Sistemas Paralelos e Distribuídos

Práticas – Aula 4

OpenMP

- Intro
- Conceitos
- Exemplos e exercícios
- Execução em laboratório

Intro

OpenMP

- Iniciativa do Parallel Computing Forum, culminou com 1ª versão em 1997
- API de programação multi-thread: diretivas, funções e variáveis de ambiente
- Desenhado para realizar paralelismo com base em memória compartilhada
- Desenvolvimento colaborativo: Governo, indústria e academia
- Versão 6 é mais atual
- Fluxo baseado na execução de thread master e thread slaves
- Alto grau de abstração, com execução paralela gerida pelo compilador
- Alta portabilidade

Conceitos

- Utiliza diretivas *pragma*

- Específica número de threads através do ambiente

- Faz o fork-join de forma transparente no código

- Rever slides em: https://www3.nd.edu/~zxu2/acms60212-40212/Lec-12-OpenMP.pdf

Conceitos

- Requer *omp.h*

- Principais diretivas
  - omp parallel
  - omp master
  - omp single
  - omp for
  - omp critical
  - omp atomic

- Principal váriavel
  - OMP_NUM_THREADS

- Principais funções da biblioteca
  - omp_get_num_threads()
  - omp_get_thread_num()
  - omp_set_num_threads()

Conceitos

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

int main(int argc, char** argv){
    #pragma omp parallel
    {
        printf("Hello from process: %d\n", omp_get_thread_num());
    }
    return 0;
}
```

Conceitos

- Para se compilar o programa e executá-lo, utilizamos a a flag ***-fopenmp***

#> gcc hello.c -o hello -fopenmp

#> export OMP_NUM_THREADS=4 && ./hello

Conceitos

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

int main(int argc, char** argv){

    Int thread_num = 0;

    #pragma omp parallel
    {
        #if _OPENMP
          thread_num =  omp_get_thread_num();
        #endif

        printf("Hello from process: %d\n",thread_num );

    }
    return 0;
}
```

Conceitos

Sincronismo com “critical” e escopo com “shared”

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

int main()
{

int x;
x = 0;

#pragma omp parallel shared(x)
  {

  //#pragma omp critical
  x = x + 1;

  }
printf("A suma eh %d\n",x);
return 0;
}
```

Conceitos

Private

```
#include <stdlib.h>
#include <stdio.h>
#include "omp.h"
int main()

{
int nthreads, A[10] , tid;

omp_set_num_threads(4);
 #pragma omp parallel private (tid)
 {

  tid = omp_get_thread_num();

  if (tid == 0)
      A[9] = 999;

  printf("The tid is %d and the array A[9] is %d \n",tid, A[9]);
  }
}
```

Conceitos

Barrier

```
#include <stdlib.h>
#include <stdio.h>
#include "omp.h"
int main()

{
int nthreads, A[10] , tid;

omp_set_num_threads(4);
 #pragma omp parallel private (tid)
 {

  tid = omp_get_thread_num();

  if (tid == 0)
       A[9] = 999;
  #pragma omp barrier
  printf("The tid is %d and the array A[9] is %d \n",tid, A[9]);
  }

}
```

Exemplo

Calcular a raiz quadrada de 1milhão de números, dados de forma aleatória.

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>

void inicializa(int **v, int size)
{
        (*v) = (int *) malloc(sizeof(int)*size);
        for (int i=0; i < size; i++)
        {
                (*v)[i] = rand() %10000;
        }
}

float raiz(int x)
{
      int k = 0;
      while(k < 5000) k++;
      return sqrt(x);
}

int main(int argc, char **argv)
{
        srand(time(NULL));
        int *vetor;
        int size = 1000000;
        inicializa(&vetor, size);
        for (int i=0; i < size; i++)
        {
        vetor[i] = raiz(vetor[i]);
        }
return 0;
}
```

Exercício

Como codificar o exemplo anterior de forma paralela com openmp?
Utilizemos um número de threads no qual 1M seja divisível.

Exercício

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>
#include <omp.h>
void inicializa(int **v, int size)
{
        (*v) = (int *) malloc(sizeof(int)*size);
        for (int i=0; i < size; i++)
                (*v)[i] = rand() %10000;
}
 float raiz(int x)
{
       int k = 0;
       while(k < 5000) k++;
       return sqrt(x);
}

int main(int argc, char **argv)
{
  srand(time(NULL));
  int *vetor;
  int size = 1000000;
  inicializa(&vetor, size);
#pragma omp parallel
{
 int local_init, local_end, lote;
 lote = size/omp_get_num_threads();
 local_init = omp_get_thread_num() * lote;
 local_end = (omp_get_thread_num() + 1) * lote;

 for (int i =local_init; i < local_end; i++)
       vetor[i] = raiz(vetor[i]);
 }
return 0;
}
```

Exercício

Construir um programa que dispara quatro threads para realizar a soma de 1 até 100.
Se possível comparar com a mesma solução de um programa serial.

Exercício

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

void main()
{

int i, sum = 0;
int thread_sum[4];
omp_set_num_threads(4);

#pragma omp parallel
{
        int ID = omp_get_thread_num();
        thread_sum[ID]=0;
        #pragma omp for
        for(i = 1;i <= 100; i++)
        {
          thread_sum[ID] += i;
          int k=0;
          while (k < 50000000) k++;
        }
}
for (i=0;i<4;i++)
 sum += thread_sum[i];

printf("Sum = %d\n",sum);

}
```

Laboratório
Referências