Processamento Paralelo

AULA 4

Multithreaded Programming

OpenMP

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

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

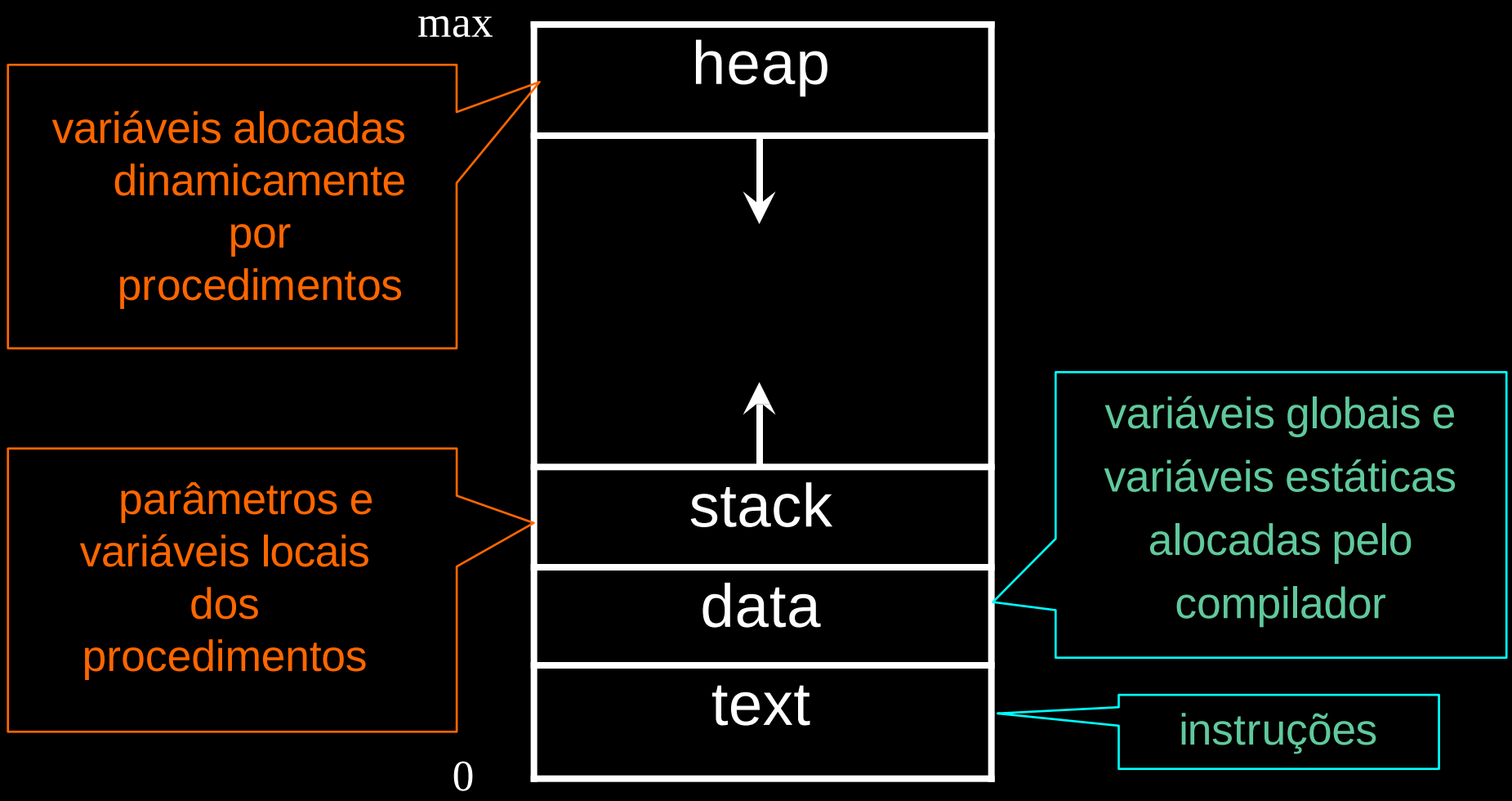
Agenda

- Processos e Threads
- Excursão Inicial por OpenMP
- Padrão OpenMP

Processos

- Processo é um programa em execução
- Unidade de trabalho do sistema operacional (SO)
  - SO multiplexa (no tempo) a execução dos processos na(s) CPU(s) sob seu controle
  - O próprio SO é um conjunto de processos privilegiados
  - Cada processo é representado e manipulado no SO por uma estrutura de dados específica
- Cada processo possui um espaço de endereçamento próprio

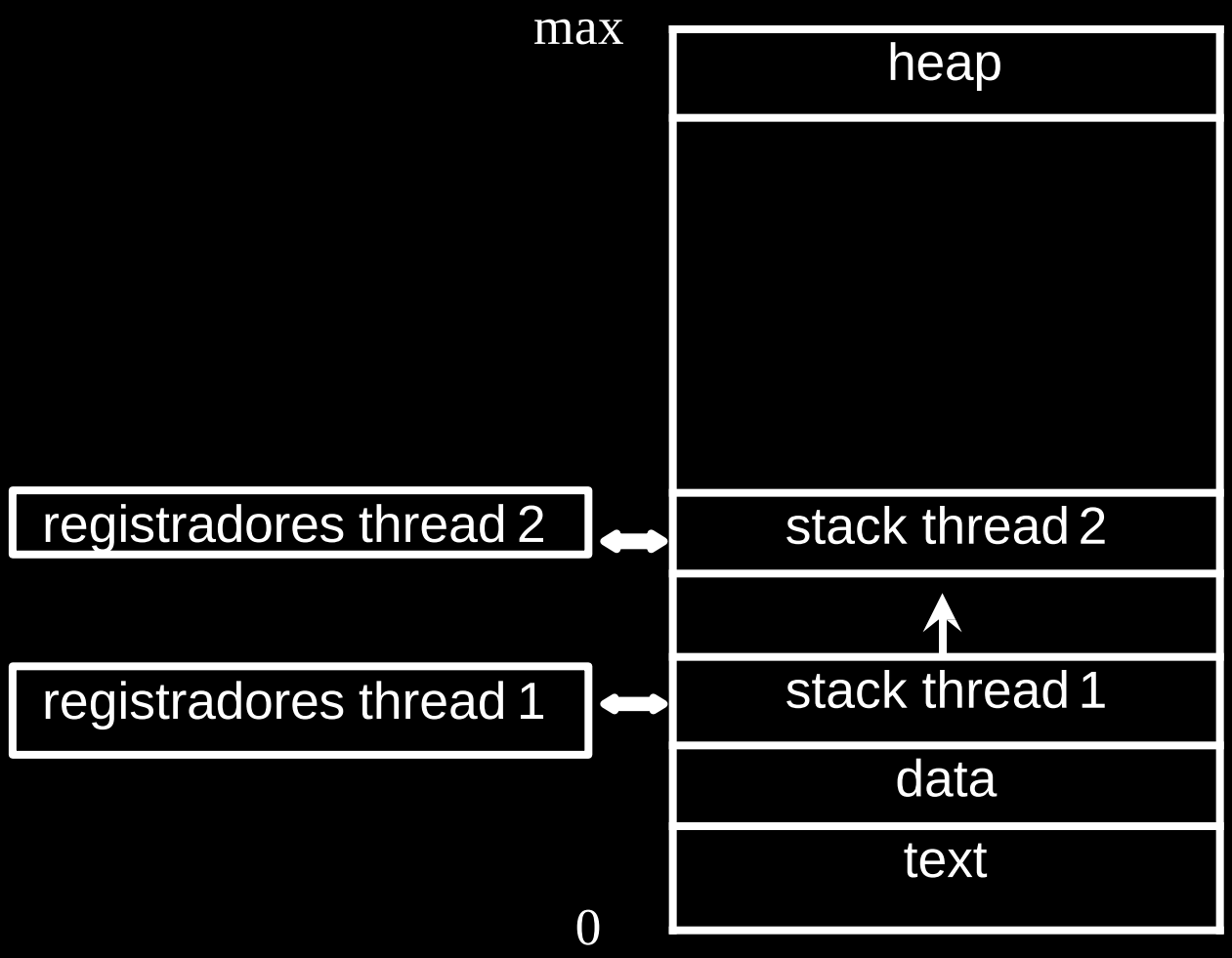
Espaço de Endereçamento de um Processo



Threads

- Thread (fio de execução) é um processo leve
  - Criar threads é muito mais rápido do que criar processos
- Cada thread é composta por uma pilha (stack) e um conjunto de registradores
  - A thread herda do processo que a criou os demais atributos
- Threads criadas por um processo compartilham o mesmo espaço de endereçamento
  - Logo, compartilham text, data, heap (o stack é privativo de cada thread)
- Conceitualmente, um processo contém pelo menos uma thread
  - Não necessariamente implementada dessa forma

Espaço de Endereçamento de Threads



PTHREADS

- O padrão POSIX para threads é PTHREADS
  - IEEE 1003.1c-1995

- Caracterísiticas de PTHREADS:
  1. Fundamentalmente, uma thread é o conjunto de registradores e um stack, herdando os demais atributos do processo que a criou;
  2. Threads são criadas por invocação da primitiva pthread_create
  3. A criação de uma thread gera a execução de um método (um dos argumentos de pthread_create) que já utiliza os registradores e stack recém criados
  4. pthread_create retorna imediatamente após a criação; as threads pai e filho executam concorrentemente, disputando os núcleos (processadores) existentes
  5. O processo que cria threads aguarda o término da execução de cada thread criada invocando pthread_join

Agenda

- Processos e Threads
- Excursão Inicial por OpenMP
- Padrão OpenMP

Excursionando por OpenMP

Fontes:

Padrão

Referências didáticas

- Padrão OpenMP em www.openmp.org
- Chandra, Menon, Dagum, Kohr: “Parallel Programming in OpenMP”, Academic Press, 2001
- Chapman, Jost, Van der Pas, “Using OpenMP”, MIT Press, 2008

Padrão OpenMP

Padrão *de facto* (www.openmp.org) com interfaces para C, C++ e Fortran

- O padrão é emitido pelo Open MP Architecture Review Board (ARB)
  - Entidade sem fins lucrativos
  - Membros da indústria (IBM, HP, NEC, Intel, ...), dos usuários (EPCC, NASA Ames, ...) e da academia (...)
- Histórico:
  - Diretivas proprietárias dominantes na década de 1980
  - Tentativa ANSI/ISO infrutífera nessa década
  - OpenMP ARB formado com membros da indústria em 1996
  - Padrão OpenMP 1.0 em 1997
  - Padrão OpenMP 2.0 em 2000
  - Padrão OpenMP 3.0 em 2008
  - Padrão OpenMP 4.0 em 2013
  - Padrão OpenMP 4.5 em 2015 (usaremos este)
  - Padrão OpenMP 5.0 em 2018

Padrão OpenMP

- Implementa o modelo fork-join de paralelismo por meio de diretivas (comentários identificados), variáveis de ambiente e funções
  - Diretivas em Fortran: !$OMP
  - Diretivas em C: #pragma omp
- Compiladores geram PTHREADS a partir das diretivas
  - Implementadas por todos os principais compiladores C e Fortran
- Como as diretivas são comentários, programas OpenMP cuidadosamente programados podem ser executados sequencialmente
  - Executam em paralelo quando compilados com chave para OpenMP

Compilação e Execução OpenMP

- Há uma chave de compilação que “liga” OpenMP
  - Chave não padronizada, varia com o compilador

    icc (ou ifort) -openmp

    pgcc (ou pgf90) -mp

    gcc (ou gfortran) -fopenmp
  - Vide “man page” do compilador

- O número de threads utilizado na execução é definido pelo valor da variável de ambiente OMP_NUM_THREADS. Em bash:

  export OMP_NUM_THREADS=4

- Logo:
  - Programas OpenMP devem ser escritos para qualquer número de tarefas paralelas, pois a quantidade de tarefas é definida na execução

Visão Superficial de OpenMP

- Ao paralelizar um programa, busque independências
  - Tarefas independentes podem ser executadas simultaneamente
- Visão Superficial: as tarefas em OpenMP são iterações de laços (*for* em C, *do* em Fortran)
  - Pois laços tipicamente contém a maior parte da computação
- Nessa visão superficial, OpenMP só será utilizado em laços com iterações independentes
  - pois iterações independentes podem ser executadas simultaneamente
- Nessa visão superficial, não seria possível paralelizar laços que calculam máximo de um vetor, por exemplo

Programa Sequencial

```
PROGRAM simple
  USE omp_lib
  IMPLICIT NONE
  INTEGER, PARAMETER :: vSize=1000000000
  REAL :: vec(vSize)
  INTEGER :: i
  DO i = 1, vSize
    vec(i) = REAL(i - vSize/2) ** 2
  END DO
  DO i = 1, vSize
    vec(i) = SQRT(vec(i))
  END DO
  WRITE(*,'(" maximum = ", f9.0,"; minimum =",f9.0)')&
    MAXVAL(vec), MINVAL(vec)
END PROGRAM simple
```

Constantes simbólicas e interface de procedimentos

Inicialização (ou Input)

Computação

Output

Iterações independentes permitem paralelismo

OpenMP Versão 0

```
PROGRAM simple
  USE omp_lib
  IMPLICIT NONE
  INTEGER, PARAMETER :: vSize=1000000000
  REAL :: vec(vSize)
  INTEGER :: i
  DO i = 1, vSize
       vec(i) = REAL(i - vSize/2) ** 2
  END DO
  !$OMP PARALLEL DO
  DO i = 1, vSize
       vec(i) = SQRT(vec(i))
  END DO
  WRITE(*,'(" maximum = ", f9.0,"; minimum =",f9.0)')&
                  MAXVAL(vec), MINVAL(vec)
END PROGRAM simple
```

Paralelo

Execução simultânea das iterações do laço

OpenMP Versão 0

Procs	Tempo (s)	Speed-up
1	167,87	1,00
2	128,20	1,31
3	109,62	1,54
4	102,36	1,65
5	97,99	1,72
6	95,08	1,77

OpenMP Versão 0 – O que falta?

```
PROGRAM simple  USE
     omp_lib  IMPLICIT NONE
     INTEGER, PARAMETER :: vSize=1000000000
     REAL :: vec(vSize)  INTEGER :: i
     DO i = 1, vSize
          vec(i) = REAL(i - vSize/2) ** 2
     END DO
     !$OMP PARALLEL DO
     DO i = 1, vSize
          vec(i) = SQRT(vec(i))
     END DO
     WRITE(*,'(" maximum = ", f9.0,"; minimum =",f9.0)') &
                    MAXVAL(vec), MINVAL(vec)
END PROGRAM simple
```

Paralelo

OpenMP Versão 1

```
PROGRAM simple
      USE omp_lib
      IMPLICIT NONE
      INTEGER, PARAMETER :: vSize=1000000000
      REAL :: vec(vSize)
      INTEGER :: i
      !$OMP PARALLEL DO
      DO i = 1, vSize
            vec(i) = REAL(i - vSize/2) ** 2
      END DO
      !$OMP PARALLEL DO
      DO i = 1, vSize
            vec(i) = SQRT(vec(i))
      END DO
      WRITE(*,'(" maximum = ", f9.0,"; minimum =",f9.0)')&
                  MAXVAL(vec), MINVAL(vec)
END PROGRAM simple
```

Paralelo

Paralelo

OpenMP Versão 1

Procs	Speed-up V0	Speed-up V1
1	1,00	1,00
2	1,31	1,41
3	1,54	1,83
4	1,65	2,07
5	1,72	2,25
6	1,77	2,37

OpenMP Versão 1 – O que falta?

```
PROGRAM simple
      USE omp_lib
      IMPLICIT NONE
      INTEGER, PARAMETER :: vSize=1000000000
      REAL :: vec(vSize)
      INTEGER :: i
      !$OMP PARALLEL D
      DO i = 1, vSize
         vec(i) = REAL(i - vSize/2) ** 2
      END DO
      !$OMP PARALLEL DO
      DO i = 1, vSize
            vec(i) = SQRT(vec(i))
      END DO
      WRITE(*,'(" maximum = ", f9.0,"; minimum =",f9.0)')&
                  MAXVAL(vec), MINVAL(vec)
END PROGRAM simple
```

Paralelo

Paralelo

Sequencial

Medindo o Tempo de Execução

Trecho	% Tempo Exec Sequencial
Initialização	7,79
SQRT	55,52
Min, Max	36,69

Aplicando Amdhal

Versão	Fração sequencial (%)	Speed-up máximo	Speed-up medido
V0	44,48	2,24	1,77
V1	36,69	2,72	2,37

Como Paralelizar a Escrita?

```
PROGRAM simple
  USE omp_lib
  IMPLICIT NONE
  INTEGER, PARAMETER :: vSize=1000000000
  REAL :: vec(vSize)
  INTEGER :: i
!$OMP PARALLEL DO
  DO i = 1, vSize
    vec(i) = REAL(i - vSize/2) ** 2
  END DO
!$OMP PARALLEL DO
  DO i = 1, vSize
    vec(i) = SQRT(vec(i))
  END DO
  WRITE(*,'(" maximum = ", f9.0,"; minimum =",f9.0)')&
    MAXVAL(vec), MINVAL(vec)
END PROGRAM simple
```

OpenMP Versão 2

```
PROGRAM simple  USE omp_lib
 IMPLICIT NONE
 INTEGER, PARAMETER :: vSize=1000000000
 REAL :: vec(vSize), maxTot(0:7), minTot(0:7)
 INTEGER :: i, this, nThreads
 !$OMP PARALLEL DO
 DO i = 1, vSize
    vec(i) = REAL(i - vSize/2) ** 2
 END DO
 !$OMP PARALLEL DO
 DO i = 1, vSize
    vec(i) = SQRT(vec(i))
 END DO
 maxTot = -1.0; minTot = REAL(vSize) ** 2
```

```
 !$OMP PARALLEL DO PRIVATE(this)
 DO i = 1, vSize
    this = OMP_GET_THREAD_NUM() ;
    nThreads = OMP_GET_NUM_THREADS()
    maxTot(this) = MAX(maxTot(this), vec(i))
    minTot(this) = MIN(minTot(this), vec(i))
 END DO
 WRITE(*,'(" maximum = ", f12.0,
                "; minimum =",f12.0)')
          &  MAXVAL(maxTot(0:nThreads-1)),
          MINVAL(minTot(0:nThreads-1))
END PROGRAM simple
```

OMP_NUM_THREADS=8

OpenMP Versão 2 - Detalhe

```
REAL :: ... maxTot(0:7), minTot(0:7)

maxTot = -1.0;   minTot = REAL(vSize) ** 2
!$OMP PARALLEL DO PRIVATE(this)
DO i = 1, vSize
  this = OMP_GET_THREAD_NUM()
  nThreads = OMP_GET_NUM_THREADS()
  maxTot(this) = MAX(maxTot(this), vec(i))
  minTot(this) = MIN(minTot(this), vec(i))
END DO
WRITE(*,'(" maximum = ", f12.0,"; minimum =",f12.0)')&
  MAXVAL(maxTot(0:nThreads-1)), &
  MINVAL(minTot(0:nThreads-1))
```

Paralelo

Sequencial

OpenMP – Versão 2

Procs	Speed-up
1	1,00
2	2,00
3	2,99
4	3,98
5	4,99
6	5,99

Conclusões da Excursão Inicial

- Paralelismo OpenMP explora independências na codificação do programa
  - Parece fácil
  - Na maioria dos casos, é a forma mais fácil de gerar paralelismo
- Paralelismo OpenMP pode mudar substancialmente o programa
  - Devido ao paralelismo desejado
- Pelo que vimos, paralelismo OpenMP aplica-se a laços com iterações independentes
  - Visão parcial;
  - OpenMP é mais eficiente quando aplicado a grandes regiões independentes do programa

Agenda

- Processos e Threads
- Excursão Inicial por OpenMP
- Padrão OpenMP

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

Sintaxe das Diretivas em C

- Sintaxe:

  #pragma omp nome-da-diretiva <lista-de-cláusulas> new-line

  onde
  - <lista-de-cláusulas> contém 0 ou mais cláusulas separadas por virgula
  - espaços separam o nome da diretiva da sentinela (omp) e da lista de cláusulas

- Diretivas são "case sensitive" em C - todas em caixa baixa (minúsculas)

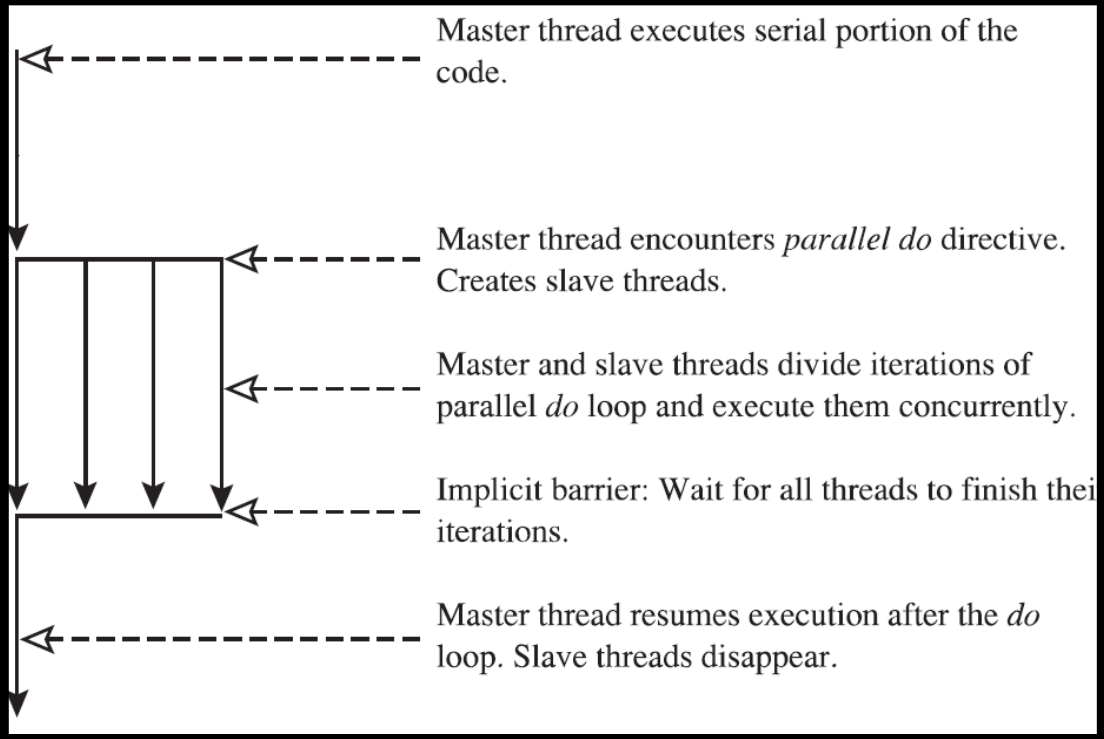
- Declaração de constantes simbólicas por "#include <omp.h>"

- Compilação condicional:
  - A macro _OPENMP é definida se o programa foi compilado com a chave de compilação OpenMP ativada

```
#ifdef _OPENMP
   threadID = omp_get_thread_num();
#else
   threadID = 0
#endif
```

Modelo de Execução

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 grupo de threads e torna-se o mestre do grupo
3. Todas as threads do grupo executam os comandos na região paralela
4. Ao término da região paralela, todas as threads do grupo sincronizam (barreira implícita) e apenas a thread mestre continua a execução; as demais threads são destruídas
5. Qualquer nº de regiões paralelas pode ser criado.



Definindo Número de Threads

- O número de threads criadas em uma região paralela é definido pelo valor da variável de ambiente OMP_NUM_THREADS. Em bash:

```
export OMP_NUM_THREADS=4
```

  - Pode ser alterado por invocação de funções de OpenMP e por cláusulas de diretivas

Obtendo Número de Threads

- No texto do programa OpenMP, a função omp_get_num_threads() retorna o número de threads do grupo atualmente em execução
  - Retorna 1 na parte sequencial do programa
- As threads são numeradas de 0 a (omp_get_num_threads() – 1)
  - Master Thread é a thread 0
- Já a função omp_get_thread_num() retorna o número (id) da thread que a invoca
  - entre 0 e (omp_get_num_threads() – 1) em região paralela
  - 0 na região sequencial do programa

Threads e Núcleos

- Conjunto de *N* threads pode ser executado por qualquer número de núcleos (processadores)
  - Por exemplo, 8 threads podem ser executadas por um único núcleo
- Em PTHREADS, o conjunto de threads faz parte de um único processo
  - Esse processo realiza “time sharing” com outros processos no SO
  - Pode haver disputa por núcleos se outros processos estiverem em execução
- Em execuções iterativas, o SO tipicamente fornece OMP_NUM_THREADS núcleos ao processo OpenMP
  - Restrito ao número máximo de núcleos na máquina
- Em execuções batch, o usuário define o número de núcleos no script enviado para o sistema batch e no comando aprun (*)

Threads e Núcleos

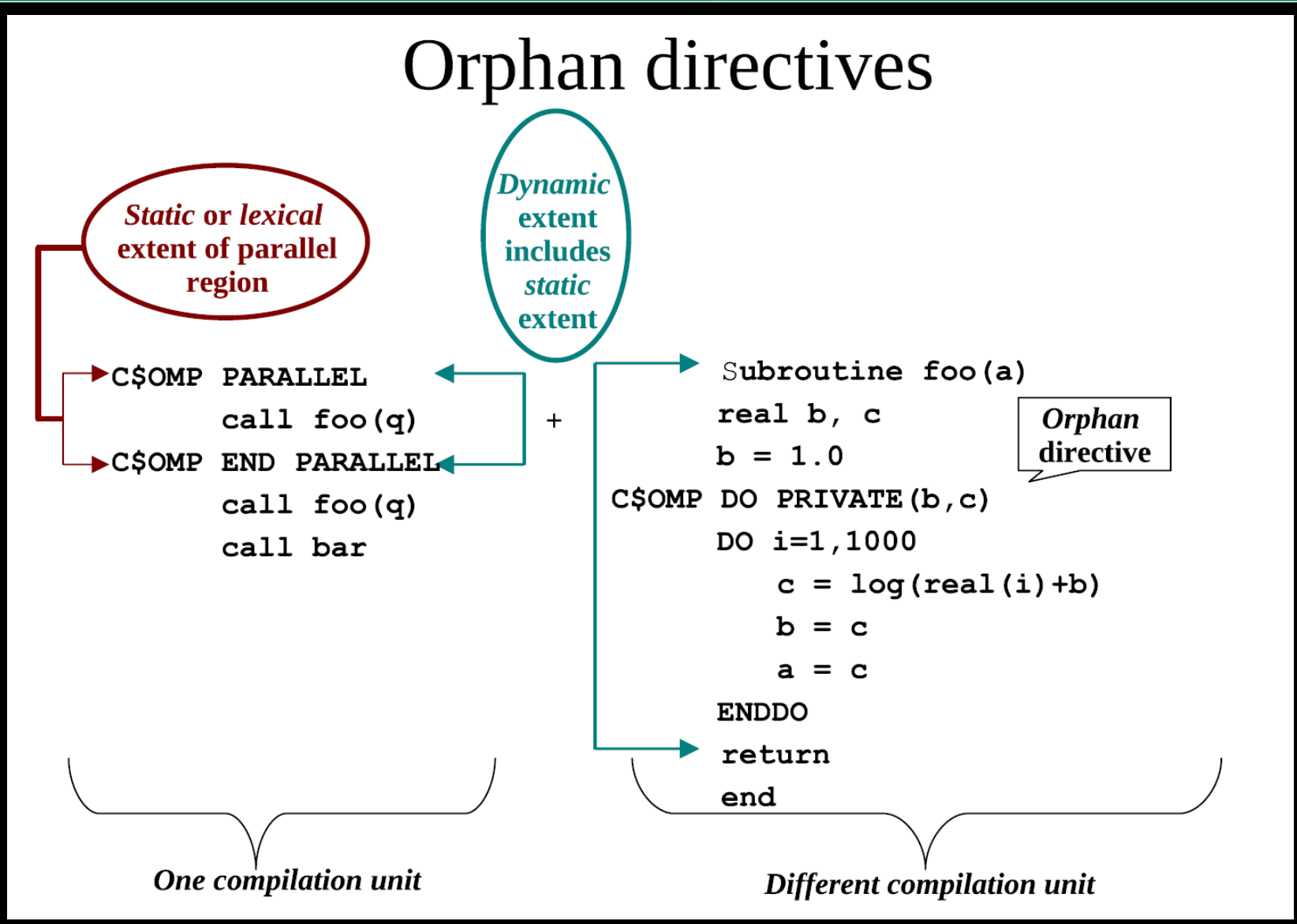
- PTHREADS possui mecanismo de atribuição de threads a núcleos
  - “scheduler” interno a PTHREADS, independente do “scheduler” do SO
  - Enquanto o SO atribui núcleos a processos, PTHREADS atribui threads a núcleos

- Em suma:
  - O SO (e o batch system) atribui um conjunto de núcleos a um processo, que pode ser com exclusividade ou não
  - PTHREADS controla como as N threads são atribuídas aos M núcleos fornecidos pelo SO

Escopo das Diretivas

- Abrangência Léxica (estática) de uma diretiva OpenMP:
  - É o código compreendido entre o começo e o fim de um bloco estruturado (em C, por exemplo) que vem após a dita diretiva.
  - Qualquer chamada de função/procedure dentro do bloco gera uma abrangência dinâmica à qual a diretiva também se aplica.
- Abrangência Dinâmica de uma diretiva OpenMP:
  - Inclui tanto o seu alcance estático quanto os statements que são parte de sua árvore de chamadas.
- Diretivas Órfãs OpenMP:
  - Uma diretiva que está dentro da abrangência dinâmica mas não dentro da abrangência estática de outra diretiva.

Escopo das Diretivas



Sumário

- Threads permitem a criação e destruição rápida de tarefas paralelas
- Excursão inicial por OpenMP demonstra a importância de medir o tempo de execução dos trechos do programa