OpenMP

Adaptado do Material do
Calebe de Paula Bianchini
calebe.bianchini@mackenzie.br

HelloWorld

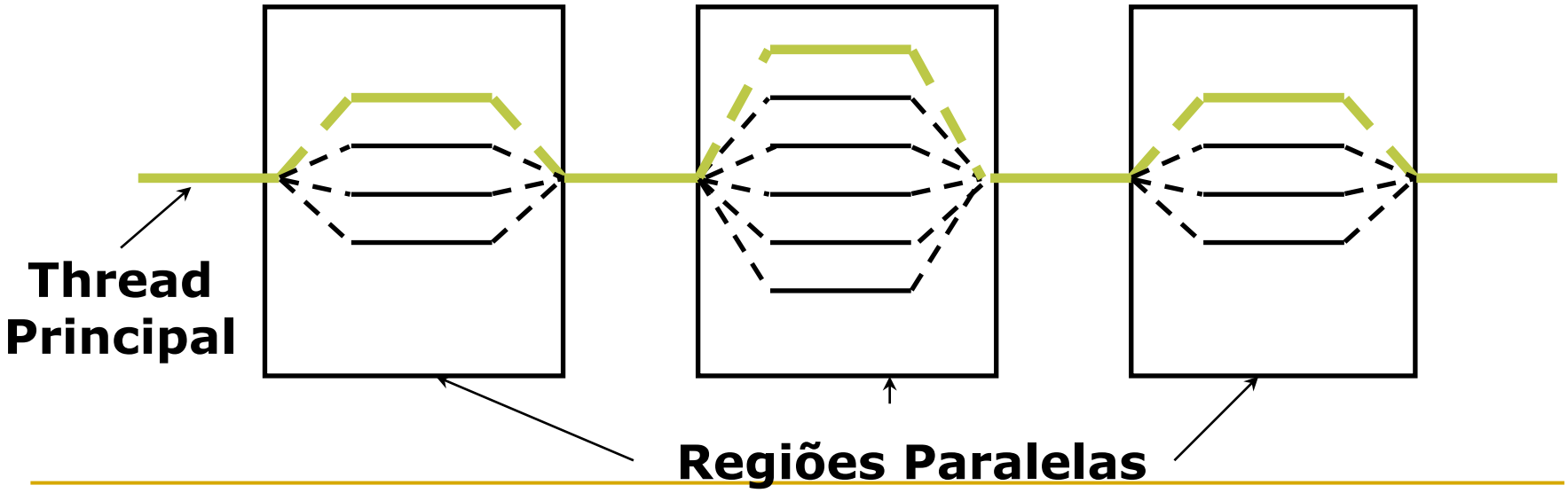
```
#pragma omp parallel
{
  // Code inside this region runs in parallel.
  printf("Hello!\n");
}
```

O que é OpenMP*?

- Diretivas de compilação para programação multithreading
- O código pode ser em C/C++ ou em Fortran
- Suporta um modelo de paralelismo de dados
- Paralelismo Incremental
  - Combina código serial e paralelo em um único código fonte

Modelo de Programação

- Paralelismo Fork-join:
  - A thread principal cria um time de threads conforme a necessidade
  - O paralelismo é adicionado incrementalmente: o programa seqüencial evolui para um programa paralelo



Sintaxe Pragma OpenMP

- Maioria das construções em OpenMP são diretivas de compilação ou pragmas.
  - Para C e C++, os pragmas são da seguinte forma:

#pragma omp construct [clause [clause]…]

Quantas Threads?

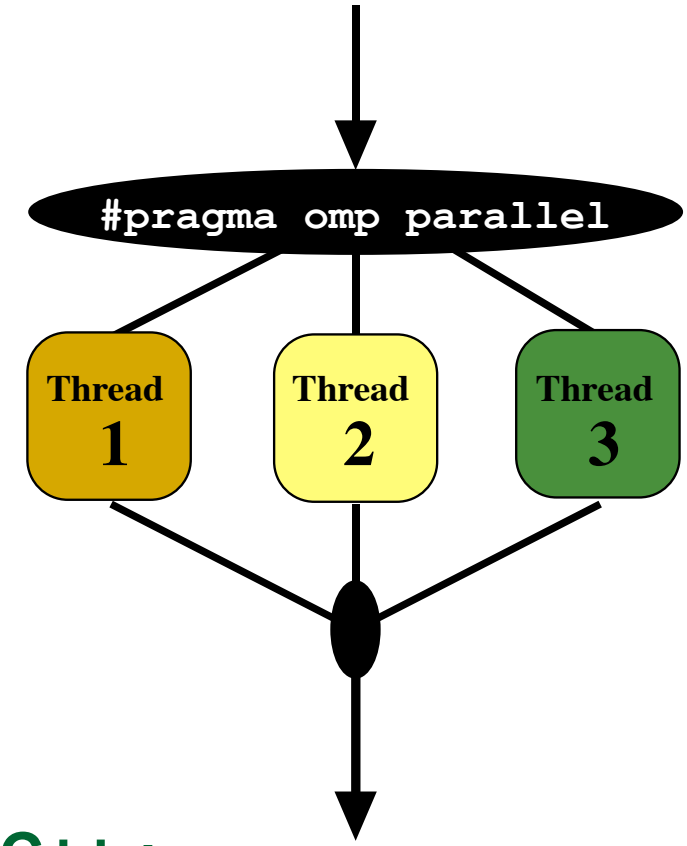
- Atualiza a variável de ambiente para o número de threads

```
set OMP_NUM_THREADS=4
```

- Não há um default padrão para esta variável
  - Maioria dos sistemas:
    - # de threads = # de processadores

Regiões Paralelas

- Define uma região paralela através de um bloco estruturado de código
- As threads são criadas em paralelo
- As threads ficam bloqueadas no final da execução
- Os dados globais são compartilhados entre as threads

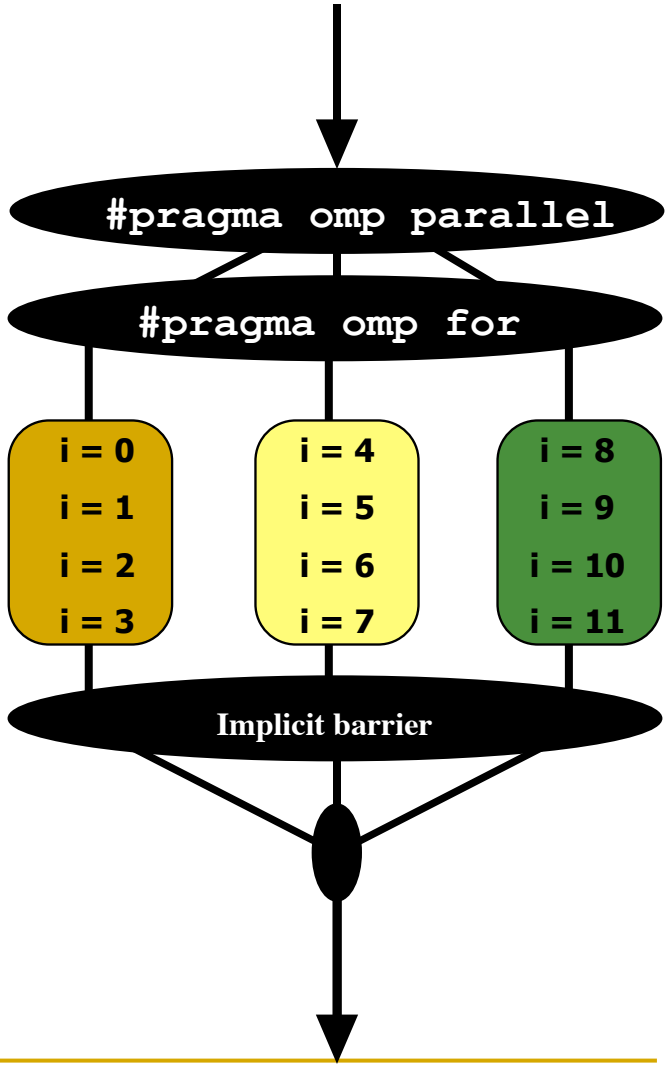


C/C++ :

```
#pragma omp parallel
    {
        block
    }
```

Construções de Work-Sharing

```
#pragma omp parallel
#pragma omp for
   for(i = 0; i < 12; i++)
      c[i] = a[i] + b[i]
```



Ambiente dos Dados

- OpenMP utiliza memória compartilhada como modelo de programação
  - Maioria das variáveis são compartilhadas por default
  - Variáveis globais são compartilhadas entre as threads
  - Variáveis que são index em loop são privadas

Combinando pragmas

- Estes dois segmentos de código são equivalentes

```
#pragma omp parallel
{
  #pragma omp for
  for (i=0; i< MAX; i++)
  {
       res[i] = huge();
  }
}
```

```
#pragma omp parallel for
    for (i=0; i< MAX; i++) {
         res[i] = huge();
    }
```

Escopo dos Atributos dos Dados

- O padrão é shared, mas pode ser modificado com:

```
default (shared | none)
```

- Escopo dos atributos

```
shared(varname,…)
```

```
private(varname,…)
```

Exemplos de Escopo Shared e Private

```
void* work(float* c, int N) {
  float x, y; int i;
 #pragma omp parallel for private(x,y)
     for(i=0; i<N; i++) {
        x = a[i]; y = b[i];
        c[i] = x + y;
     }
}
```

```
float dot_prod(float* a, float* b, int N)
{
  float sum = 0.0;
#pragma omp parallel for shared(sum)
   for(int i=0; i<N; i++) {
     sum += a[i] * b[i];
   }
  return sum;
}
```

Exemplo: Multiplicação

```
float dot_prod(float* a, float* b, int N)
{
  float sum = 0.0;
   #pragma omp parallel for shared(sum)
   for(int i=0; i<N; i++) {
     sum += a[i] * b[i];
   }
  return sum;
}
```

Oque está errado?

Exemplo: Laços Aninhados

```
#pragma omp parallel for
for(int y=0; y<25; ++y)
{
  #pragma omp parallel for
  for(int x=0; x<80; ++x)
  {
    tick(x,y);
  }
}
```

O que acontece ?

Proteção de Dados Compartilhados

- Deve ser protegido o acesso e a modificação a dados compartilhados

```
float dot_prod(float* a, float* b, int N)
{
  float sum = 0.0;
  #pragma omp parallel for shared(sum)
   for(int i=0; i<N; i++) {
     #pragma omp critical
      sum += a[i] * b[i];
   }
  return sum;
}
```

OpenMP: Construção Crítica

#pragma omp critical [(lock_name)]

- Define uma região crítica em um bloco estruturado

```
float R1, R2;
#pragma omp parallel
{ float A, B;
  #pragma omp for
  for(int i=0; i<niters; i++){
    B = big_job(i);
    #pragma omp critical a
          consum (B, &R1);
    A = bigger_job(i);
    #pragma omp critical b
          consum (A, &R2);
  }
}
```

OpenMP: Reduction

reduction (op : list)

- As variáveis em “list” devem ser shared na região paralela

- Dentro da construção paralela
  - Uma cópia PRIVATE de cada variável é criada e inicializada dependendo da operação
  - Essas cópias são atualizadas localmente pelas threads
  - Ao final da construção, as cópias locais são combinadas através da operação “op” em um único valor, e combinado com o valor original da variável SHARED.

Exemplo: Reduction

```
#pragma omp parallel for reduction(+:sum)
   for(i=0; i<N; i++) {
     sum += a[i] * b[i];
   }
```

- Uma cópia local de sum para cada thread
- Todas as cópias locais de sum são adicionadas e armazenadas na sua variável "global"

Distribuição das Iterações

- A cláusula schedule afeta como as iterações do loop são mapeadas nas threads

schedule(static [,chunk])

  - Blocos de iterações de tamanho “chunk” para as threads
  - Distribuição Round robin

schedule(dynamic[,chunk])

  - Threads recebem “chunk” iterações
  - Quando as iterações recebidas terminam, a thread requisita o próximo conjunto

schedule(guided[,chunk])

  - O escalonamento dinâmico inicia com blocos de tamanho grande
  - Com o passar do tempo, o tamanho do bloco diminui; mas sempre maior do que o “chunk”

Exemplo de Cláusula Schedule

```
#pragma omp parallel for schedule (static, 8)
    for( int i = start; i <= end; i += 2 )
    {
       if ( TestForPrime(i) )  gPrimesFound++;
    }
```

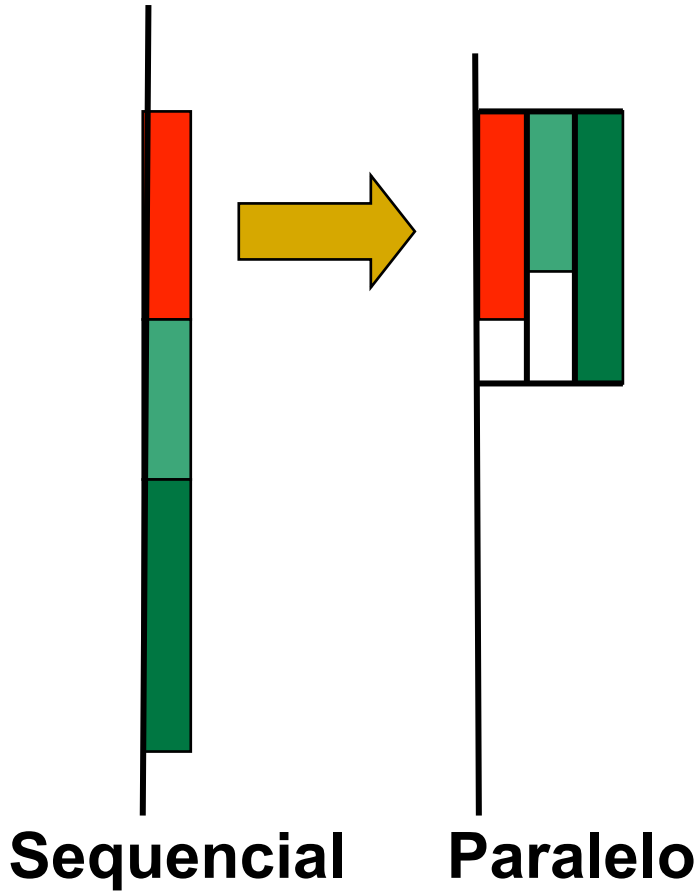
- Iterações são divididas em pedaços de tamanho 8
  - Se start = 3, então o primeiro pedaço é

    `i`={3,5,7,9,11,13,15,17}

Seções Paralelas

- Seções independente de código podem executar concorrentemente

```
#pragma omp parallel sections
{
    #pragma omp section
    phase1();
    #pragma omp section
    phase2();
    #pragma omp section
    phase3();
}
```



Single Construct

- Denota o bloco de código a ser executado por somente uma thread
  - A thread escolhida é dependente da implementação
- Possui uma barreira implícita ao final

```
#pragma omp parallel
{
   DoManyThings();
#pragma omp single
   {
     ExchangeBoundaries();
   }  // threads wait here for single
   DoManyMoreThings();
}
```

Master Construct

- Denota o bloco de código a ser executado somente pela thread mestre
- Não possui uma barreira implícita ao final

```
#pragma omp parallel
{
   DoManyThings();
#pragma omp master
   {                  // if not master skip to next stmt
     ExchangeBoundaries();
   }
   DoManyMoreThings();
}
```

Cláusula Nowait

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

```
#pragma single nowait
{ [...] }
```

- Utilizado quando as threads esperariam entre computações independentes

```
#pragma omp for schedule(dynamic,1) nowait
 for(int i=0; i<n; i++)
   a[i] = bigFunc1(i);

#pragma omp for schedule(dynamic,1)
 for(int j=0; j<m; j++)
   b[j] = bigFunc2(j);
```

Barrier Construct

- Barreiras explícitas de sincronização
- Cada thread deve esperar até que todas as threads cheguem na barreira

```
#pragma omp parallel shared (A, B, C)
{
        DoSomeWork(A,B);
        printf("Processed A into B\n");
#pragma omp barrier
        DoSomeWork(B,C);
        printf("Processed B into C\n");
}
```

Maiores Informações sobre OpenMP

- Site Oficial
  - http://www.openmp.org
- Tutorial
  - https://computing.llnl.gov/tutorials/openMP/