Questão 1

Completo

Atingiu 1,0 de 1,0

Julgue as afirmativas a seguir e marque a opção correta:

I - Um socket do tipo TCP pode receber múltiplas conexões ao mesmo tempo, desde que haja processos filhos para tratarem cada uma dessas conexões

II - Nem todas as conexões TCP são full-duplex e ponto-a-ponto

III - Uma conexão de transporte na arquitetura TCP/IP é marcada por uma quíntupla que é composta por <IP remoto, IP, local, porta remota, porta local, e o protocolo (TCP, UDP, por exemplo)>

- a. Apenas as afirmativas II e III são verdadeiras
- b. Apenas a afirmativa I é verdadeira
- c. Apenas as afirmativas I e II são verdadeiras
- d. Nenhuma das opções corresponde às alternativas apresentadas
- e. Apenas as afirmativas I e III são verdadeiras

Sua resposta está correta.

A resposta correta é:
Nenhuma das opções corresponde às alternativas apresentadas

Questão 2

Completo

Atingiu 1,3 de 1,5

O código MPI a seguir faz a impressão colaborativa de um vetor dinâmico, no qual *(i)* o MASTER faz a inicialização do vetor e *(ii)* cada processo faz a impressão de uma quantidade de elementos desse vetor, de acordo com o número de processos ativado. Percebe-se, no entanto, que esse programa possui um problema de alto consumo de memória, uma vez que todos os processos são obrigados a alocar o vetor completo (linha 15), embora façam a impressão de apenas um subconjunto desse vetor (para um vetor de 1 milhão de inteiros e 5 processos, serão alocados 5 milhões de inteiros, gerando alocação excessiva de memória). Promova as alterações nesse código, de modo a reduzir o consumo de memória e, ao mesmo tempo, garantir a impressão equitativa do vetor entre os processos.

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <mpi.h>
4  #define MASTER 0
5  int main(int argc, char* argv[]) {
6      int rank, nprocs, *v, tamvet;
7      MPI_Init(&argc, &argv);
8      MPI_Comm_rank(MPI_COMM_WORLD, &rank);
9      MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
10     if (rank == MASTER) {
11         printf("Tamanho do vetor: ");
12         fflush(stdout); scanf("%d", &tamvet);
13     } /* fim-if */
14     MPI_Bcast(&tamvet, 1, MPI_INT, MASTER, MPI_COMM_WORLD);
15     v = (int *) malloc(tamvet * sizeof(int));
16     if (rank == MASTER) {
17         for (int i=0; i< tamvet; i++)
18             v[i]=(i+1)*10;
19     } /* fim-if */
20     MPI_Bcast(v, tamvet, MPI_INT, MASTER, MPI_COMM_WORLD);
21     int chunk = tamvet/nprocs; int ini = rank * chunk;
22     if (rank == (nprocs-1))
23         chunk=chunk+(tamvet%nprocs);
24     int fim = ini + chunk;
25     printf("%d/%d: ", rank, nprocs);
26     for (int i=ini; i<fim; i++)
27         printf("%d ", v[i]);
28     printf("\n");
29     MPI_Finalize();
30     return 0;
31 } /* fim-main */
```

Questão 3

Completo

Atingiu 1,0 de 1,0

Observe o seguinte código MPI, cujo objetivo é conseguir gravar 100 elementos do vetor data em arquivo:

```
1 #include <mpi.h>
2 #include <stdio.h>
3 #include <stdlib.h>
4 #define FILE_NAME "file.bin"
5 #define MAX 100
6
7 int main(int argc, char** argv) {
8   int rank, size;
9   MPI_Init(NULL, NULL);
10  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
11  MPI_Comm_size(MPI_COMM_WORLD, &size);
12  int data[MAX];
13  MPI_File fh;
14
15  int chunk = MAX / size;
16  int start = rank * chunk * sizeof(int);
17  if (rank == (size-1)) chunk+=(MAX%size);
18  for (int i = 0; i < chunk; i++)
19    data[i] = rank * chunk + i + 1;
20  MPI_File_open(MPI_COMM_WORLD, FILE_NAME, MPI_MODE_CREATE | MPI_MODE_WRONLY, MPI_INFO_NULL, &fh);
21  MPI_File_write_at(fh, start, data, chunk, MPI_INT, MPI_STATUS_IGNORE);
22  MPI_File_close(&fh);
23  MPI_Finalize();
24 } /* fim-main */
```

Considerando o propósito definido para o código, avalie as afirmativas e, a seguir, marque a opção correta:

I - O código não funciona adequadamente porque a função da linha 21 necessita um laço para garantir que cada processo faça a escrita dos elementos sob sua responsabilidade na posição correta do arquivo

II - Este código funciona adequadamente e a instrução da linha 17 garante que os valores sequenciais, de 1 a 100, no vetor data, independente do número de processos

III - O código apresentado não funciona adequadamente porque a função de escrita (linha 21) exige que o vetor a ser gravado seja dividido em partes iguais entre os processos MPI

- a. Apenas a afirmativa II está correta
- b. Apenas as afirmativas I e III estão corretas
- c. Apenas a afirmativa I está correta
- d. Apenas a afirmativa III está correta
- e. Nenhuma das alternativas está correta

Questão 4

Completo

Atingiu 1,0 de 1,0

Analise o código a seguir.

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <cuda_runtime.h>
4  #include <cuda.h>
5
6  __global__ void vecAdd(int *c) {
7      int i=threadIdx.x;
8      int j=blockIdx.x;
9      int k=blockDim.x;
10
11     c[i] = i+j+k;
12 } /* fim vecAdd */
13
14 int main(int argc, char *argv[]) {
15     int *c, *dc;
16     int n=atoi(argv[1]);
17     int size = n * sizeof(int);
18
19     cudaDeviceReset();
20     c = (int*) malloc(size);
21     cudaMalloc((void **) &dc, size);
22     vecAdd <<<2,n/2>>> (dc);
23     cudaDeviceSynchronize();
24     cudaMemcpy(c, dc, size, cudaMemcpyDeviceToHost);
25
26     printf("Resultado:\n");
27     for (int i=0; i<n; i++)
28         printf(" %d ", c[i]);
29
30     printf("\n");
31     cudaFree(dc);
32
33     return 0;
34 } /* fim-main */
```

Suponha que n (linha 16) é igual a 10, analise as afirmativas a seguir e marque a alternativa INCORRETA.

- a. Se houver substituição do comando da linha 11 por c[k] = i+j+k;, o vetor a ser impresso é 0 0 0 0 0 6 0 0 0 0
- b. Da forma como está, o vetor a ser impresso é 6 7 8 9 10 0 0 0 0 0
- c. Se houver substituição do comando da linha 11 por c[j] = i+j+k;, o vetor a ser impresso é 5 6 0 0 0 0 0 0 0 0
- d. Se for utilizada a fórmula c[i*(j+k)]=i+j+k, o vetor a ser impresso não é 5 6 7 8 9 0 0 0 0 0
- e. Se for utilizada a fórmula c[i+(j*k)]=i+j+k, o vetor a ser impresso é 0 0 0 0 0 5 6 7 8 9

Questão 5

Completo

Atingiu 0,0 de 1,0

Analise as afirmativas a seguir e marque a alternativa correta.

I - A comunicação do tipo publish/subscribe é apropriada para distribuir serviços entre vários processos, mas possui restrições para implementar balanceamento de carga em clusters de servidores.

II - Brokers AMQP são flexíveis a ponto de permitirem que processos se comuniquem usando uma mesma fila/stream ou filas/streams separadas, dependendo do tipo do problema.

III - Um sistema publish/subscribe viabiliza buffers temporários e formatação de mensageria, que são mecanismos muito apropriados para comunicações do tipo transiente.

a. Apenas I está correta

b. Apenas II e III estão corretas

c. Apenas I e III estão corretas

d. Nenhuma das respostas

e. Apenas III está correta

Sua resposta está incorreta.

A resposta correta é:
Apenas II e III estão corretas

Questão 6

Completo

Atingiu 0,4 de 2,5

Carrossel de *threads* OpenMP

Dado um valor n de entrada, construa um programa OpenMP que produza como saída a soma dos naturais no intervalo de [0 ... n-1] multiplicado pelo thid (identificador da *thread*) responsável por cada número natural da série. Considerar ainda que a primeira *thread* a iniciar o processo de soma deve ser escolhida pela *thread* MASTER (via função randômica).

- O valor n de entrada representa os naturais de 0 a [n-1]. Por exemplo, se *n*=8, a série a ser considerada é 0 1 2 3 4 5 6 7.
- A *thread* MASTER (zero) não participa do processo de soma. Por exemplo, se OMP_NUM_THREADS=5, apenas as *threads* 1, 2, 3 e 4 trabalhadoras participarão do carrossel de threads para a operação de soma.
- As *threads* trabalhadoras devem funcionar em ordem circular, de modo a percorrer todos os números da série estabelecida no programa. Por exemplo, supondo OMP_NUM_THREADS=4 e N=10, o carrossel será formado pelas *threads* 1, 2 e 3, e cada uma delas cuidará de mais de um dos números da série.
- A thread MASTER deve escolher, aleatoriamente, qual das *threads* trabalhadoras iniciará o processo de soma. Como exemplo, assumindo N = 8 e OMP_NUM_THREADS=5, se a Master escolher a *thread* trabalhadora 2 para iniciar o carrossel, a soma ficará assim: Soma = 0*2 + 1*3 + 2*4 + 3*1 + 4*2 + 5*3 + 6*4 + 7*1 = 68. Obs.: Perceba que cada termo da soma é o produto de um elemento da série pelo thid da *thread* do carrossel.
- O valor de saída refere-se à soma calculada pelo produto dos naturais da série pelos *thids* das *threads* do carrossel.
- O tipo int é suficiente para representar o valor a ser impresso na saída (considere que todos os casos de teste produzem somas que cabem numa variável do tipo int)

Entrada

O arquivo de entrada contém n>0

Saída

O arquivo de saída contém o valor relativo à soma da lista de números naturais do intervalo [0 ... (n-1)] multiplicados pelo thid do carrossel de *threads*.

Obs.: o arquivo de código deve deve ser nomeado com a matrícula do aluno, por exemplo, 2019000894.c

Questão 7

Completo

Atingiu 1,0 de 1,0

Analise o programa a seguir:

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <omp.h>
#define MAX 14

int main(int argc, char*argv[]){
    int sum=0;

    #pragma omp parallel for reduction(+:sum) schedule(runtime)
    for (int i=1; i<=MAX; i++) {
        printf("%4d @ %d\n", i, omp_get_thread_num());
        sleep(i<4? i+1:1);
        sum+=i;
    } /* fim-for */
    printf("Soma = %d\n", sum);
    return 0;
} /*fim-main*/
```

Sobre este código, julgue as afirmações feitas a seguir:

I - O escalonamento com *static,1* produz um desempenho pior do que escalonamento com *dynamic,1*

II - O escalonamento com *static,2* produz um desempenho melhor do que o escalonamento com *static,1*

III - O escalonamento com *static,3* tem desempenho pior do que o escalonamento com *dynamic,2*

Agora marque a alternativa CORRETA:

- a. As afirmativas I e III são verdadeiras
- b. Nenhuma das opções corresponde às afirmativas apresentadas
- c. Apenas a afirmativa I é verdadeira
- d. Apenas a afirmativa II é verdadeira
- e. Apenas a afirmativa III é verdadeira

Questão 8

Completo

Atingiu 1,0 de 1,0

Analise o código a seguir e responda o que se segue

```
1   #include <stdio.h>
2   #include <omp.h>
3   int main(){
4       int tid=0, nthreads=0;
5       printf("\nRegião serial (thread única)\n\n");
6       #pragma omp parallel
7       {
8           tid      = omp_get_thread_num();
9           nthreads = omp_get_num_threads();
10          printf("Região paralela (thread %d de %d threads)\n", tid, nthreads);
11      } /*fim-pragma */
12      printf("\nRegião serial (thread única)\n\n");
13      #pragma omp parallel num_threads(4)
14      {
15          tid = omp_get_thread_num();
16          nthreads = omp_get_num_threads();
17          printf("Região paralela (thread %d de %d threads)\n", tid, nthreads);
18      } /* fim pragma */
19      printf("\nRegião serial (thread única)\n\n");
20      return 0;
21  } /* fim-main */
```

1. Se OMP_NUM_THREADS=6, na segunda região paralela desse código (linhas 13 a 18), serão geradas 10 threads e, portanto, 10 impressões (linha 17)
2. Se a linha 15 for movida para ficar fora da região paralela (entre as linhas 11 e 13), esse código passa a ser não compilável, pois não é possível saber o número de threads em uma região serial do código
3. Esse código é mais apropriado para funcionar em arquiteturas UMA (Uniform Memory Access) ou de memória compartilhada do que em arquiteturas NUMA (Non Uniform Memory Access)

a. Apenas a segunda e a teroeira afirmação está correta

b. Apenas a primeira e a terceira afirmação está correta

c. Apenas a primeira afirmação está correta

d. Nenhuma das alternativas apresentadas é válida

e. Apenas a terceira afirmação está correta