UNIVERSIDADE FEDERAL DO CEARÁ

Computação Paralela - Atividade 02

Nome: João Mateus Dias do Carmo
Matrícula: 390187
Curso: Engenharia de computação

Repositório da Disciplina

Quixadá-CE

Questão 1: Dado o código abaixo.

```
double **mtxMul (double **c, double **a, double **b, int n) {
  for (int i = 0; i < n; i++)
     for (int j = 0; j < n; j++) {
         c[i][j] = 0.0;
         for (int k = 0; k < n; k++)
            c[i][j] = c[i][j] + a[i][k] * b[k][j];
      }
  return c;
}
```

Pela forma serial e sabendo que o ***n*** pedido na questão é 100, podemos calcular a quantidade de iterações que ocorrerá no ***for*** mais interno.

NumeroIte= 100x100x100

NumeroIte=1000000 ou 10^6;

É pedido que faça alterações no código acima para 8 threads.

1. **Resposta (Apenas o laço for mais externo é paralelizado):**

```
double **mtxMul (double **c, double **a, double **b, int n) {
  #pragma omp parallel for
  for (int i = 0; i < n; i++)
     for (int j = 0; j < n; j++) {
         c[i][j] = 0.0;
         for (int k = 0; k < n; k++)
            c[i][j] = c[i][j] + a[i][k] * b[k][j];
      }
  return c;
}
```

Se apenas o primeiro laço for paralelizado da forma mostrada no código temos que fazer a divisão mais igualitária possível entre as 8 threads, fazendo a divisão inteira obtemos que cada thread irá ter uma carga de 12 execuções, mas 4 dessas threads terão um ciclo a mais para cumprir.

Cada ciclo acima terá uma carga de 100x100 no laço mais interno, logo temos a relação mostrada na tabela abaixo.

Definindo abaixo a coluna Relação Numérica como sendo a carga para cada thread e os termos ***q*** e ***r*** como sendo quociente e resto respectivamente.

Threads	Relação numérica(q+r)	Valor de carga para cada execução
Thread 0:	12+1	13x(100x100)
Thread 1:	12+1	13x(100x100)
Thread 2:	12+1	13x(100x100)
Thread 3:	12+1	13x(100x100)
Thread 4:	12+0	12x(100x100)
Thread 5:	12+0	12x(100x100)
Thread 6:	12+0	12x(100x100)
Thread 7:	12+0	12x(100x100)
Total:	Σ=100	$\Sigma = 4 \cdot 10^4(13 + 12) = 10^6$

2. **Resposta (A diretiva collapse(2) é aplicada ao laço mais externo):**

```
double **mtxMul (double **c, double **a, double **b, int n) {
  #pragma omp parallel for collapse(2)
  for (int i = 0; i < n; i++)
      for (int j = 0; j < n; j++) {
          c[i][j] = 0.0;
          for (int k = 0; k < n; k++)
              c[i][j] = c[i][j] + a[i][k] * b[k][j];
       }
  return c;
}
```

Usando a diretiva collapse(2) no primeiro ***for*** será feito uma junção dos dois primeiros ***for***, assim seguindo a mesma lógica do primeiro exemplo, dividimos a quantidade de execuções entre as 8 threads, assim temos a tabela.

Threads	Relação numérica	Valor de carga para cada execução
Thread 0:	1250	1250x100
Thread 1:	1250	1250x100
Thread 2:	1250	1250x100
Thread 3:	1250	1250x100
Thread 4:	1250	1250x100
Thread 5:	1250	1250x100
Thread 6:	1250	1250x100
Thread 7:	1250	1250x100
Total:	Σ=10000	$\Sigma = 8 \cdot 1250 \cdot 100$

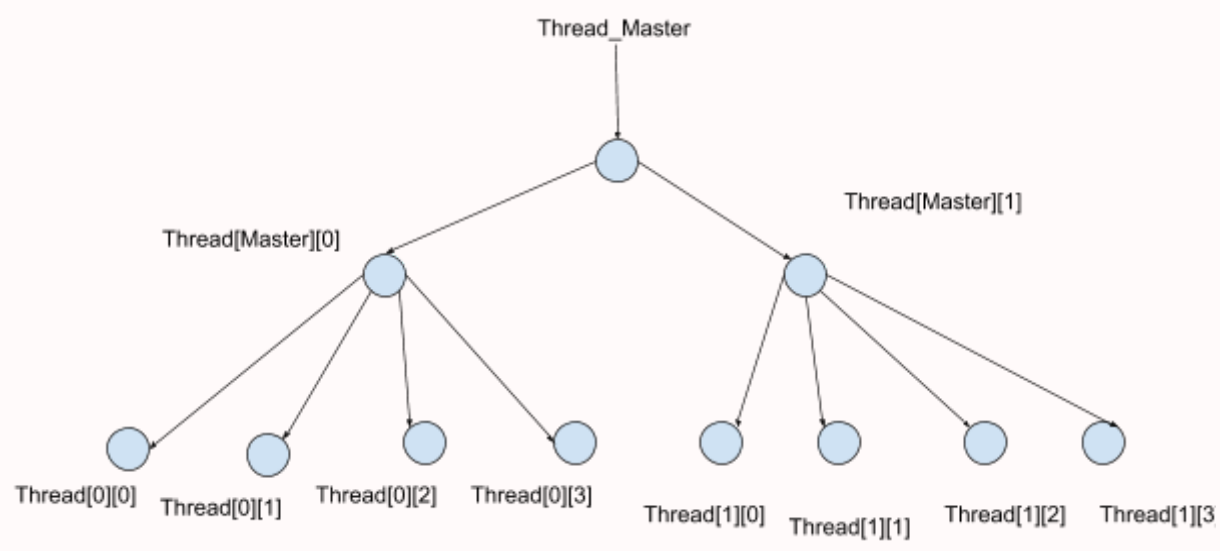
3. **Resposta (Ativando regiões paralelas aninhadas e OMP_NUM_THREADS=2,4):**

```
double **mtxMul (double **c, double **a, double **b, int n) {
  #pragma omp parallel for
  for (int i = 0; i < n; i++)
    #pragma omp parallel for
     for (int j = 0; j < n; j++) {
         c[i][j] = 0.0;
         for (int k = 0; k < n; k++)
             c[i][j] = c[i][j] + a[i][k] * b[k][j];
      }
  return c;
}
```

Para a resolução desta questão irei usar uma nomenclatura diferente das anteriores para a melhor representação das criações das threads aninhadas. Sendo Thread([id_thread_pai],[id_current]) sendo a thread pai de todas chamada de ThreadMaster.

A forma que eu encontrei para a resolução desta questão foi usando uma árvore de criação das threads.

Sendo assim para a análise de cada etapa de criação temos.



Threads	Relação numérica(q+r)	Valor de carga para cada execução
Thread[0][0]:	12+1	13x(100x100)
Thread[0][1]:	12+1	12x(100x100)
Thread[0][2]:	12+0	12x(100x100)
Thread[0][3]:	12+0	13x(100x100)
Thread[1][0]:	12+1	13x(100x100)
Thread[1][1]:	12+1	13x(100x100)
Thread[1][2]:	12+0	12x(100x100)
Thread[1][3]:	12+0	12x(100x100)
Total:	Σ=100	$\Sigma = 4 \cdot 10^4(13 + 12) = 10^6$

Questão 2: Dado o código abaixo.

```
#include <stdio.h>
#include <omp.h>
#include <unistd.h>

int main (int argc , char *argv[]) {
    int max;
    sscanf (argv[1], "%d", &max);
    long int sum = 0;
    #pragma omp parallel for reduction(+:sum) schedule(runtime)
    for (int i = 1; i <= max; i++) {
        printf ("%2d @ %d\n", i, omp_get_thread_num());
        sleep (i < 4 ? i + 1 : 1);
        sum = sum + i;
    }
    printf ("%ld\n", sum);
    return 0;
}
```

1. **Resposta 1 (static,1, static,2 e static,3):**

   1.1. **static,1:**

Thread id	Valores executados por cada thread.
Thread 0	{1;9;13;17}
Thread 1	{2;6;10;14;18}
Thread 2	{3;7;11;15;19}
Thread 3	{4;8;12;16;20}

   1.2. **static,2:**

Thread id	Valores executados por cada thread.
Thread 0	{(1,2),(9,10),(17,18)}
Thread 1	{(3,4),(11,12),(19,20)}
Thread 2	{(5,6),(13,14)}
Thread 3	{(7,8),(15,16)}

1.3. static,3:

Thread id	Valores executados por cada thread.
Thread 0	{(1,2,3),(13,14,15)}
Thread 1	{(4,5,6),(16,17,18)}
Thread 2	{(7,8,9),(19,20)}
Thread 3	{(10,11,12)}

2. Resposta 2 (dynamic,1, dynamic,2 e dynamic,3):

2.1. dynamic,1:

Thread id	Valores executados por cada thread.
Thread 0	{3,11,15,19}
Thread 1	{1,6,9,13,17}
Thread 2	{4,5,7,10,14,18}
Thread 3	{2,8,12,16,20}

2.2. dynamic,2:

Thread id	Valores executados por cada thread.
Thread 0	{(5,6),(9,10),(15,16)}
Thread 1	{(3,4),(17,18)}
Thread 2	{(1,2),(19,20)}
Thread 3	{(7,8),(11,12),(13,14)}

2.3. dynamic,3:

Thread id	Valores executados por cada thread.
Thread 0	{(7,8,9),(19,20)}
Thread 1	{(1,2,3)}

Thread 2	{(4,5,6),(13,14,15)}
Thread 3	{(10,11,12),(16,17,18)}