Computação Paralela e Distribuída

Ano lectivo 2023-24

Rascunho - Será reorganizado e actualizado

Tema#03

Programação de Sistemas de Memória Partilhada

João José da Costa
joao.costa@isptec.co.ao

Coordenação de Engenharia Informática
Departamento de Engenharias e Tecnologias
Instituto Superior Politécnico de Tecnologias e Ciências

OpenMP

Conceitos Fundamentais

Objectivos

Instrutivo

- Escrever programa OpenMP (OMP) simples

Educativo

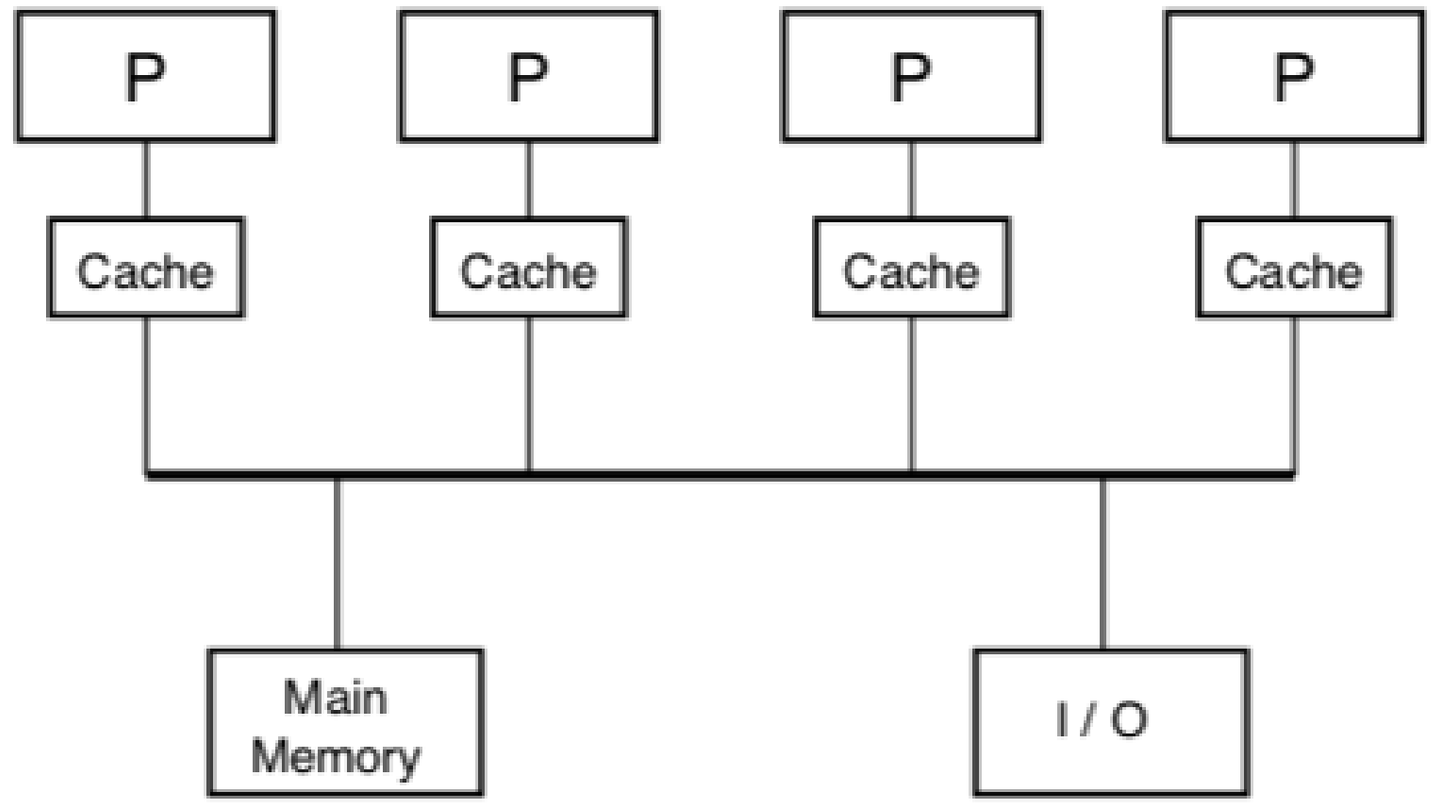
- Sentir a necessidade de aplicar os conceitos de OMP e de co-relacionar as PThreads discutido em sistemas operativos com OMP para escrever soluções eficientes.

Tópicos

- Programação concorrente de memória partilhada
- Revisão de Sistemas Operativos: Pthreads
- OpenMP
  - Cláusula Parallel
  - Variáveis privada/partilhadas

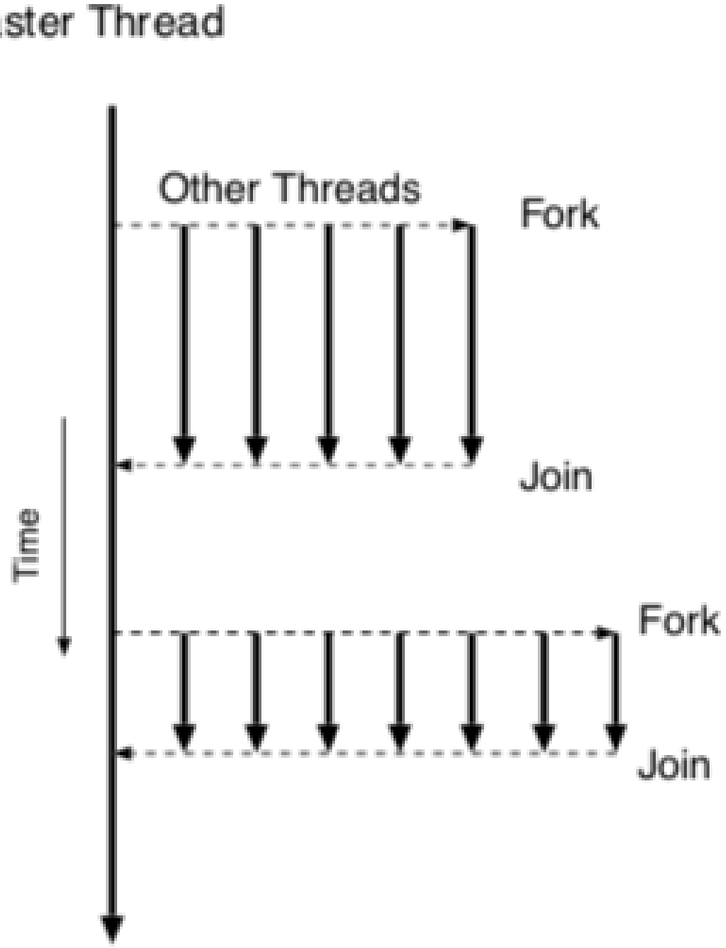
Memória Partilhada

- Arquitectura UMA (Uniform Memory Access) ou SMP (Symmetric Shared-Memory Multiprocessors).



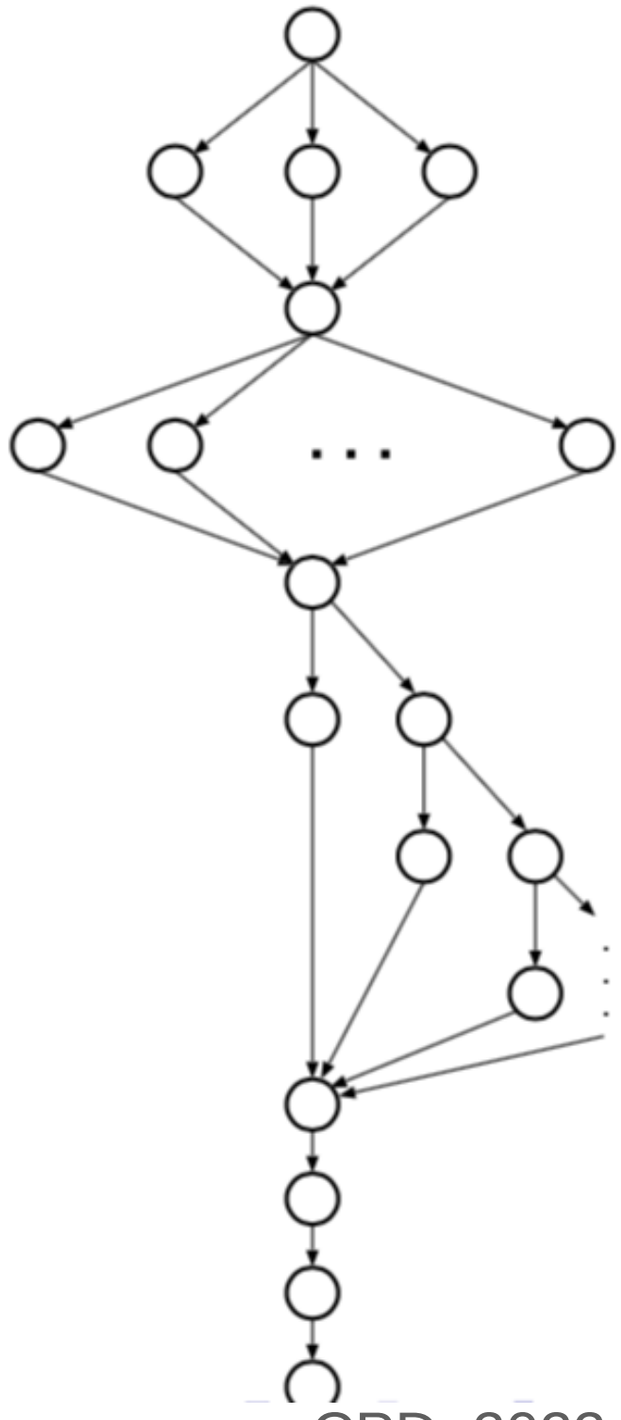
Paralelismo fork/join

- Criação/término “barato” de tarefas convida para paralelização incremental: processo de converter um programa sequencial para um programa paralelo em pouco tempo.
  - Inicialmente apenas a thread master está activa
  - Thread master executa código sequencial
  - Fork: thread master cria ou desperta threads adicionais para executar código paralelo
  - Join: no final do código paralelo, as threads criadas morrem ou são suspensos

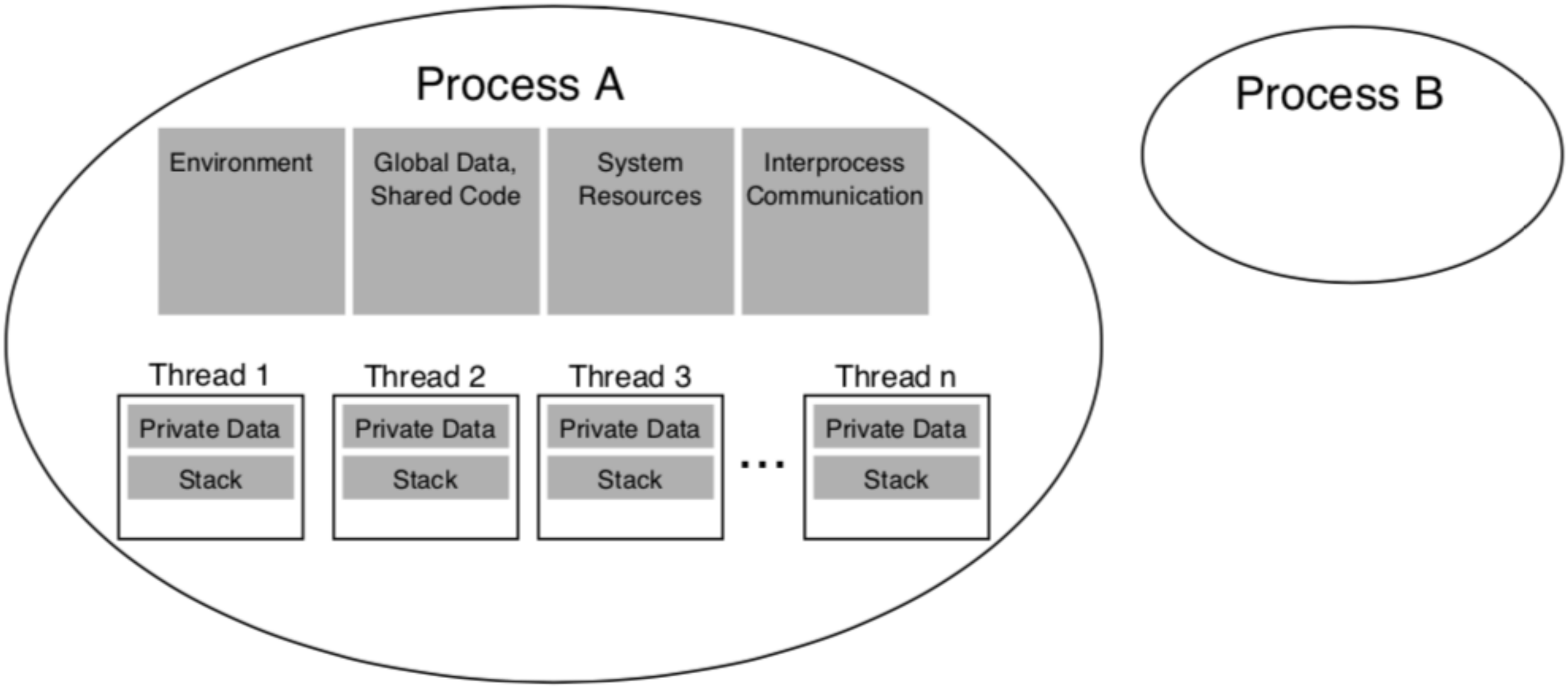


Paralelismo fork/join

```
read(A, B);

x = initX(A, B);
y = initY(A, B);
z = initZ(A, B);

for(i = 0; i < N_ENTRIES; i++)
    x[i] = compX(y[i], z[i]);

for(i = 1; i < N_ENTRIES; i++){
    x[i] = solveX(x[i-1]);
    z[i] = x[i] + y[i];
}

finalize1(&x, &y, &z);
finalize2(&x, &y, &z);
finalize3(&x, &y, &z);
```



Processos e Threads



POSIX Thread (PThread)

Criação de Thread

```
int pthread_create(pthread_t *thread, pthread_attr_t *attr,
                   void *(*start_routine)(void*), void *arg)
```

Exemplo:

```
pthread_t pt_worker;

void *thread_function(void *args) { /* thread code */ }

pthread_create(&pt_worker, NULL,
               thread_function, (void *) thread_args);
```

POSIX Thread (PThread)

Término e Sincronização

```
int pthread_exit(void *value_ptr)
```

```
int pthread_join(pthread_t thread, void **value_ptr)
```

POSIX Thread (PThread)

Sumar os valores em linhas da matriz

```c
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <pthread.h>

int buffer[N][SIZE];

void *sum_row (void *ptr){
  int index = 0, sum = 0;
  int *b = (int *) ptr;

  while (index < SIZE - 1)
    sum += b[index++]; /* sum row*/

  b[index]=sum; /* store sum
                   in last col. */

  pthread_exit(NULL);
}

int main(void){
  int i,j;
  pthread_t tid[N];

  for(i = 0; i < N; i++)
    for(j = 0; j < SIZE-1; j++)
      buffer[i][j] = rand()%10;
```

```c
for(i = 0; i < N; i++)
  if(pthread_create(&tid[i], NULL, sum_row,
                    (void *) &(buffer[i])) != 0){
    printf("Error creating thread, id=%d\n", i);
    exit(-1);
  }
  else
    printf ("Created thread w/ id %d\n", i);

for(i = 0; i < N; i++)
  pthread_join(tid[i], NULL);
printf("All threads have concluded\n");

for(i = 0; i < N; i++){
  for(j = 0; j < SIZE; j++)
    printf(" %d ", buffer[i][j]);
  printf ("Row %d \n", i);
}

exit(0);
}
```

POSIX Thread (PThread)

Sincronização

```
int pthread_mutex_init(pthread_mutex_t *mutex,
                       pthread_mutexattr_t *attr);
int pthread_mutex_lock(pthread_mutex_t *mutex);
int pthread_mutex_unlock(pthread_mutex_t *mutex);
```

Exemplo:

```
pthread_mutex_t count_lock;

pthread_mutex_init(&count_lock, NULL);

pthread_mutex_lock(&count_lock);
atomic_function();
pthread_mutex_unlock(&count_lock);
```

POSIX Thread (PThread)

Sincronização

```
int count;

void *sum_row(void *ptr){
  int index = 0, sum = 0;
  int *b = (int *) ptr;

  while(index < SIZE - 1)
    sum += b[index++]; /* sum row */

  b[index] = sum; /* store sum
                     in last col. */
  count++;

  pthread_exit(NULL);
}
```

Problema????

POSIX Thread (PThread)

Sincronização

```
int count;
pthread_mutex_t count_lock;

void *sum_row(void *ptr){
  int index = 0, sum = 0;
  int *b = (int *) ptr;

  while(index < SIZE - 1)
    sum += b[index++]; /* sum row */

  b[index]=sum; /* store sum
                   in last col. */

  pthread_mutex_lock(&count_lock);
  count++;
  pthread_mutex_unlock(&count_lock);

  pthread_exit(NULL);
}
```

```
main() { /*...*/

  pthread_mutex_init(&count_lock, NULL);

}
```

OpenMP

- Open specification for Multi-Threaded, Shared Memory Parallelism.
- Padrão de API (Application Programming Inerface)
  - Directivas de pré-processador
  - Chamada de biblioteca
  - Variáveis de ambiente
- Leia mais em www.openmp.org

OpenMP vs Threads

- (Supostamente) Melhor do que threads:
  - Modelo de programação mais simples
  - Separa um programa em regiões seriais e paralelas, em vez de T threads de execução simultânea.
- Similar a threads:
  - O programador deve detectar dependências
  - O programador deve evitar corridas de dados
- Leia mais em www.openmp.org

OpenMP
Receitas de Programação Paralela

- Threads
  1. Comece com um algoritmo paralelo
  2. Implemente, tendo em mente:
     - condição de corridas
     - Sincronização
     - Sintaxe de threading
  3. Teste e depuração
  4. Voltar ao passo 2

- OpenMP
  1. Comece com algum algoritmo
  2. Implemente serialmente, ignorando:
     - condição de corridas
     - Sincronização
     - Sintaxe de threading
  3. Teste e depuração
  4. Paralelizar automaticamente
     - com relativamente poucas anotações que especificam paralelismo e sincronização

OpenMP Directiva

- Directivas de paralelização:
  - parallel region
  - parallel for
  - parallel sections
  - task

- Directivas do ambiente de dado:
  - shared, private, threadprivate, reduction, etc.

- Directivas de sincronização:
  - barrier, critical.

Formato C/C++

```
#pragma omp nome-da-directive [cláusula,…] \n
```

- Sensível a maiúscula e minúscula.
- Linhas de directivas longas podem ser continuadas em linhas sucessivas escapando o caractere de nova linha com um “\” no final da linha de directive.
- Aplique sempre na próxima instrução, que deve ser um bloco estruturado. Exemplos:

```
#pragma omp ...
statement

#pragma omp ...
{ statement1; statement2; statement3; }
```

Região Parallel

```
#pragma omp parallel [cláusulas]
```

- Cria N threads paralelas
- Todas executam o bloco subsequente
  - Todos esperam uns pelos outros ao final da execução do bloco
    - sincronização de barreira

Quantas Threads?

- O número de threads criadas é determinado por, em ordem de precedência:
  - Uso da função de biblioteca omp_set_num_threads()
  - Configuração da variável de ambiente OMP_NUM_THREADS
  - Padrão de implementação - geralmente o número de CPUs
- É possível consultar o número de CPUs:

```
int omp_get_num_procs (void)
```

Região Parallel

Exemplo

```
main() {

    printf("Serial Region 1\n");

    omp_set_num_threads(4);

#pragma omp parallel
    {
      printf("Parallel Region\n");
    }

    printf("Serial Region 2\n");
}
```

Qual é o output???

Região Parallel

API para contar threads e obter o id

```
#include <omp.h>

int omp_get_thread_num()

int omp_get_num_threads()

void omp_set_num_threads(int num)
```

Exemplo:

```
#pragma omp parallel
{
  if( !omp_get_thread_num() )
    master();
  else
    slave();
}
```

OpenMP

Directivas de Partilha de Trabalho

- Ocorre sempre dentro de uma região paralela
- Divide a execução da região de código incluída entre os membros da equipa
- Não cria novas threads
- Duas directivas principais são:
  - parallel for
  - parallel section

OpenMP
Directiva for

```
#pragma omp parallel
  #pragma omp for [clauses]
    for( ; ; ) { ... }
```

- Cada thread executa um subconjunto das iterações
- Todos as threads sincronizam no final do parallel for
- Restricções:
  - Nenhuma dependência de dados entre as iterações
  - A correcção do programa não deve depender de qual thread executa uma iteração específica

Paradigma do Paralelismo de Dados

OpenMP
Directiva for

```
#pragma omp parallel
  #pragma omp for
    for ( ; ; ) { ... }
```

- É equivalente a

```
#pragma omp parallel for
  for ( ; ; ) { ... }
```

OpenMP

Exemplo de PThread Revisitado

```
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <pthread.h>

int buffer[N][SIZE];

void *sum_row (void *ptr){
  int index = 0, sum = 0;
  int *b = (int *) ptr;

  while (index < SIZE - 1)
    sum += b[index++]; /* sum row*/

  b[index]=sum; /* store sum
                   in last col. */

  pthread_exit(NULL);
}

int main(void){
  int i,j;
  pthread_t tid[N];

  for(i = 0; i < N; i++)
    for(j = 0; j < SIZE-1; j++)
      buffer[i][j] = rand()%10;
```

```
  for(i = 0; i < N; i++)
    if(pthread_create(&tid[i], 0, sum_row,
                      (void *) &(buffer[i])) != 0){
      printf("Error creating thread, id=%d\n", i);
      exit(-1);
    }
    else
      printf ("Created thread w/ id %d\n", i);

  for(i = 0; i < N; i++)
    pthread_join(tid[i], NULL);
  printf("All threads have concluded\n");

  for(i = 0; i < N; i++){
    for(j = 0; j < SIZE; j++)
      printf(" %d ", buffer[i][j]);
    printf ("Row %d \n", i);
  }

  exit(0);
}
```

OpenMP

Exemplo de PThread Revisitado

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

int buffer[N][SIZE];

void *sum_row (void *ptr){
  int index = 0, sum = 0;
  int *b = (int *) ptr;

  while (index < SIZE - 1)
    sum += b[index++]; /* sum row*/

  b[index]=sum; /* store sum
                   in last col. */


}

int main(void){
  int i,j;


  for(i = 0; i < N; i++)
    for(j = 0; j < SIZE-1; j++)
      buffer[i][j] = rand()%10;
```

```
#pragma omp parallel for
  for(i = 0; i < N; i++)
    sum_row(buffer[i]);



  printf("All threads have concluded\n");

  for(i = 0; i < N; i++){
    for(j = 0; j < SIZE; j++)
      printf(" %d ", buffer[i][j]);
    printf ("Row %d \n", i);
  }

  exit(0);
}
```

OpenMP

Múltiplas Directivas de Partilha de Trabalho

Pode ocorrer dentro da mesma região paralela:

```
#pragma omp parallel
{
  #pragma omp for
    for( ; ; ) { ... }
  #pragma omp for
    for( ; ; ) { ... }
}
```

Barreira implícita no final de cada for.

OpenMP

Parallel Sections

Paralelismo funcional: vários blocos são executados em paralelo

```
#pragma omp parallel
{
  #pragma omp sections
  {
    #pragma omp section
    { a=...;
      b=...; }
    #pragma omp section /* <- delimiter! */
    { c=...;
      d=...; }
    #pragma omp section
    { e=...;
      f=...; }
    #pragma omp section
    { g=...;
      h=...; }
  } /*omp end sections*/
} /*omp end parallel*/
```

OpenMP

Parallel Sections

```
#pragma omp parallel
  #pragma omp sections
    { ... }
```

- É equivalente a

```
#pragma omp parallel sections
  { ... }
```

OpenMP
Modelo de Memória

- Programas concorrentes acessam dois tipos de dados
  - Dados partilhados, visíveis para todos as threads
  - Dados privados, visíveis para uma única thread (geralmente alocado em pilha)
- Threads:
  - Variáveis globais são partilhadas
  - As variáveis locais são privadas
- OpenMP:
  - Todas as variáveis são por padrão partilhadas.
  - Algumas exceções:
    - a variável de loop de um parallel for é privada
    - as variáveis stack (locais) nas sub-rotinas chamadas são privadas
  - Ao usar directivas de dados, algumas variáveis podem se tornar privadas ou receber outras características especiais.

OpenMP

Variáveis privadas

`#pragma omp parallel for private( lista)`

- Faz uma cópia para cada thread para cada variável na lista.
  - Sem armazenamento associado com o objecto original
  - Todas referências são para o objecto local
  - Os valores não são definidos na entrada e na saída
- Também aplica para outra região e directivas de partilha de trabalho.

OpenMP

Variáveis partilhadas

`#pragma omp parallel for shared( lista)`

- Similarmente, há uma directiva de partilha de dado
- É responsabilidade do programador assegurar que todas as threads propriamente acessem as variáveis partilhadas (discutiremos a sincronização mais tarde).

OpenMP
Variáveis - Exemplo

PThreads

```
// shared, globals
int n, *x, *y;

void loop() {
  int i; // private, stack

  for(i = 0; i < n; i++)
    x[i] += y[i];
}
```

OpenMP

```
#pragma omp parallel \
  shared(n,x,y) private(i)
  {
    #pragma omp for
      for(i = 0; i < n; i++)
        x[i] += y[i];
  }
```

OpenMP
Variáveis - Exemplo

PThreads

```
// shared, globals
int n, *x, *y;

void loop() {
  int i; // private, stack

  for(i = 0; i < n; i++)
    x[i] += y[i];
}
```

OpenMP

```
#pragma omp parallel for
{
  for(i = 0; i < n; i++)
    x[i] += y[i];
}
```

OpenMP

Cláusula private - Exemplo

```
for(i = 0; i < n; i++)
  for(j = 0; j < n; j++)
    a[i][j] = b[i][j] + c[i][j];
```

- Faça o loop externo paralelo, para reduzir o número de forks/joins.
- Dê a cada thread sua própria cópia privada da variável j.

```
#pragma omp parallel for private(j)
for(i = 0; i < n; i++)
  for(j = 0; j < n; j++)
    a[i][j] = b[i][j] + c[i][j];
```

OpenMP

Cláusula firstprivate/lastprivate

- Conforme mencionado, os valores das variáveis privadas são indefinidos na entrada e na saída.
- Uma variável privada dentro de uma região não tem associação de armazenamento com a mesma variável fora da região

- `firstprivate (lista)`
- As variáveis na lista são inicializadas com o valor que a variável original tinha antes de entrar na construção paralela

- `lastprivate (lista)`
- A thread que executa a última iteração ou secção sequencialmente actualiza o valor das variáveis na lista

OpenMP

Cláusula firstprivate/lastprivate

```
main()
{
    a = 1;
    #pragma omp parallel for private(i),firstprivate(a),\
        lastprivate(b)
    for (i = 0; i < n; i++) {
        ...
        b = a + i; /* a indefinido, a menos que declarado firstprivate */
        ...
    }
    a = b; /* b indefinido, a menos que declarado lastprivate */
}
```

OpenMP

Variáveis threadprivate

- As variáveis privadas são privadas em uma base de região paralela.

- As variáveis threadprivate são variáveis globais privadas durante a execução do programa.

- `#pragma omp threadprivate(x)`
- Os dados iniciais são indefinidos, a menos que o copyin seja usado

- `copyin (lista)`
- os dados do thread mestre são copiados para as cópias threadprivate

OpenMP

Variáveis threadprivate - Exemplo

```
int a, b, i, tid;
float x;
#pragma omp threadprivate(a,x)
main(){
    printf("Regiao paralela #1");
    #pragma omp parallel private(b,tid)
    {
       tid = omp_get_thread_num();
       b = a = tid; x = 1.1 * tid + 1.0;
       printf("Thread %d: a,b,x= %d %d %f\n",tid,a,b,x);
    } /* fim da regiao paralela */
    printf("Regiao paralela #2");
    #pragma omp parallel private(tid)
    {
       tid = omp_get_thread_num();
       printf("Thread %d: a,b,x = %d %d %f\n",tid,a,b,x);
    } /*fim da regiao paralela*/
}
```

Revisão

- Programação concorrente de memória partilhada

- Revisão de Sistemas Operativos: Pthreads

- OpenMP
  - Cláusula Parallel
  - Variáveis privada/partilhadas

Próxima aula

- Sincronismo
- Paralelismo condicional
- Cláusula de redução
- Opções de escalonamento
- Directiva Task
- Paralelismo aninhado

Bibliografia

- Consulte dentro da subpasta “references” no repositório da disciplina.