Processamento Paralelo e Distribuído

AULA 5

Multithreaded Programming with OpenMP – Aula 2

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

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

Agenda

OpenMP

Região Paralela e Condição de Corrida

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
   - Cláusulas de Redução

Exemplo

```
void main()
{
      #pragma omp parallel
      {
            if (omp_get_thread_num() == 0)
                  printf("Sou thread %d diferente\n",
                        omp_get_thread_num());
            else
                  printf("Sou thread %d igual\n",
                        omp_get_thread_num());
      }
}
```

Execução com 8 threads

```
Sou thread 0 diferente
Sou thread 2 igual
Sou thread 3 igual
Sou thread 7 igual
Sou thread 5 igual
Sou thread 1 igual
Sou thread 6 igual
Sou thread 4 igual
```

Implementação Típica

O corpo da região paralela é encapsulado em um procedimento e delimitado por dois laços:

1° laço: percorre as threads, invocando pthread_create, passando o nome do procedimento como argumento;

- Equivalente a executar fork

EXECUÇÃO DO CORPO DA REGIÃO PARALELA

2° laço: percorre as threads, invocando pthread_join, que aguarda o término das threads.

- Equivalente a executar join

Condição de Corrida

- Na região paralela, todas as threads executam o mesmo código, simultaneamente, no mesmo espaço de endereçamento
- O que ocorre se múltiplas threads acessarem simultaneamente a mesma posição de memória, algumas lendo e outras escrevendo?
- O fenômeno é denominado *condição de corrida* (race condition)

Exemplo

- O que é impresso, supondo execução com 8 threads?

```
int m;
m = 0;
#pragma omp parallel
{
      m = m + 1;   (assembly [possível]:  read m;  inc ; write m)
      printf(" thread %d; m=%d\n", omp_get_thread_num(), m);
}
printf(" sequen  ; m=%d\n", m);
```

Três Execuções

thread 5; m=8	thread 7; m=7	thread 6; m=8
thread 1; m=8	thread 2; m=7	thread 2; m=8
thread 6; m=8	thread 5; m=7	thread 7; m=8
thread 4; m=8	thread 6; m=7	thread 1; m=8
thread 0; m=8	thread 3; m=7	thread 3; m=8
thread 3; m=8	thread 4; m=7	thread 4; m=8
thread 2; m=8	thread 1; m=7	thread 5; m=8
thread 7; m=8	thread 0; m=7	thread 0; m=8
sequen ; m=8	sequen ; m=7	sequen ; m=8

- Não determinístico por causa da condição de corrida

Condição de Corrida

- A variável *m* é armazenada em uma única posição de memória para todas as threads
- As threads competem por ler e escrever na posição de memória representada por *m*
  - conflito leitura-escrita
- O resultado depende da ordem de acesso à variável compartilhada (velocidade relativa das threads)
- Condição de corrida (“race condition”) é o fenômeno que ocorre quando o resultado de uma computação muda com a ordem (velocidade) de execução das tarefas componentes

Como eliminar condição de corrida?

1. Condição de corrida ocorre por acessos simultâneos:
   - Acessos escrita – escrita ou leitura – escrita geram condição de corrida
   - Acessos leitura – leitura não geram condição de corrida

2. Se a posição de memória simultaneamente lida e escrita (ou simultaneamente escrita) for duplicável sem alterar o resultado da computação:
   - Replica-se a variável correspondente àquela posição de memória, criando-se uma variável (posição de memória distinta) para cada thread
   - Tal operação é denominada privatização

3. Caso contrário, mude o algoritmo (ou a codificação) eliminando a condição de corrida

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
   - Cláusulas de Redução

Cláusulas de Escopo

- Mapeia os identificadores na região paralela em posições de memória

- private (lista-de-variáveis)
  - Uma nova área de armazenagem é criada para cada variável na lista e para cada thread
  - O valor inicial da nova área de armazenagem é indefinido
  - Ao término da construção paralela a área de armazenagem é destruída e o valor final da variável original é indefinido

- shared (lista-de-variáveis)
  - Todas as threads utilizam a área de armazenagem pré-existente para cada variável na lista

- Por default, todas as variáveis em uma região paralela são shared

Exemplo

```
int i, aux, vec[10];
#pragma omp parallel private(aux, i)
for (i=0; i<10; i++)   {
      aux = (i+1)*(i+2);
      vec[i] = (aux-1)/(aux+1);
}
```

- Todas as threads executam todas as iterações do laço
- Condição de corrida em *i*, *aux* e *vec* [*i*]
  - Conflito leitura-escrita em *i* e *aux,* eliminado por private
  - Conflito escrita-escrita em *vec* [*i*]

Implementação Típica de Private

- Variáveis privadas são variáveis locais ao procedimento invocado por pthreads_create
  - portanto seu endereço de memória muda de thread a thread;
- Variáveis globais são argumentos do procedimento invocado por pthreads_create
  - portanto mesmo endereço de memória para todas as threads.
- Suponha invocação de um procedimento no interior da região paralela. Aonde são armazenadas as variáveis locais ao procedimento e as variáveis globais utilizadas no procedimento?
  - Locais armazenadas na pilha de cada thread durante a execução da região paralela; logo, privadas e duplicadas;
  - Globais na área anteriormente alocada (não duplicadas)

Cláusulas de Escopo

- Já vimos private e shared

- firstprivate (lista-de-variáveis)
  - Idêntico a PRIVATE, ampliado por:
  - O valor inicial da variável original é copiado para a área de armazenagem recém criada

Exemplo

- O que é impresso, supondo 8 threads?

```
int main()   {
    int m;
    m = 0;
    #pragma omp parallel firstprivate(m)
    {
        m = m + 1;
        printf(" thread %d; m=%d\n",
            omp_get_thread_num(), m);
    }
    printf(" sequen          ; m=%d\n", m);
    exit(0);
}
```

Três Execuções

thread 5; m=1	thread 6; m=1	thread 5; m=1
thread 0; m=1	thread 1; m=1	thread 0; m=1
thread 7; m=1	thread 5; m=1	thread 6; m=1
thread 1; m=1	thread 3; m=1	thread 4; m=1
thread 2; m=1	thread 2; m=1	thread 2; m=1
thread 3; m=1	thread 0; m=1	thread 3; m=1
thread 6; m=1	thread 4; m=1	thread 1; m=1
thread 4; m=1	thread 7; m=1	thread 7; m=1
sequen ; m=0	sequen ; m=0	sequen ; m=0

- Determinístico

Implementação Típica

- Cada variável declarada como firstprivate torna-se argumento de entrada do procedimento;
- O procedimento cria (declara) uma variável local que será utilizada no lugar da variável declarada como firstprivate;
- No início do procedimento, o valor do argumento é copiado para a variável local correspondente.

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
   - Cláusulas de Redução

Cláusula de Redução

- Semântica da cláusula reduction (*operator: list*)
  1. Uma cópia privada de cada variável na lista é criada e inicializada
     - a variável obrigatoriamente é shared
     - o valor inicial é fixo para cada operador (valor nulo do operador)
       - <val> <op> <valor_nulo> = <val>
  2. Cada thread executa na sua cópia privada da variável
  3. Ao término da execução da construção OMP, a variável compartilhada é atualizada com o valor do operador aplicado a todas as cópias privadas e ao valor anterior da variável compartilhadas

- Restrições:
  1. Poucos operadores (+, *, .AND., MAX, MIN, etc)
  2. O operador é considerado associativo
     - Reprodutibilidade binária é violada em variáveis ponto flutuante

Exemplo

- O que é impresso, supondo 8 threads?

```
int main()    {
    int m;
    m = 0;
    #pragma omp parallel reduction(+:m)
    {
        m = m + 1;
        printf(" thread %d; m=%d\n",
              omp_get_thread_num(), m);
    }
    printf(" sequen          ; m=%d\n", m);
    exit(0);
}
```

Três Execuções

thread 5; m=1	thread 6; m=1	thread 5; m=1
thread 0; m=1	thread 1; m=1	thread 0; m=1
thread 7; m=1	thread 5; m=1	thread 6; m=1
thread 1; m=1	thread 3; m=1	thread 4; m=1
thread 2; m=1	thread 2; m=1	thread 2; m=1
thread 3; m=1	thread 0; m=1	thread 3; m=1
thread 6; m=1	thread 4; m=1	thread 1; m=1
thread 4; m=1	thread 7; m=1	thread 7; m=1
sequen ; m=8	sequen ; m=8	sequen ; m=8

- Determinístico

Implementação Típica

- No início da região paralela cria-se uma instância de cada variável na cláusula REDUCTION para cada thread

- A instância é passada como argumento para pthreads_create. É inicializada dentro do procedimento invocado e atua como acumulador, interno ao procedimento, da operação definida por REDUCTION

- Os valores acumulados por cada instância (thread) são totalizados no final da região paralela, acrescidos do valor inicial da variável.

Aninhamento de Regiões Paralelas

- Se durante a execução de uma região paralela uma thread encontra outra região paralela, a thread cria um novo time de threads e torna-se o mestre desse time.

- Por default, a execução de regiões paralelas aninhadas é serializada, i.e., a região paralela é executada por uma única thread (em OpenMP2.0, alterado em OpenMP3.0)

- O default pode ser alterado pela variável de ambiente

  OMP_NESTED (deprecated)

  OMP_MAX_ACTIVE_LEVELS

  ou pela invocação da função

  omp_set_nested (deprecated)
  omp_set_max_active_levels

Sumário

- Região Paralela delimita paralelismo em OpenMP.
- Condição de corrida é problema critico, resolvido por privatização e outras cláusulas complementares (reduction, por exemplo) em muitos casos.
- A alternativa é mudar o algoritmo.