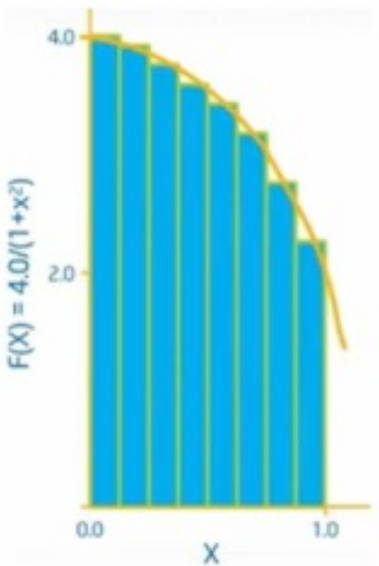
Questão 1

Resposta salva

Vale 1,50 ponto(s).

Marcar questão

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 MPI, de modo a permitir que os processos, em conjunto e colaborativamente, consigam encontrar o valor de Pi, pela distribuição do cálculo das áreas dos retângulos entre os processos.



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 */
```

Tempo restante 0:08:46

Questão 2

Ainda não respondida

Vale 0,75 ponto(s).

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   #include <string.h>
4   #define MAX 100
5
6   int main(int argc, char *argv[]) {
7
8           #pragma omp parallel
9           {
10                  int soma=0;
11                  #pragma omp for
12                  for (int i=0;i<MAX;i++) {
13                    soma +=omp_get_num_threads()*i;
14                  } /* fim-for */
15                  printf("Thread[%d] iterou %d vezes\n",
16                                  omp_get_thread_num(), soma);
17          } /* fim omp parallel */
18          return 0;
19  }
```

I - A execução com o comando **OMP_NUM_THREADS=4 t1** vai imprimir que cada thread foi executada 25 vezes
II - Se este programa for acionado tendo a variável OMP_NUM_THREADS um valor maior do que o número de núcleos da máquina, apenas as threads equivalentes ao número de núcleos serão criadas
III - Se o programa for executado numa máquina com 10 núcleos de processamento e a variável OMP_NUM_THREADS estiver com valor igual a 20, o programa não será ativado

- a. Apenas as afirmativas II e III estão corretas
- b. Apenas a afirmativa III está correta
- c. Todas as afirmativas estão erradas
- d. Apenas a afirmativa I está correta
- e. Apenas a afirmativa II está correta

Tempo restante 0:08:36

Questão 3

Ainda não respondida

Vale 0,75 ponto(s).

Analise as afirmações a seguir e marque a alternativa correta

I - O problema dos generais bizantinos é uma metáfora que descreve a dificuldade de se entrar em um acordo quando entidades centralizadas decidem em nome da maioria

II - A tecnologia blockchain é uma solução eficiente para o problema dos generais bizantinos

III - O algoritmo Paxos é uma solução de consenso distribuído cuja variante pode ser usada para coordenação e resolução de impasses em redes blockchain

a. Todas estão corretas
b. Apenas I e III estão corretas
c. Apenas I e II estão corretas
d. Apenas II está correta
e. Apenas II e III estão corretas

Questão 4

Ainda não respondida

Vale 1,50 ponto(s).

No código a seguir, os pragmas declarados nas linhas 11 e 14 garantem a divisão equilibrada do trabalho entre o total de threads especificadas na variável de ambiente OMP_NUM_THREADS.

```
1   #include <stdio.h>
2   #include <omp.h>
3   #define TAM 12
4   int main () {
5       int A[TAM], B[TAM], C[TAM];
6       int i;
7       for (i=0; i<TAM; i++) {
8           A[i]=2*i - 1;
9           B[i]=  i + 2;
10      }
11      #pragma omp parallel
12      {
13          int tid = omp_get_thread_num();
14          #pragma omp for
15          for (i=0; i<TAM; i++) {
16              C[i] = A[i]+ B[i];
17              printf("Thread[%d] calculou C[%d]\n", tid, i);
18          } /* fim-for */
19      } /* fim-pragma */
20      for (i=0; i<TAM; i++)
21          printf("C[%d]=%d\n", i, C[i]);
22  } /* fim-main */
23
```

Apresente uma nova versão desse código que garanta a distribuição equilibrada de trabalho entre as threads (de acordo com o valor de OMP_NUM_THREADS), considerando apenas o pragma de paralelização descrito na linha 11 (ou seja, assuma a não existência do pragma da linha 14).

Tamanho máximo para arquivos: 10 Kb, número máximo de anexos: 2

Arquivos

Você pode arrastar e soltar arquivos aqui para adicioná-los.

Tempo restante 0:08:13

Questão 5

Resposta salva

Vale 0,75 ponto(s).

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

LIMPAR MINHA ESCOLHA

Tempo restante 0:08:05

Questão 6

Resposta salva

Vale 0,75 ponto(s).

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. Nenhuma das opções consegue julgar as afirmativas apresentadas
- c. Apenas as afirmativas II e III estão corretas
- d. Apenas as afirmativas I e II estão corretas
- e. Apenas as afirmativas I e III estão corretas

LIMPAR MINHA ESCOLHA

Questão 7

Resposta salva

Vale 1,50 ponto(s).

Elabore um programa OpenMP que imprima os elementos de uma matriz M [100:8] posições (do tipo int), considerando o seguinte:

- Apenas uma das threads deve inicializar a matriz, com a fórmula: M[i][j]=i+j
- A matriz deve ser impressa de modo colaborativo por todas as threads ativadas, da linha 0 até a linha 99, nesta ordem
- Cada thread deve imprimir pelo menos uma das linhas da matriz
- Cada thread, uma vez acionada, deve imprimir a matriz a partir da linha que ainda não foi impressa
- Considerar que este programa pode ser executado por, no máximo, 6 threads
- O número de posições a serem impressas deve obedecer a um ***offset*** dinâmico, ou seja, um valor randômico – menor que 15 – que é calculado por cada thread, no momento em que a thread é escalonada para imprimir a matriz
- O programa deve controlar a impressão de modo que a matriz inteira seja impressa em ordem e de modo que nenhuma posição seja impressa mais de uma vez. Por exemplo, se a matriz já foi impressa até a linha 18 e o ***offset*** dinâmico calculado foi 10, a thread atual deve imprimir da linha 19 considerando 10 posições adiante
- O programa termina quando a matriz M inteira, linha por linha, tiver sido impressa.

Tempo restante 0:07:48

Questão 8

Ainda não respondida

Vale 0,75 ponto(s).

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=4 t1** vai imprimir o valor 40

II - Se a linha 9 for suprimida, o binário equivalente acionado com o comando **OMP_NUM_THREADS=3 t1** imprimirá sempre o valor 30

III - Se na linha 7 for acrescentada a declaração **private(i)** e houver supressão da linha 9, o binário equivalente acionado com o comando OMP_NUM_THREADS=6 t1, o programa vai imprimir 60

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

Questão 9

Resposta salva

Vale 1,00 ponto(s).

Por quê os algoritmos de hash consistente são necessários em redes P2P? Que tipo de problemas essa algoritmo resolve? Na resposta, apresentar um exemplo esclarecendo o funcionamento do algoritmo.

Tempo restante 0:07:20

Questão 10

Ainda não respondida

Vale 0,75 ponto(s).

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 primeira afirmação está correta
- b. Apenas a segunda e a terceira afirmação está correta
- c. Nenhuma das alternativas apresentadas é válida
- d. Apenas a terceira afirmação está correta
- e. Apenas a primeira e a terceira afirmação está correta