Processamento Paralelo e Distribuído

AULA 6

Multithreaded Programming with OpenMP – Aula 3

Professor: Luiz Augusto Laranjeira
luiz.laranjeira@gmail.com

Material originalmente produzido pelo Prof. Jairo Panetta e adaptado para a FGA pelo Prof. Laranjeira.

Introdução à Programação Paralela no Modelo Fork-Join utilizando o Padrão OpenMP 2.0

Elementos de OpenMP

1. Diretivas
   - Sintaxe
   - Região Paralela
   - Cooperação de Trabalho
   - Combinação de Região Paralela e Cooperação de Trabalho
   - Sincronização
   - Ambiente
2. Funções
   - Ambiente
   - Sincronismo
   - Tempo
3. Variáveis de Ambiente
   - Ambiente

Modelo de Execução

- OpenMP implementa o modelo fork-join de execução paralela:

1. Um programa OpenMP executa sequencialmente (uma única thread denominada ”master thread”) até encontrar uma região paralela
2. Ao entrar na região paralela a thread mestre cria um time de threads e torna-se o mestre do time
3. Os comandos na região paralela são executados por todas as threads no time
4. Ao término da região paralela, todas as threads do time sincronizam (barreira implícita) e apenas a thread mestre continua a execução; as demais threads são destruídas
5. Qualquer número de regiões paralelas pode ser criado

Elementos de OpenMP

- Organização das Diretivas que criam e dividem paralelismo:
  - ...
  - Região Paralela
  - Cooperação de Trabalho
  - Combinação de Região Paralela e Cooperação de Trabalho
  - ...
- Região Paralela:
  - Cria paralelismo (abre e fecha)
  - Todas as threads executam o mesmo código
  - Não reduz o tempo de execução
- Cooperação de trabalho:
  - Divide a execução do código entre as threads
  - Reduz o tempo de execução
  - Só é permitida dentro de uma região paralela
- Combinação de Região Paralela e Cooperação de Trabalho:
  - Abre paralelismo e divide o trabalho em uma única diretiva

Elementos de OpenMP

1. Diretivas
   - Sintaxe
   - Região Paralela
   - Cooperação de Trabalho
   - Combinação de Região Paralela e Cooperação de Trabalho
   - Sincronização
   - Ambiente
2. Funções
   - Ambiente
   - Sincronismo
   - Tempo
3. Variáveis de Ambiente
   - Ambiente

Região Paralela

- Uma região paralela é definida por

  #pragma omp parallel <lista-de-clausulas>

  <bloco-estruturado>

  onde:

1. <bloco-estruturado> é um bloco estruturado de comandos da linguagem base – são proibidos saltos (“jumps”) de/para o bloco

2. <lista-de-clausulas> pode ser vazia ou conter:
   - Cláusulas de Paralelismo (não veremos)
   - Cláusulas de Escopo (private, shared, firstprivate)
   - Cláusulas de Redução (reduction)

Elementos de OpenMP

1. Diretivas
   - Sintaxe
   - Região Paralela
   - Cooperação de Trabalho
   - Combinação de Região Paralela e Cooperação de Trabalho
   - Sincronização
   - Ambiente
2. Funções
   - Ambiente
   - Sincronismo
   - Tempo
3. Variáveis de Ambiente
   - Ambiente

Cooperação de Trabalho

- A região paralela apenas cria o time de tarefas (threads); não divide o trabalho entre as tarefas
  - Cada thread executa todo código na região paralela
  - A divisão de trabalho pode ser implementada estabelecendo trabalhos distintos para threads distintas, utilizando thread ID
    - Trabalhoso, fácil de errar, dependente do número de threads

- Comandos de cooperação de trabalho dividem o trabalho pelas threads

- Os comandos de cooperação de trabalho são:
  - sections – divide explicitamente bloco de comandos
  - single – atribui bloco de comandos a uma thread
  - for – divide laços com controle explícito de índices

Cooperação de Trabalho - sections

- A execução de uma sequencia de <blocos> é dividida entre as threads de forma que cada <bloco> é executado por uma única thread

- Todas as threads sincronizam (barreira) ao fim do comando

```
#pragma omp sections <lista-inicial-de-clausulas>
{
#pragma omp section
<structured-block>
#pragma omp section
<structured-block>
...
}
```

Exemplo

```
#pragma omp parallel
{
  #pragma omp sections
  {
    #pragma omp section
      work1();
    #pragma omp section
      work2();
  }
}
```

- O número máximo de threads que realizam trabalho útil é definido pelo texto do programa

Cooperação de Trabalho – single

- Uma única thread executa o <bloco-de-comandos>
- Todas as threads sincronizam (barreira) ao final do comando

```
#pragma omp single <lista-inicial-de-cláusulas>
<structured-block>
```

Exemplo

```
#pragma omp single
work1();
```

- Qual é a utilidade deste comando?
  - Execução sequencial no interior de uma região paralela evita quebrar a região paralela em duas

Cooperação de Trabalho - for

- A diretiva OpenMP para um laço é definida por

  #pragma omp for <lista-de-clausulas>

  <laço for>

  onde lista de cláusulas pode conter:

  - Cláusulas de Escopo
  - Cláusula de Redução (já vista em região paralela)
  - Cláusula de Divisão de Tarefas

for

- As iterações do <laço-for> são divididas entre as threads

  #pragma omp for <lista-de-clausulas>

  <laço-for>

- O <laço-for> obrigatoriamente tem contador explícito de iterações
  - for (i=0, i<n, i++)
- O compilador gera código que divide as iterações do laço pelas threads. Por exemplo, uma possível forma é
  - for (i=omp_get_thread_num(), i<n, i+=omp_get_num_threads())
- O compilador automaticamente privatiza o índice do laço for
  - Mas não as outras variáveis (e índices) no interior do laço

Loop: Restrições

1. O corpo do laço pode ser um aninhamento de laços; apenas o laço externo do aninhamento é dividido entre as threads (em OpenMP2.0, modificada em OpenMP3.0)

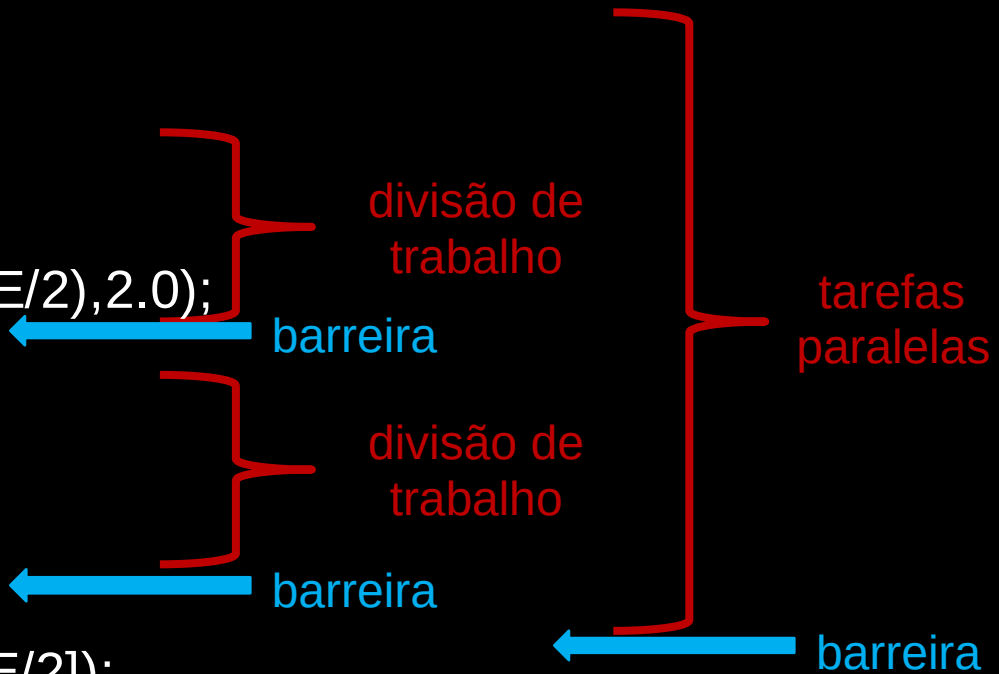
2. São proibidos saltos para dentro e para fora do <laço-for>
   - Exclui laços com break (por que?)

3. Há uma barreira implícita ao final do laço

4. Cláusulas de escopo:
   - PRIVATE, SHARED, FIRSTPRIVATE, LASTPRIVATE

Exemplo

```
#define VSIZE 1024
  int i;
  float vec[VSIZE];
#pragma omp parallel
  {
#pragma omp for private(i)
  for (i=0; i<VSIZE; i++)
       vec[i] = pow((float)(i-VSIZE/2),2.0);
#pragma omp for private(i)
  for (i=0; i<VSIZE; i++)
       vec[i] = sqrt(vec[i]);
  }
  printf(" meio %f\n", vec[VSIZE/2]);
```



Cooperação de Trabalho

- Comandos de cooperação de trabalho podem estar na extensão dinâmica de uma região paralela ou não
  - Execução paralela quando na extensão dinâmica
  - Execução sequencial (uma única thread) quando não
- Comandos de cooperação de trabalho são encontrados por todas as threads
  - Comandos consecutivos são encontrados na mesma ordem por todas as threads
- É proibido aninhar comandos de cooperação de trabalho
- Há uma barreira implícita ao final de cada comando de cooperação de trabalho
  - Todas as threads sincronizam no comando terminal
  - A barreira é eliminada pela inserção da cláusula nowait no comando

Cooperação de Trabalho - Loop

- A diretiva OpenMP para um laço é definida por

  #pragma omp for <lista-de-clausulas>

  <laço for>

  onde lista de cláusulas pode conter:

  - Cláusulas de Escopo
  - Cláusula de Redução (já vista em região paralela)
  - Cláusula de Decomposição de Tarefas

Cláusulas de Escopo de Loop

- private, shared, firstprivate: já vistas
- lastprivate (<lista-de-variáveis>)
  - As variáveis são privatizadas
  - O valor da variável privada na thread que executa a última iteração do laço é copiada para a variável original na barreira ao término da construção

```
#pragma omp parallel
{
#pragma omp for lastprivate(i)
    for (i=0; i<n-1; i++)
        a[i] = b[i] + b[i+1];
}
```

Exemplo de uso da diretiva lastprivate

  - Com a utilização de lastprivate, o valor da variável privada *i* ao fim da região paralela será: $i = n-1$

Lastprivate – Outro Exemplo

```
int x = 44;
#pragma omp parallel      // 4 threads
 {
 #pragma omp for private(i)
     for (i=0; i<8; i++)  {
          x = i;
          printf("Tnum = %d, x = %d\n",
             omp_get_thread_num(), x);
     }
 }
 printf("x = %d\n, x);
```

```
Tnum = 0, x = 0
Tnum = 0, x = 1
Tnum = 2, x = 4
Tnum = 2, x = 5
Tnum = 3, x = 6
Tnum = 3, x = 7
Tnum = 1, x = 2
Tnum = 1, x = 3
x = 44
```

```
int x = 44;
#pragma omp parallel      // 4 threads
 {
 #pragma omp for lastprivate(i)
     for (i=0; i<8; i++)  {
          x = i;
          printf("Tnum = %d, x = %d\n",
             omp_get_thread_num(), x);
     }
 }
 printf("x = %d\n, x);
```

```
Tnum = 0, x = 0
Tnum = 0, x = 1
Tnum = 2, x = 4
Tnum = 2, x = 5
Tnum = 3, x = 6
Tnum = 3, x = 7
Tnum = 1, x = 2
Tnum = 1, x = 3
x = 8
```

Cooperação de Trabalho - Loop

- A diretiva OpenMP para um laço é definida por

  #pragma omp for <lista-de-clausulas>
  <laço for>

  onde lista de cláusulas pode conter:

  – Cláusulas de Escopo
  – Cláusula de Redução (já vista em região paralela)
  – Cláusula de Decomposição de Tarefas

Cláusula de Decomposição de Tarefas

- schedule (type, [chunk])
- divide o espaço de iterações do laço em trechos de tamanho chunk e atribui trechos a threads da forma determinada por type

- Onde
  - chunk, se presente, é uma expressão inteira que define o número de iterações atribuído a cada thread
  - type é um de:
    - static
    - dynamic
    - guided (Não veremos)
    - runtime (Não veremos)

- A cláusula schedule atribui conjuntos de iterações do laço a tarefas (decomposição de tarefas, task decomposition). O tamanho do conjunto é definido por chunk

- Mas também atribui tarefas a threads (mapeamento de tarefas, task mapping). O mapeamento é definido por type

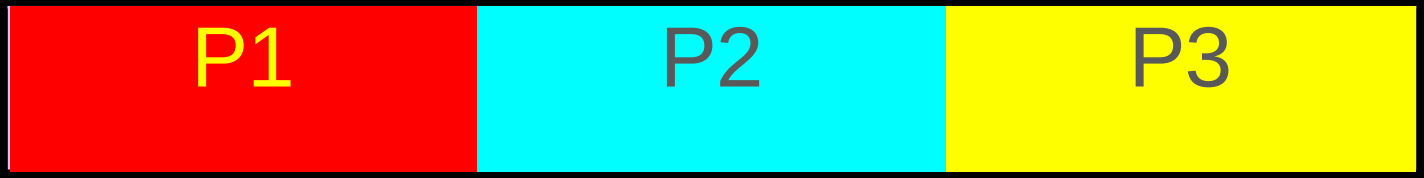
Mapeamento Estático

- schedule (static, [chunk])
  - Divide o espaço de iterações do laço em trechos de “chunk” iterações consecutivas
  - Atribui estaticamente trechos consecutivos à threads consecutivas, em “round-robin”
    - o primeiro trecho à thread 0, o segundo trecho à thread 1, e assim sucessivamente
  - Caso “chunk” não seja especificado, o espaço de índices do laço é dividido em trechos aproximadamente iguais e cada thread recebe um único trecho

Exemplo

Chunk Ausente

12 iterações, três threads



Distribuição por blocos

Block Distribution

Exemplo

Chunk Unitário

12 iterações, três threads

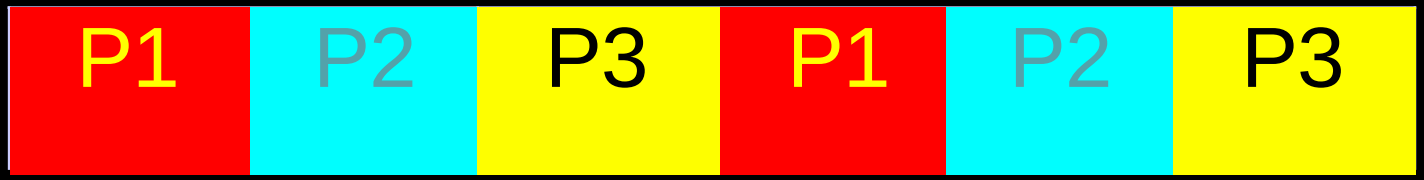
Distribuição Cíclica

Cyclic Distribution

Exemplo

Chunk = 2

12 iterações, três threads



Distribuição Cíclica por Blocos

Block Cyclic Distribution

Cláusula schedule

- Ex: em uma região paralela com 2 threads, o laço

```
#pragma omp for schedule (static, k)
for (i=1, i<=10, i++)
```

tem mapeamento de tarefas:

- cíclico se k=1
- por blocos de tamanho 5 se k ausente
- por blocos de tamanho 2 divididos ciclicamente se k=2

schedule (dynamic)

- schedule (dynamic, [chunk])
  - Atribui blocos de “chunk” iterações consecutivas a threads dinamicamente
  - Isto é, assim que uma thread termina a execução de um “chunk” de iterações ela obtém o próximo “chunk” dinamicamente
    - A ausência de chunk é equivalente a chunk=1

- Qual é a utilidade dessa construção?

- A cláusula schedule (dynamic) é particularmente útil se o tempo de execução das iterações do laço variar dinamicamente com a iteração do laço (desbalanceamento dinâmico de carga)

Semântica de Loop

- O número de iterações do laço é computado antes de entrar na construção
- As iterações do laço são divididas entre as threads conforme especificado pela cláusula schedule. Na ausência da cláusula schedule, o default depende da implementação.
- Diretivas LOOP em laços distintos com o mesmo número de iterações e schedule na mesma região paralela podem distribuir iterações a threads em ordem distinta, exceto em schedule (static)
- Programas que assumem qual thread executa qual iteração não são aderentes ao padrão, exceto no caso schedule (static)

Elementos de OpenMP

1. Diretivas
   - Sintaxe
   - Região Paralela
   - Cooperação de Trabalho
   - Combinação de Região Paralela e Cooperação de Trabalho
   - Sincronização
   - Ambiente
2. Funções
   - Ambiente
   - Sincronismo
   - Tempo
3. Variáveis de Ambiente
   - Ambiente

Combinação de Região Paralela e Cooperação de Trabalho

- Combina região paralela e cooperação de trabalho na mesma diretiva
  - Cria uma única região paralela com uma única diretiva de cooperação de trabalho
- Exemplo:
  - #pragma omp parallel for
- Semântica: Idêntico a usar a diretiva PARALLEL seguida pela diretiva de cooperação de trabalho
  - Custo: Overhead de criar e destruir as threads
- Benefício: O restante do código é serial
  - Não precisa abrir uma seção paralela extensa e lidar com paralelismo fora dos laços

Exemplo

```
#define VSIZE 1024
  int i;
  float vec[VSIZE];
#pragma omp parallel for private(i)
  for (i=0; i<VSIZE; i++)
    vec[i] = pow((float)(i-VSIZE/2),2.0);
#pragma omp parallel for private(i)
  for (i=0; i<VSIZE; i++)
    vec[i] = sqrt(vec[i]);
  printf(" meio %f\n", vec[VSIZE/2]);
```

- Abrir e fechar paralelismo é custoso
- É preferível uma única região paralela

Exemplo

```
#define VSIZE 1024
  int i;
  float vec[VSIZE];
#pragma omp parallel
  {
#pragma omp for private(i)
  for (i=0; i<VSIZE; i++)
       vec[i] = pow((float)(i-VSIZE/2),2.0);
#pragma omp for private(i)
  for (i=0; i<VSIZE; i++)
       vec[i] = sqrt(vec[i]);
  }
  printf(" meio %f\n", vec[VSIZE/2]);
```