Iniciado em	segunda-feira, 11 dez. 2023, 14:10
Estado	Finalizada
Concluída em	segunda-feira, 11 dez. 2023, 15:29
Tempo empregado	1 hora 19 minutos
Avaliar	**3,30** de um máximo de 10,00(**33**%)

Questão 1

Completo

Atingiu 1,00 de 2,00

O programa abaixo trabalha com vetores de tamanho variável localmente. Com uso da biblioteca RPC, gere uma estrutura equivalente na qual o programa principal e a função chamada (somavet) residam em hosts distintos.

Obs.: Sugere-se criar uma pasta contendo os arquivos da sua aplicação. Essa pasta deve ser compactada e enviada como resposta para essa questão.

```
1  int somavet(int *x, int n) {
2        int i, result;
3        printf("Requisicao para adicao de %d numeros\n", n);
4        result=0;
5        for (i=0; i<n; i++) {
6          result += x[i];
7        } /* fim-for */
8        return (result);
9  } /* fim somavet */
10 int main( int argc, char *argv[]) {
11   int *ints, n_termos, i, res;
12   if (argc<2) {
13     fprintf(stderr,"Uso correto: %s num1 num2 ...\n",argv[0]);
14     exit(0); }
15     /* recupera os numeros que devem ser adicionados */
16   n_termos = argc-1;
17   ints = (int *) malloc(n_termos * sizeof( int ));
18   if (ints==NULL) {
19     fprintf(stderr,"Erro na alocacao de memoria\n");
20     exit(0); }
21 /* preenche a estrutura dinamica com os valores informados pelo usuario */
22   for (i=1;i<argc;i++) {
23     ints[i-1] = atoi(argv[i]);
24   } /* fim-for */
25 /* imprime o resultado da soma */
26   res = somavet(ints, n_termos);
27   printf("%d",ints[0]);
28   for (i=1; i<n_termos; i++) {
29     printf(" + %d",ints[i]);
30   } /* fim-for */
31   printf(" = %d\n",res);
32   return(0);
33 } /* fim programa principal */
34
35
36
```

q1SomaVet.zip

Comentário:

Questão 2

Incorreto

Atingiu 0,00 de 1,00

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 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

b. Nenhuma das alternativas está ERRADA

c. Da forma como está, o vetor a ser impresso é 6 7 8 9 10 0 0 0 0 0

d. 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

e. 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 ✖

Sua resposta está incorreta.

A resposta correta é:
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 3

Incorreto

Atingiu 0,00 de 1,00

Analise as afirmativas abaixo e marque a opção correta

I - No modelo P2P, o DHT (***Distributed Hash Table***) é uma técnica de localização de peers que se caracteriza por ter boa performance em redes com grande quantidade de peers que podem sair e entrar na rede a qualquer instante

II - O modelo P2P é um tipo de arquitetura na qual os peers são máquinas que possuem a função de serem servidoras e clientes ao mesmo tempo, como uma se fosse uma variante do modelo Cliente/Servidor tradicional das aplicações de rede TCP/IP

III - Algoritmos de consenso derivados do Paxos, como o Zookeeper e o Raft tem, como uma de suas funções, a sincronização dos logs de execução entre os peers de uma rede P2P

a. Apenas I está correta

b. Apenas I e II estão corretas ✖

c. Apenas II e III estão corretas

d. Apenas II está correta

e. Apenas III está correta

Sua resposta está incorreta.

A resposta correta é:
Apenas III está correta

Questão 4

Incorreto

Atingiu 0,00 de 1,00

O código a seguir foi compilado com OpenMP e o binário tem o nome t1. Com base no código fonte, analise as afirmações e marque V para as verdadeiras e F para as falsas.

```
1   #include <stdio.h>
2   #include <omp.h>
3
4   int main(void) {
5           printf(" %d\n", omp_get_num_threads());
6           #pragma omp parallel
7           {
8
9           }
10          printf("%d\n", omp_get_max_threads());
11          return 0;
12  }
```

I - A execução com o comando **OMP_NUM_THREADS=4 t1** vai imprimir o valor 4 na linha 5 e o valor 12 na linha 10, se o computador onde esse programa estiver rodando tiver 12 núcleos

II - Este programa vai imprimir sempre o valor 1 na linha 5, independente do número de threads definidas na variável OMP_NUM_THREADS

III - O comando da linha 10 vai imprimir sempre o valor 1, uma vez que este está fora da região paralela definida pelo **pragma omp parallel**

- a. Apenas a afirmativa I está correta
- b. Apenas as afirmativas I e II estão corretas ✖
- c. Apenas as afirmativas II e III estão corretas
- d. Apenas as afirmativas I e III estão corretas
- e. Nenhuma das opções consegue julgar as afirmativas apresentadas

Sua resposta está incorreta.

A resposta correta é:
Nenhuma das opções consegue julgar as afirmativas apresentadas

Questão 5

Incorreto

Atingiu 0,00 de 1,00

O código a seguir foi compilado com OpenMP e o binário tem o nome t1. Com base no código fonte, analise as afirmações e marque V para as verdadeiras e F para as falsas.

```
1   #include <stdio.h>
2   #include <omp.h>
3
4   int main(int argc, char *argv[]) {
5
6           int i=0;
7           #pragma omp parallel
8           {
9                   if (omp_get_thread_num() == 1)
10                          i=i+10;
11          }
12          printf("i=%d\n", i);
13          return 0;
14  }
```

I - A execução com o comando **OMP_NUM_THREADS=1 t1** vai imprimir o valor zero para a variável i

II - Se na linha 7 for acrescentada a declaração **private(i)**, o binário equivalente acionado com o comando **OMP_NUM_THREADS= 2 t1** vai imprimir o valor 20

III - Ao suprimir a linha 9, o binário equivalente vai imprimir valores aleatórios para a variável i, desde que o número de threads seja maior que 1

a. Nenhuma das opções apresentadas consegue julgar corretamente as afirmativas

b. Apenas a afirmativa I está correta ✖

c. Apenas as afirmativas II e III estão corretas

d. Apenas as afirmativas I e III estão corretas

e. Apenas as afirmativas I e II estão corretas

Sua resposta está incorreta.

A resposta correta é:
Apenas as afirmativas I e III estão corretas

Questão 6

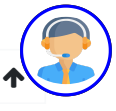
Completo

Atingiu 2,00 de 2,00

Elabore um programa MPI com N processos, sendo o master o responsável por inicializar o vetor e os slaves, responsáveis por imprimir uma porção do vetor, proporcional ao número de slaves identificados pelo programa.

```
/*
Como compilar e executar:
$ mpicc q6.c -o saida

$ mpirun -n 3 ./saida

chunk: index = offset * (index - 1)
num worker_id offset index
0 1 3 3 * (worker_id - 1) = 0
1 1 3 3 * (worker_id - 1) = 0
2 1 3 3 * (worker_id - 1) = 0
3 2 3 3 * (worker_id - 1) = 3
4 2 3 3 * (worker_id - 1) = 3
5 2 3 3 * (worker_id - 1) = 3
*/
#include <stdio.h>
#include <mpi.h>
#include <unistd.h>
#include <stdlib.h>

#define DEBUG 1
#define TAMANHO_VETOR 16
#define MASTER 0
#define ESCRAVO 1

// define o tipo da mensagem
#define TAG 0

int main(int argc, char **argv) {
MPI_Init(&argc, &argv);
// obtém o id do processo
int myRank;
int size;

int vet[TAMANHO_VETOR];

// inicializa o rank
MPI_Comm_rank(MPI_COMM_WORLD, &myRank);

// define o tamanho do rank
MPI_Comm_size(MPI_COMM_WORLD, &size);

// Tamanho do vetor é dividido pela quantidade de workers
int tamanho_chunk = TAMANHO_VETOR / (size - 1);
int tamanho_vetor_aux = tamanho_chunk;

// Caso a divisão do vetor seja inexata, o ultimo chunk deve ser maior
int resto_div = (TAMANHO_VETOR % (size - 1));

if (myRank == 0) {
// Estamos no MASTER
```

```
int offset = tamanho_chunk;

// Inicializa o vetor
for (int i = 0; i < TAMANHO_VETOR; i++)
vet[i] = i;
// envio do vetor para cada worker
for (int worker_id = 1; worker_id < size; worker_id++) {
int index = offset * (worker_id - 1);

// Verificamos o tamanho do vetor no ultimo worker
if (worker_id == (size - 1)) {
if (resto_div != 0) {
tamanho_chunk += resto_div;
tamanho_vetor_aux = tamanho_vetor_aux + resto_div;
offset = offset + resto_div;

#ifndef DEBUG
printf("Tamanho do ultimo chunk: %d, tamanho_vetor_aux: %d\n", tamanho_chunk, tamanho_vetor_aux);
#endif
}
}

int *aux = malloc(sizeof(int) * tamanho_chunk);

for (int i = index, j = 0; i < (index + offset); i++, j++) {
aux[j] = vet[i];
#ifndef DEBUG
printf("vet[%d] = %d\n", i, vet[i]);
printf("aux[%d] = %d\n", j, vet[i]);
#endif
}

printf("MESTRE[0]: enviando o vetor [");

for (int i = 0; i < tamanho_vetor_aux; i++)
printf(" %d ", aux[i]);
printf("] de tamanho %d para o WORKER[%d]\n", tamanho_vetor_aux, worker_id);

MPI_Send(aux, tamanho_vetor_aux, MPI_INT, worker_id, TAG, MPI_COMM_WORLD);
free(aux);
}
} else {
// estamos no nó escravo
int *aux = NULL;

if ((myRank == size - 1) && resto_div != 0) {
aux = malloc(sizeof(int) * (tamanho_vetor_aux + resto_div));
tamanho_vetor_aux += resto_div;
} else {
aux = malloc(sizeof(int) * tamanho_vetor_aux);
}

MPI_Recv(aux, tamanho_vetor_aux, MPI_INT, MASTER, TAG, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
printf("WORKER[%d]: recebeu do mestre:\n", myRank);

// Imprime o vetor recebido
for (int i = 0; i < tamanho_vetor_aux; i++) {
printf("%d ", aux[i]);
}
printf("\n");

free(aux);
}
```



```
MPI_Finalize();
return 0;
}
```

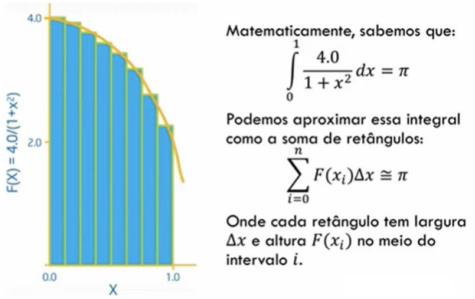
Comentário:

Questão 7

Completo

Atingiu 0,30 de 2,00

O programa a seguir é uma implementação do cálculo da função Pi em modo serial, usando único espaço de endereçamento. Com base neste código crie uma versão CUDA, de modo que o valor de PI seja obtido por cálculos realizados na GPU, obedecendo a melhor relação possível entre blocos e threads por bloco.



Matematicamente, sabemos que:

$$\int_0^1 \frac{4.0}{1+x^2} dx = \pi$$

Podemos aproximar essa integral como a soma de retângulos:

$$\sum_{i=0}^{n} F(x_i)\Delta x \cong \pi$$

Onde cada retângulo tem largura Δx e altura $F(x_i)$ no meio do intervalo i.

```c
#include <stdio.h>
#define NUM_STEPS 8000000

int main(void) {
    double x, pi, sum=0.0;
    double step;

    step = 1.0/(double) NUM_STEPS;
    for (int i=0;i<NUM_STEPS; i++) {
        x = (i+0.5) * step;
        sum+=4/(1.0+x*x);
    } /*fim-for */
    pi = sum*step;
    printf("Pi = %f\n", pi);
} /*fim-main */
```

```
/*
$ nvcc q7.cu -o q7
$ ./q7
*/

#include <stdio.h>

#define NUM_STEPS 8000000
#define THREADS_BLOCK 256

__global__ void calculatePi(int *sum_d) {
int i = threadIdx.x + blockIdx.x * blockDim.x;
double x;
double step = 1.0 / (double)NUM_STEPS;

if (i < NUM_STEPS) {
x = (i + 0.5) * step;
atomicAdd(sum_d, static_cast<int>(4.0 / (1.0 + x * x) * 1e9));
}
}

int main() {
int *sum_h, *sum_d;
double pi;

sum_h = (int *)malloc(sizeof(int));
cudaMalloc((void **)&sum_d, sizeof(int));

*sum_h = 0;
cudaMemcpy(sum_d, sum_h, sizeof(int), cudaMemcpyHostToDevice);
int num_blocks = (NUM_STEPS + THREADS_BLOCK - 1) / THREADS_BLOCK;

calculatePi<<<num_blocks, THREADS_BLOCK>>>(sum_d);
cudaMemcpy(sum_h, sum_d, sizeof(int), cudaMemcpyDeviceToHost);

pi = static_cast<double>(*sum_h) / 1e9;
printf("Valor de Pi = %f\n", pi);

free(sum_h);
```

```
cudaFree(sum_d);

return 0;
}
```

q7.c

Comentário: