Aula de Laboratório 05

Arquitetura de Computadores (2019/02)

VISÃO GERAL

Objetivo: Realizar um trabalho prático de programação paralela utilizando OpenMP. Na aula prática conceitos de criação de threads de execução, distribuição de carga de trabalho são apresentados.

ATIVIDADES

1. Hello World com OpenMP
2. Distribuição da carga de trabalho - for
3. Distribuição da carga de trabalho - sections
4. Valor máximo
5. Multiplicação de Matrizes
6. Algoritmo de Ordenação - Quicksort

MATERIAIS NECESSÁRIOS

7. Ambiente de Programação Linux
8. Ferramentas: C/C++ Compile (GCC)
9. Make (Opcional)
10. VSCode (opcional)

*Arquivos iniciais podem ser diretamente obtidos pelo seguinte link:

	https://drive.google.com/file/d/1nm1zSRBdGQIzxnCFx1-2MCRVjBb5rrh4/view?usp=sharing

ATIVIDADES

Atividade 01 - Hello World com OpenMP

Passo 1 - Hello World sem OpenMP:

```
// File: Atividade01/main.cpp

#include <iostream>
int main(){

   printf("Hello World!\n");

   return 0;
}
```

Passo 2 - Compilando o código

```
$ cd [Lab_05]_Nome_Aluno/Atividade01
$ make run

rm bin/*
g++ -Wall -Wextra -std=c++17 -ggdb -fopenmp -Iinclude -Llib
src/main.cpp -o bin/main
./bin/main
Hello World!
```

Passo 3 - Paralelizando o código

```
#include <iostream>
int main() {

#pragma omp parallel
   {
      printf("Hello World!\n");
   }
   return 0;
}
```

Passo 4 - Compilando o código

```
$ make run
rm bin/*
g++ -Wall -Wextra -std=c++17 -ggdb -fopenmp -Iinclude -Llib
src/main.cpp -o bin/main
```

```
./bin/main
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
```

Passo 5 - Hello Word 2.0

```
#include <iostream>
#include <omp.h>
int main()
{
   omp_set_num_threads(4);
   int np = 0;
   int iam = 0;
    #pragma omp parallel private(np, iam)
   {
      np = omp_get_num_threads();
      iam = omp_get_thread_num();
      printf("Hello World from thread %d of %d!\n", iam, np);
   }
   return 0;
}
```

Atividade 02 - Distribuição da carga de trabalho - omp for

Passo 1

```
#include <iostream>
#include <omp.h>

int main() {
   int i;
   int temp;
   temp=-1;

   for(i=0;i<=10;i++) {
```

```
        temp=i;
        printf("Thread id: %d | Valor de temp: %d\n", omp_get_thread_num(), temp);
    }
    printf("temp: %d\n", temp);

    return 0;
}
```

Passo 2:

```
#Para fazer na aula
Dica: utilizar a diretiva #pragma omp parallel for e verificar as variáveis ( private ou shared)
```

Passo 3 - Distribuição de carga com uma variável compartilhada

```
#include <iostream>
#include <omp.h>

int main() {
    int i;
    int temp;
    temp=-1;
    int res; // var compartilhada
    res = 0;

[...]

// Saída:
//res = 45
```

Atividade 03 - Distribuição da carga de trabalho 2 - Sections

Passo 1 - Codificando

```
#include <iostream>
#include <omp.h>

int main()
{
   // Todas as threads devem executar
   printf("Todas as threads executam esta instrução! Sou a Thread %d\n",
omp_get_thread_num());

   printf("Section A: Esta section vai ser executada somente pela thread %d !\n",
omp_get_thread_num());
   for(int i =0; i<100000; i++){
      double temp = 10.0 * 5.0;
   }

   printf("Section B: Esta section vai ser executada somente pela thread %d !\n",
omp_get_thread_num());
   for(int i =0; i<100000; i++){
      double temp = 10.0 * 5.0;
   }

   printf("Section C: Ultima secao vai ser executada pela thread %d !\n",
omp_get_thread_num());
   for(int i =0; i<100000; i++){
      double temp = 10.0 * 5.0;
   }

   printf("Por fim, todas as threads executam esta instrução ! Sou a Thread %d\n",
omp_get_thread_num());

}
```

Passo 2 - Paralelizando o código com *#pragma omp sections*

```
#Para fazer em aula
```

Passo 3 - Continuação adicionando uma variável compartilhada

```
#Para fazer em aula
```

Atividade 04 - Valor máximo

Inicializar 3 vetores de inteiros. Realizar a busca pelo maior valor presente nesses vetores. Deseja-se que a busca seja paralelizada com API do OpenMP. Saída do

programa deve exibir o valor encontrado. Dica: Utilizar o rand() para inicializar os vetores de tamanho N.

```
#DEFINE N 1000000
int main()
{
   int v1[N] = {5, ..., 6};
   int v2[N] = {3, ,..., 1};
   int v3[N] = {8,. .., 6};

//Usar rand() para gerar números aleatórios
```

Saída:

```
// Exemplo de saída
rm bin/*
g++ -Wall -Wextra -std=c++17 -ggdb -fopenmp -Iinclude -Llib
src/main.cpp -o bin/main
./bin/main
Valor máximo: 2147483578
```

Atividade 05 - Matrizes: Multiplicação de matrizes

Inicializar duas matrizes de tamanho NxM (Matriz A e Matriz B) de float. Inicializar uma matriz de resultado. Executar o algoritmo de multiplicação de matrizes paralelizado com a API do OpenMP. Saída do programa deve printar na tela a matriz de resultado e a quantidade total de multiplicações que foram realizadas na operação.

```
#define N 100
#define M 100
// Sugestão de função de impressão
void imprime(int linhas, int colunas, float **matriz) {
   int l, c;
   for (l = 0; l < linhas; l++) {
      for (c = 0; c < colunas; c++)
         printf("%.2f ", matriz[l][c]);
      printf("\n");
   }
}
```

Atividade 06
Algoritmos de Ordenação - Divisão e Conquista - Quicksort

Implementar o algoritmo de ordenação Quicksort paralelizado com a API do OpenMP. Deseja-se ordenar um vetor de inteiros de tamanho N. Saída do programa: Print do vetor ordenado.

Ponto extra para os que fizerem a avaliação de tempo de processamento utilizando 1, 2, 3, 4, ... threads.

Sugestão: utilizar as funções clock() e omp_get_wtime()

Algoritmo Paralelizado:

```
#DEFINE N 100000
int main(void)
{
   int a[N] = {5, ..., 1, 7, 6};
```

ATIVIDADES PARA ENTREGAR

Para a entrega da atividade, os seguintes artefatos são desejados:

- ❏ *Código fonte para as atividade 1, 2 e 3*
- ❏ *Código-fonte para a atividade 4, 5, 6*

Recomendações:

- ✓ *Enviar os arquivos compactados em formato zip e com seguinte formato de nome: [Lab_05]_Nome_Aluno.zip*