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

Aspectos adicionais sobre OpenMP

Objectivos

Instrutivo

- Escrever programa OpenMP (OMP) optimizado.

Educativo

- Sentir a necessidade de entender os mecanismos de sincronização explícita e balanceamento de carga para escrever soluções mais eficientes.

Tópicos

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

Sincronização de thread

```
#pragma omp parallel [cláusula]
{ ... }
```

Barreiras implícitas no final do `parallel` (e outras construções de controlo): as execuções continuam somente após a conclusão de todos os threads.

Pode ser substituído nas directivas `for` e `section` com a cláusula `nowait`:

```
#pragma omp for nowait
{ ... }
```

Sincronização de thread

Exemplo do uso do nowait

```
int fatorial(int numero)
{
    int fat = 1;
#pragma omp parallel
    {
        int fat_privada = 1;
#pragma omp for nowait
        for(int n = 2; n <= numero; ++n)
            fat_privada *= n;
    #pragma omp atomic
            fat *= fat_privada;
    }
    return fat;
}
```

1. Qual é a função da variável `fat_privada`?
2. Porquê que faz sentido o uso do `nowait`?
3. Porquê precisou-se definir secção atómica?

Sincronização de thread

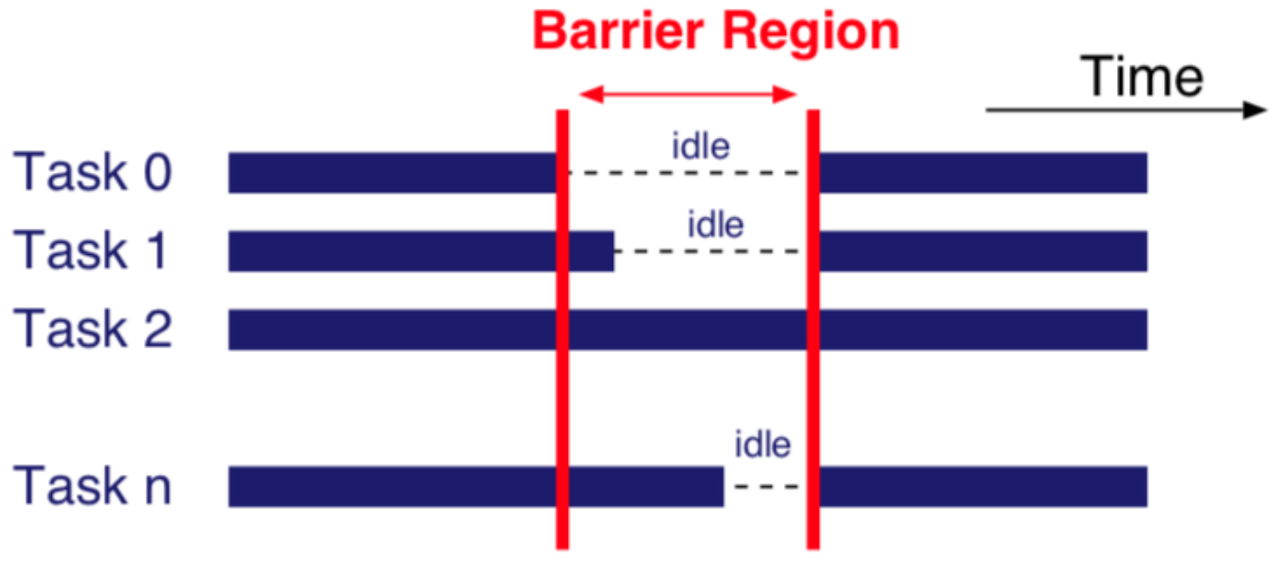
Sincronização explícita - Barreira

Uma barreira pode ser inserida explicitamente no código paralelo:

```
/* algum código multi-threaded */

#pragma omp barrier

/* restante do código multi-threaded */
```



Sincronização de thread

Sincronização explícita - Barreira

```
#pragma omp parallel
 {
   /* Todas as threads executam isto. */
   SomeCode();

   #pragma omp barrier
   /* Todas as threads executam isto, mas não
   * antes de todas concluirem a execução da
   * função SomeCode().
   */

   SomeMoreCode();
}
```

Sincronização de thread

Sincronização explícita

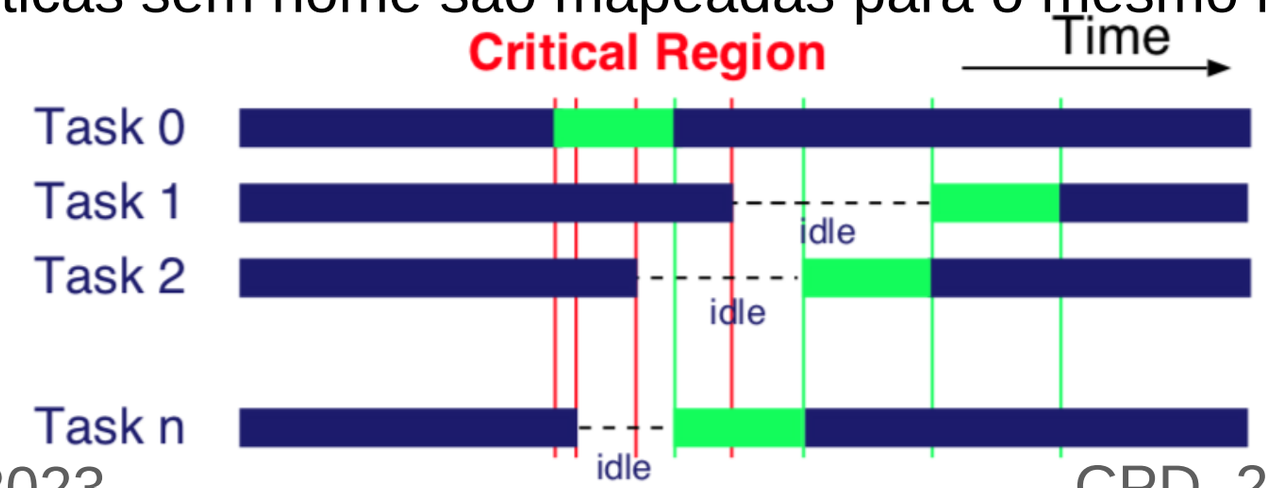
Secção crítica

Secção crítica, semelhante a mutexes em threads discutidas na unidade curricular de Sistemas Operativos.

```
#pragma omp critical [(nome)]
{ ... }
```

- uma thread espera no início de uma região crítica até que nenhuma outra thread esteja a executar uma região crítica com o mesmo nome;

- todas as directivas críticas sem nome são mapeadas para o mesmo nome não especificado.



Sincronização de thread

Sincronização explícita

Secção crítica

```
int cnt = 0;
#pragma omp parallel
{
  #pragma omp for
    for(i = 0; i < 20; i++) {
      if(b[i] == 0) {
        #pragma omp critical
          { cnt++; }
      }
      a[i] += b[i] * (i+1);
    }
}
```

Sincronização de thread

Sincronização explícita

- Uma seção crítica cria uma exclusão mútua em termos de execução de uma região do código.

- No entanto, o objectivo é a exclusão mútua do acesso aos dados.

```
#pragma omp atomic
{ ... }
```

- garante que a leitura e escrita de uma posição de memória é atómica

- aplica-se apenas à declaração imediatamente a seguir.

Sincronização de thread

Sincronização explícita

Secção atómica

```
int accum = 0;
#pragma omp parallel
{
   #pragma omp for
      for(i = 0; i < 20; i++) {
        if(b[i] == 0) {
          #pragma omp atomic
            accum += b[i] * (i+1);
        }
      }
}
```

Optmize o Código para melhor desempenho.

Sincronização de thread

Sincronização explícita

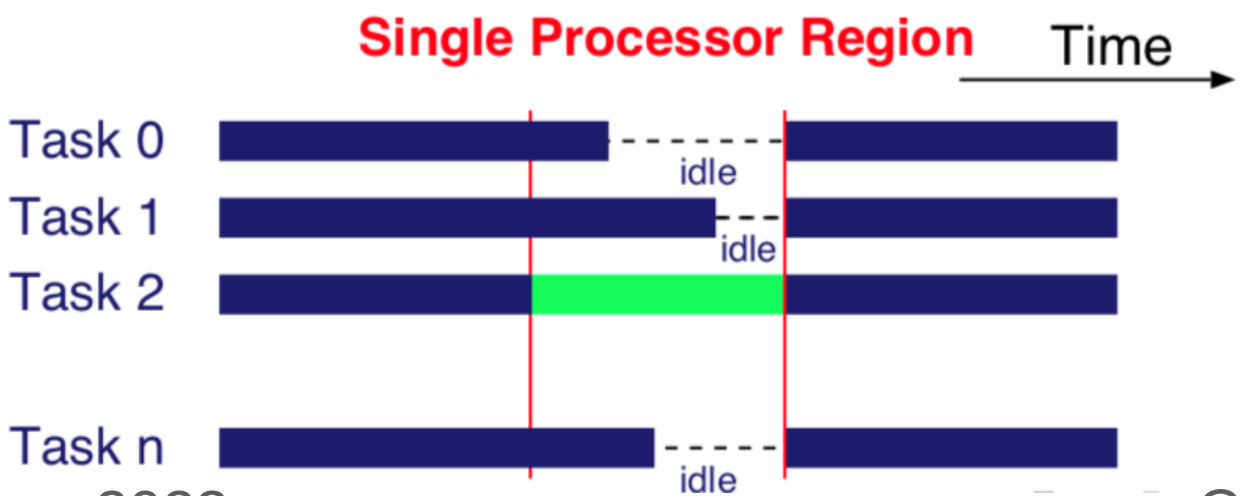
Região de processador único

Problema levemente diferente: Como fazer com que uma única thread execute uma região da secção paralela?

```
#pragma omp single
{  ...  }
```

- ideal para E/S ou inicialização
- qual thread executa a região não está definido

use master em vez de single para garantir que o thread mestre seja aquele que executa a região do processador único



Sincronização de thread

Sincronização explícita - Single

```
#pragma omp parallel
  {
    #pragma omp single
     printf(”A iniciar o work1.\n");
      work1();

    #pragma omp single
      printf(”A finalizar work1.\n");

    #pragma omp single nowait
      printf(”Finalizado work1 e a iniciar work2.\n");

     work2();
  }
```

Paralelismo condicional

- Muitas vezes, o paralelismo só é útil se o tamanho do problema for grande o suficiente.

- Para regiões com baixo esforço computacional, a sobrecarga de paralelização excede o benefício.

```
#pragma omp parallel if( expressão )
#pragma omp parallel sections if( expressão )
#pragma omp parallel for if( expressão )
```

Execute em paralelo se a expressão for avaliada como verdadeira, caso contrário, execute sequencialmente.

Paralelismo condicional

Exemplo

```
for(i = 0; i < n; i++)
  #pragma omp parallel for private (j,k) if(n-i > 100)
    for(j = i + 1; j < n; j++)
      for(k = i + 1; k < n; k++)
          a[j][k] = a[j][k] - a[i][k]*a[i][j] / a[j][j];
```

Cláusula de redução

Como paralelizar o cálculo de um produto interno?

```
#pragma omp parallel for reduction(op:lista)
```

- **op** é um operador binário (+,*,-,&,^ ,|,&&,||)
- **lista** é uma lista de variáveis partilhadas

Acções:

- uma cópia privada de cada variável de lista é criada para cada thread

- ao final da redução, o operador de redução é aplicado a todas as cópias privadas da variável e o resultado é escrito na variável global partilhada

Cláusula de redução

Exemplo

```
main() {
   int i, n = 100;
   float a[100], b[100], resultado = 0.0;

   #pragma omp parallel for
     for(i = 0; i < n; i++) {
         a[i] = i * 1.0;
         b[i] = i * 2.0;
     }
   #pragma omp parallel for reduction(+:result)
     for(i = 0; i < n; i++)
       resultado += (a[i] * b[i]);

   printf("Resultado final = %f\n", resultado);
}
```

Balanceamento de carga

Como paralelizar o cálculo de um produto interno?

```
#pragma omp parallel for reduction(op:lista)
```

- **op** é um operador binário (+,*,-,&,^ ,|,&&,||)
- **lista** é uma lista de variáveis partilhadas

Acções:

- uma cópia privada de cada variável de lista é criada para cada thread
- ao final da redução, o operador de redução é aplicado a todas as cópias privadas da variável e o resultado é escrito na variável global partilhada

Balanceamento de carga

- Com cargas de trabalho irregulares, deve-se ter cuidado ao distribuir o trabalho pelas threads.

- **Exemplo**: Multiplicação de duas matrizes C = A × B, onde a matriz A é triangular superior (todos os elementos abaixo da diagonal são 0).

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

Cláusula schedule

- Diferentes opções para distribuir o trabalho entre as threads.

```
schedule (static | dynamic | guided [,chunk])
schedule (auto | runtime)
```

- **static [, chunk]**
- As iterações são divididas em bloco de tamanho chunk e estes blocos são atribuídos as threads uma forma similar a round-robin.
- Na ausência do chunk, cada thread executa aproximadamente N/P chunks para um ciclo de tamanho N e P threads.

Exemplo, ciclo de tamanho N=8 e P=2 threads:

TID	0	1
No chunk	1-4	5-8
Chunk = 2	1-2, 5-6	3-4, 7-8

Cláusula schedule

- **dynamic [, chunk]**
- Um bloco de iterações de tamanho chunk é atribuído a cada thread (por defeito 1, se o chunk não for especificado);
- Quando uma thread termina, inicia no próximo bloco;
- Cada bloco contem chunk iterações, excepto para o ultimo bloco a ser distribuído, que pode ter menos iterações.

- **guided [, chunk]**
- O mesmo comportamento que o dynamic, mas as threads recebem diferentes tamanho de bloco;
- O tamanho de cada bloco é proporcional ao número de iterações não atribuídas divididas pelo número de threads, reduzindo para chunk.

Cláusula schedule

auto

- A decisão relativa ao escalonamento é delegada ao compilador e/ou sistema de tempo de execução.

runtime

- Esquema de escalonamento das iterações é configurado em tempo de execução através da variável de ambiente `OMP_SCHEDULE`

Opções

- **Escalonamento estático**
- Mais baixo overhead
- Pode lidar com o mais alto desbalanceada carga de trabalho

- **Chunk**
  - Chunks grande reduz o overhead e pode aumentar a taxa de acertos de cache;
  - Chunks pequeno permite o mais fino balanceamento de carga de trabalho.

Task

- A directiva task permite a definição de tarefas a serem executadas, que são adicionadas a um pool e eventually executadas por alguma thread no grupo.

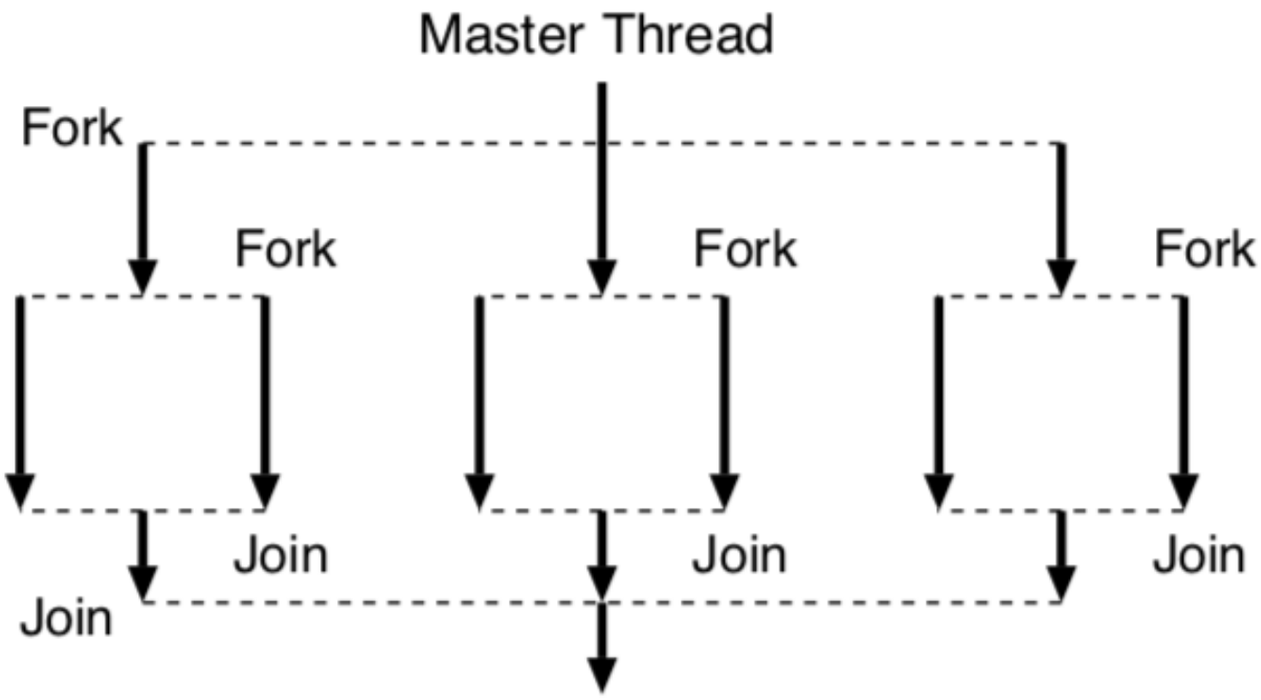
```
#pragma omp task [cláusula]
{<bloco estruturado>}
```

- Oferece um modelo flexível para paralelismo irregular.
- **Tarefas são garantidas a sua terminação em:**
- Barreiras de threads, tanto implicítas ou explícitas;
- Barreiras de terefa, isto é, #pragma omp taskwait

```
void f(no *p) {
    if (p->esq)
        #pragma omp task
          f(p->esq);
    if (p->dir)
        #pragma omp task
          f(p->dir);
  #pragma omp taskwait
    processa(p->dado);
}
```

Paralelismo encadeado

Regiões paralelas podem ser encadeadas (o suporte é dependente da implementação).



Precisa habilitar com a variável de ambiente `OMP_NESTED` ou com a rotina `omp_set_nested()`.

- Se a directive parallel é encontrado dentro de outra directive parallel, novo grupo de threads é criado.
- Novo grupo contem apenas uma thread a menos que o paralelismo encadeado está habilitado.

Paralelismo encadeado

Configura o número de threads por nível:

- Variável de ambiente: `OMP_NUM_THREADS` (p.e., 4,3,2)
- Rotina runtime: `omp_set_num_threads()` dentro da região paralela
- Cláusula: adiciona a cláusula `num_threads()` para uma directive paralela.

Set/get o número máximo de threads OpenMP disponíveis para o programa:

- Variável de ambiente: `OMP_THREAD_LIMIT`
- Rotinas de tempo de execução: `omp_get_thread_limit()`

Paralelismo encadeado

Set/get o número máximo de regiões paralelas encadeadas activas:

- Variável de ambiente: `OMP_MAX_ACTIVE_LEVELS`
- Rotinas de tempo de execução:
  `omp_set_max_active_levels()`,
  `omp_get_max_active_levels()`

Rotinas de biblioteca para determinar:

- Profundidade de encadeamento: `omp_get_level()`,
  `omp_get_active_level()`
- IDs da thread pai/avós/etc:
  `omp_get_ancestor_thread_num(level)`
- Tamanho do grupo de grupos de pai/avós/etc:
  `omp_get_team_size(level)`

Revisão

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

Bibliografia

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