antes do envio do vetor que diz o índice ao qual ele pertence. No envio do mestre para o escravo o índice é enviado e depois o vetor. O escravo guarda o índice e realiza a tarefa. Uma vez terminada a tarefa, ao enviar o resultado para o mestre, primeiro é enviado o índice, assim o mestre fica esperando o envio específico deste escravo com o vetor na mensagem subsequente. Com posse do índice e do vetor o mestre guarda o vetor ordenado na posição correta.
Após receber um vetor, o mestre verifica se ainda existe alguma tarefa a ser realizada e caso isso seja verdade ele a envia para o escravo, se não existem mais tarefas é enviada uma mensagem de suicídio.

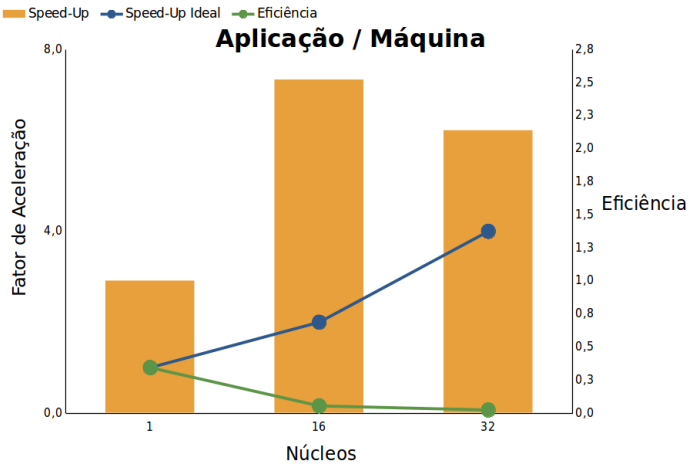
2) Dificuldades encontradas

Fazer o envio dos índices e a devida atribuição no retorno para o mestre foram tarefas um pouco mais demoradas. Como a tag da mensagem já é usada para controlar o fluxo, precisamos encontrar outro modo para enviar o índice. Cogitamos alterar a mensagem para ser uma struct contendo índice e vetor, mas para envia-la exigiria um trabalho técnico mais detalhado e complexo, exigindo uma aptidão maior em C, aumentando assim o risco de problemas. Por isso optamos pelo método mais simples de enviar o índice, que é através do envio de uma mensagem antes do envio do vetor, especificando o índice.

3) Testes

Todos os testes foram feitos utilizando o pior caso para o quicksort, que é o vetor decrescente.
Para obtermos uma maior diferença em tempo de execução para a análise, decidimos pela carga de trabalho de quatro mil vetores com cem mil valores cada.
Com as especificações definidas temos:

- Execução sequencial: 26.7 segundos
- Execução sem Hyper Threading (2 nós,16 processos): 10.59 segundos
- Execução com Hyper Threading (2 nós.32 processos): 12.48 segundos



4) Analise do desempenho

Com base no tempo sequencial e nos tempos paralelos, constatamos que a execução sem Hyper Threading (HT) é mais rápida que a sequencial pois possuímos mais processadores realizando a ordenação, porem a execução com HT é mais lenta que a sem HT pois o número de mensagens trocadas aumenta e a fila de espera para entregar tarefas e pedir novas ao mestre tente a crescer se muitos escravos terminarem suas tarefas ao mesmo tempo.

5) Observações Finais

Com a análise do desempenho concluímos que um número maior de processos nem sempre é bom para execução paralela com troca de mensagens, pois as trocas de mensagens têm um custo também, e se o somatório deste custo for alto, o ganho que se tem com o paralelismo não é compensado. Adicionado ao custo da troca de mensagens também levamos em conta a espera que pode haver no mestre por parte dos escravos, já que é o ponto de encontro dos vários processos.

```
#include <stdio.h>
#include <time.h>
#include <stdlib.h>
#include <string.h>
#include <sys/time.h>
#include "mpi.h"
#define GET_WORK 0
#define WORK_DONE 1
#define WORK 2
#define SUICIDE 3
#define WORK_INDEX 4
#define NUM_ARRAYS 4000
#define ARRAYS_SIZE 100000
#define DEBUG 0

int compare (const void * a, const void * b){return ( *(int*)a - *(int*)b );}

const double curMilis(){
   struct timeval  tv;
   gettimeofday(&tv, NULL);
   return ((tv.tv_sec) * 1000 + (tv.tv_usec) / 1000.0) +0.5; // convert tv_sec & tv_usec to millisecond
}

main(int argc, char** argv){
   int my_rank;
   int proc_n;
   int **saco;
   int toOrder[ARRAYS_SIZE];
   int ordered[ARRAYS_SIZE];
   int i,j,s;
   double t1, t2;

   srand(time(NULL));
   int r = rand();

   MPI_Status status; /* Status de retorno */
   MPI_Init(&argc , & argv); // funcao que inicializa o MPI, todo o cÃ³digo paralelo esta abaixo
   MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
   MPI_Comm_size(MPI_COMM_WORLD, &proc_n);
   int slavesAlive = proc_n-1;

   if ( my_rank == 0 ){
      t1 = MPI_Wtime();        // contagem de tempo inicia neste ponto
      saco = (int **)malloc(NUM_ARRAYS * sizeof(int *));//alocação do vetor para os vetores
      if(saco == NULL)
      {
         printf("out of memory\n");
         return -1;
      }else{
         int valor = ARRAYS_SIZE * NUM_ARRAYS;//inicialização do valor
         for(i = 0; i < NUM_ARRAYS; i++)
         {
            saco[i] =(int *) malloc(ARRAYS_SIZE * sizeof(int));//alocação de cada vetor e seu lugar
            if(saco[i] == NULL)
            {
               printf("out of memory row %d\n",i);
               return -1;
            }else{
               for(j = 0; j< ARRAYS_SIZE; j++)
               {
                  saco[i][j] = valor;//atribuição do valor ao vetor
                  valor--;
               }
```

```
            }
        }
    }

    int next = 0;
            //mestre faz primeiro envio de tarefas para os escravos
    for(s=1;s<=slavesAlive;s++){
        if(next>=NUM_ARRAYS){//se o numero de tarefas ja se esgotou, termina o escravo
            MPI_Send(&next, 1, MPI_INT,s, SUICIDE, MPI_COMM_WORLD);
            slavesAlive--;
        }else {
            MPI_Send(&next,1,MPI_INT,s,WORK_INDEX,MPI_COMM_WORLD);//envia o indice do vetor para o escravo 's'
            MPI_Send(saco[next], ARRAYS_SIZE, MPI_INT,s, WORK, MPI_COMM_WORLD);//envia o vetor para o escravo
's'
            next++;
        }
    }
    int orderedI = 0;
    while(slavesAlive > 0){//enquanto existirem escravos vivos, fica esperando mensagens
            MPI_Recv(&orderedI,1, MPI_INT,MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD,
&status);//recebe uma mensagem do escravo e abaixo verifica se é um indice de vetor
            if(status.MPI_TAG == WORK_INDEX){
        MPI_Recv(ordered, ARRAYS_SIZE, MPI_INT,status.MPI_SOURCE, WORK_DONE, MPI_COMM_WORLD,
&status);  //espera o vetor do mesmo escravo q enviou o indice
        memcpy(saco[orderedI],ordered,ARRAYS_SIZE*sizeof(int));//coloca o vetor ordenado na matriz
                    if(next>=NUM_ARRAYS){//se o numero de tarefas ja se esgotou, termina o escravo
                            MPI_Send(&next, 1, MPI_INT,status.MPI_SOURCE, SUICIDE, MPI_COMM_WORLD);
                            slavesAlive--;
        }else {//se nao envia a proxima tarefa
            MPI_Send(&next,1,MPI_INT,status.MPI_SOURCE,WORK_INDEX,MPI_COMM_WORLD);//envia o indice do
vetor para o escravo
            MPI_Send(saco[next], ARRAYS_SIZE, MPI_INT,status.MPI_SOURCE, WORK, MPI_COMM_WORLD);//envia
o vetor para o escravo
            next++;
        }
            }
    }
    printf("[%f]@master leaving...\n",curMilis());
    t2 = MPI_Wtime();       // contagem de tempo termina neste ponto
    printf("[%f]@[%d]MPI_Wtime measured work time to be: %1.2f\n",curMilis(),my_rank, t2-t1);
  }
  else
  {
    int tag = WORK;
    int toOrderI=0;
    do{
        MPI_Recv(&toOrderI, 1, MPI_INT, 0, MPI_ANY_TAG,MPI_COMM_WORLD,&status);//recebe o comando do mestr
        tag = status.MPI_TAG;
        if(tag == WORK_INDEX){//se o comando recebido foi um indice...
            MPI_Recv(toOrder, ARRAYS_SIZE, MPI_INT,0, MPI_ANY_TAG, MPI_COMM_WORLD, &status);//...fica
esperando o vetor em seguida
            tag = status.MPI_TAG;
        }
            if(tag == WORK){//recebeu um vetor para ordenar
        qsort (toOrder, ARRAYS_SIZE, sizeof(int), compare);//ordena o vetor
        MPI_Send(&toOrderI,1, MPI_INT,0, WORK_INDEX, MPI_COMM_WORLD);//envia o indice do vetor para o
mestre
        MPI_Send(toOrder,ARRAYS_SIZE, MPI_INT,0, WORK_DONE, MPI_COMM_WORLD);//envia o vetor para o
mestre
        }
    }while(tag != SUICIDE);//se a ultima tag foi a de suicidio, termina execução
  }
  MPI_Finalize();
}
```