

# Programação Paralela: Uma disciplina introdutória

*O paralelismo ao alcance de todos*

Prof. Dr. Gerson Geraldo H. Cavalheiro



Engenharia de Computação, Engenharia de Automação e Sistemas de Informação  
**Universidade Federal do Rio Grande**

25 de outubro de 2012

# Sumário

- ▶ Introdução
- Arquiteturas multi-core
- Programação multithread
- OpenMP
- Prática de programação
- Considerações finais

# Introdução

- Objetivos da apresentação
- Arquitetura de von Neumann
- Lei de Moore
- Arquiteturas paralelas
- Programação concorrente
- Motivação da programação em multi-core

# Objetivos da apresentação

- Desenvolver as habilidades de programação concorrente em arquiteturas multi-core.
- Discutir os benefícios da utilização de arquiteturas multi-core no desenvolvimento de programas concorrentes.
- Apresentar OpenMP, uma ferramenta para programação concorrente.
- Caracterizar a programação concorrente como uma necessidade de mercado

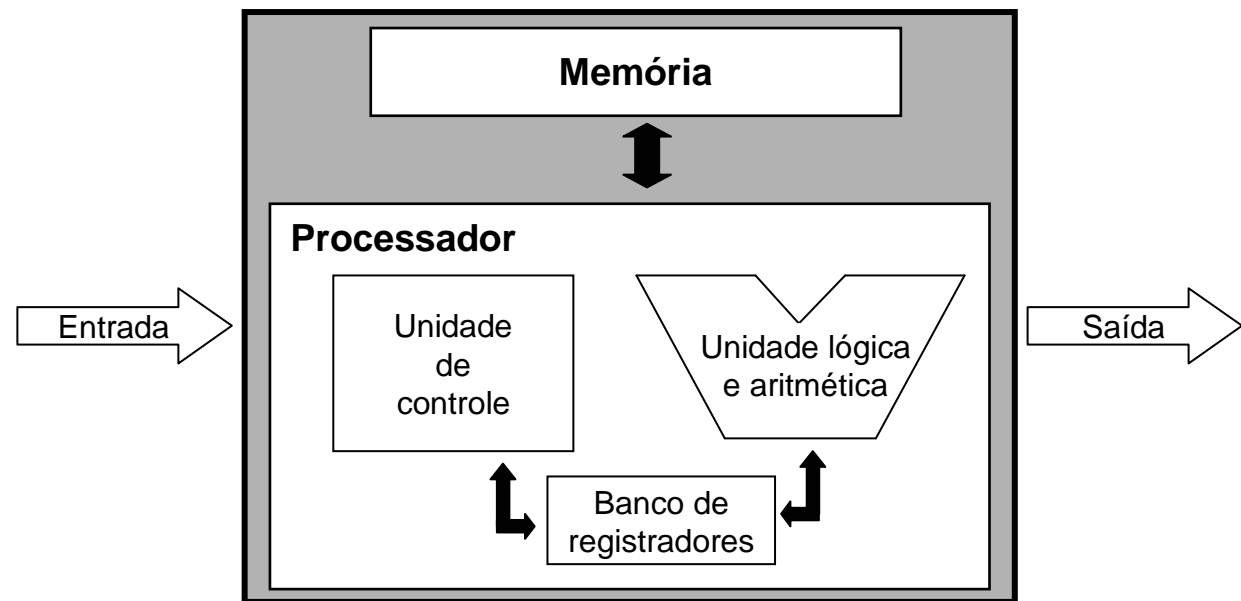
# Objetivos da apresentação

- Conceitos da arquitetura multi-core
  - Evolução
  - Arquitetura básica
  - Produtos de mercado
- Programação concorrente
  - Conceitos e princípios
  - Programação concorrente e paralela
  - Componentes de um programa concorrente
- Multiprogramação leve

# Arquitetura de von Neumann

## ■ Modelo de von Neumann (von Neumann, 1945)

- ❑ Execução seqüencial
- ❑ Memória para dados e programas
- ❑ Efeito colateral



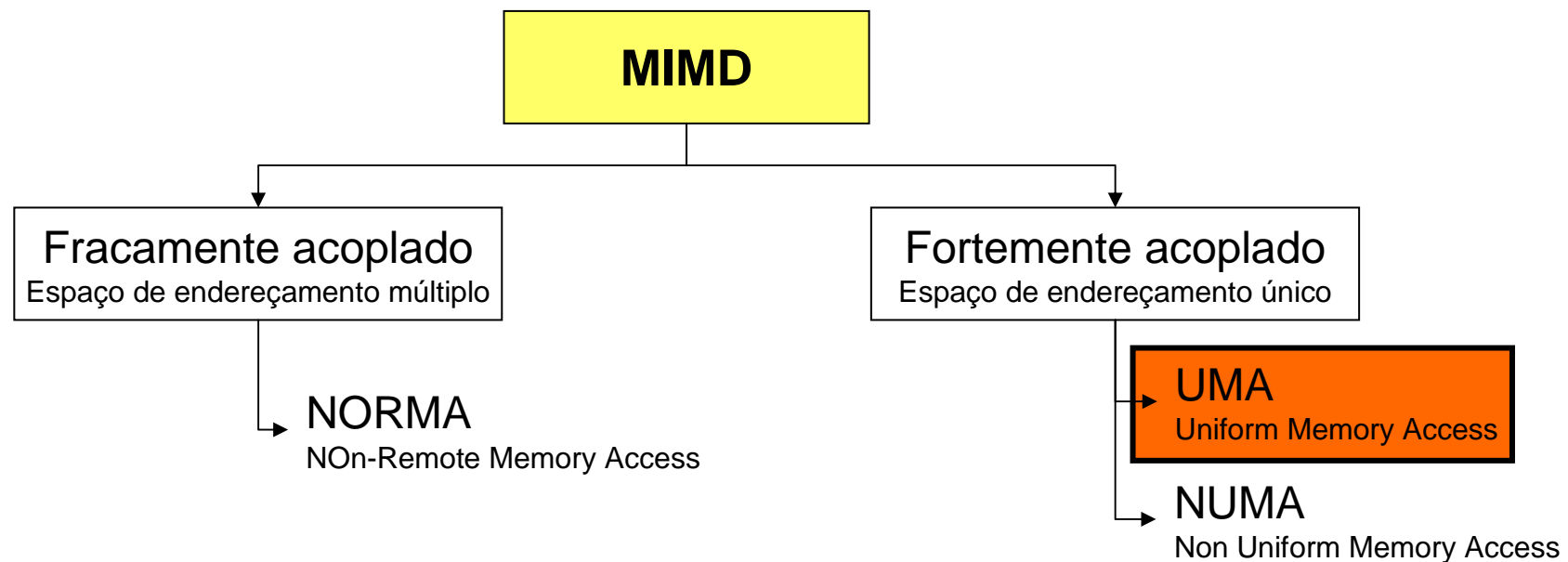
# Arquiteturas paralelas

- Classificação de Flynn (Flynn, 1972)
  - Segundo fluxos de controle e de dados

Dados \ Controle	Simples	Múltiplo
Simples	<b>SISD</b> von Neumann	<b>SIMD</b> array, sistólico
Múltiplo	<b>MISD</b> dataflow, pipeline	<b>MIMD</b> multicomputadores, multiprocessadores

# Arquiteturas paralelas

- Classificação de Flynn (Flynn, 1972)
- Classificação por compartilhamento de memória
  - Extensão a classificação de Flynn para a classe MIMD considerando a apresentação da memória

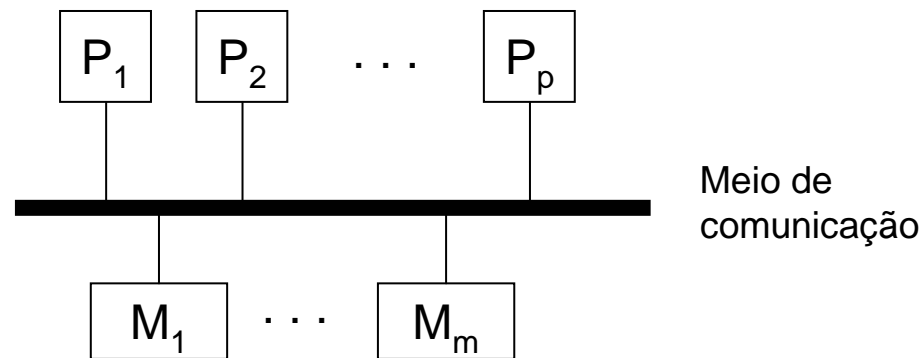




# Arquiteturas paralelas

## ■ UMA: Uniform Memory Access

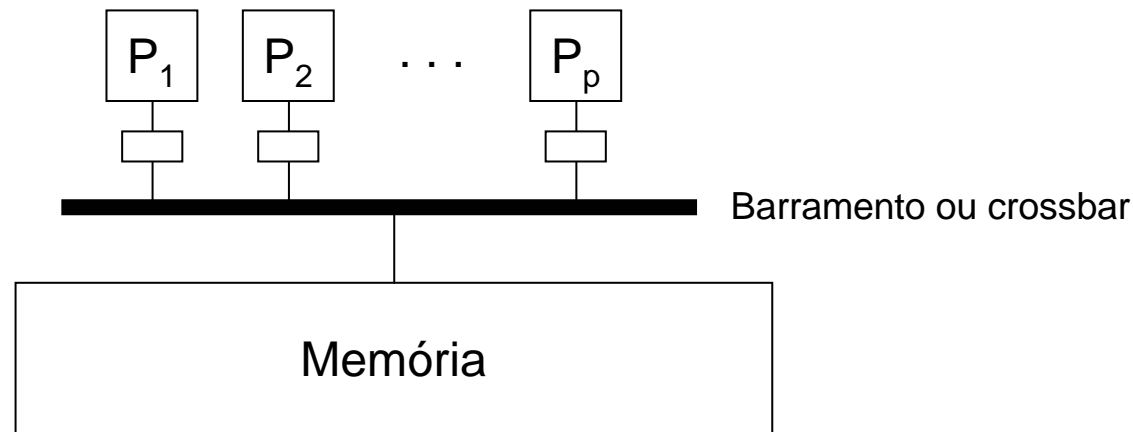
- ❑ **Multiprocessador**
- ❑ Memória global a todos os processadores
- ❑ Tempo de acesso idêntico a qualquer endereço de memória



Configuração de uma máquina UMA com  
 $p$  processadores e  $m$  módulos de memória

# Arquiteturas paralelas

- **UMA: Uniform Memory Access**
  - **SMP: Symmetric MultiProcessor**



- Os processadores são idênticos e operam como definido pela arquitetura von Neumann
- O acesso a memória é compartilhado

# Programação concorrente

## ■ Concorrência

- ❑ Possibilidade de execução simultânea de dois ou mais fluxos de execução, não impedindo disputa por recursos
  - Importante: a definição inclui a idéia de disputa por recursos mas não é restrita à ela!
- ❑ A concorrência incorpora a idéia de que existe uma convergência dos fluxos para um ponto comum
  - Uma eventual sincronização
  - A produção do resultado final
- ❑ E também para idéia de ação colaborativa

# Programação concorrente

- **Concorrência**

- Indica independência temporal
- O número de atividades concorrentes pode ser maior que a quantidade de recursos de processamento disponíveis

- **Paralelismo**

- Indica execução simultânea
- A quantidade de recursos de processamento disponível permite que duas ou mais atividades sejam executadas simultaneamente

- É usual utilizar a expressão **programação paralela** para o tipo de programação focada em desempenho

- **Na nossa terminologia**, programação paralela e programação concorrente são utilizadas com o mesmo significado

# Motivação da programação em multi-core

## ■ **Tendência**

- ❑ Limites tecnológicos para aumento de desempenho com foco no aumento da frequência de clock
- ❑ Arquiteturas com menor consumo de energia
  - Grande problema da sociedade moderna
- ❑ Ainda mais cores no futuro próximo

## ■ **Facilidade de exploração**

- ❑ Reflete naturalmente o modelo de programação multithread
- ❑ Suportado pelos SOs atuais

## ■ **Oportunidade**

- ❑ Migração das maioria das aplicações desenvolvidas para mono-core

## ■ **Aumento de desempenho**

- ❑ Execução paralela de aplicações em HW de baixo custo

# Sumário

- ✓ Introdução
- ▶ **Arquiteturas multi-core**
- Programação multithread
- OpenMP
- Prática de programação
- Considerações finais

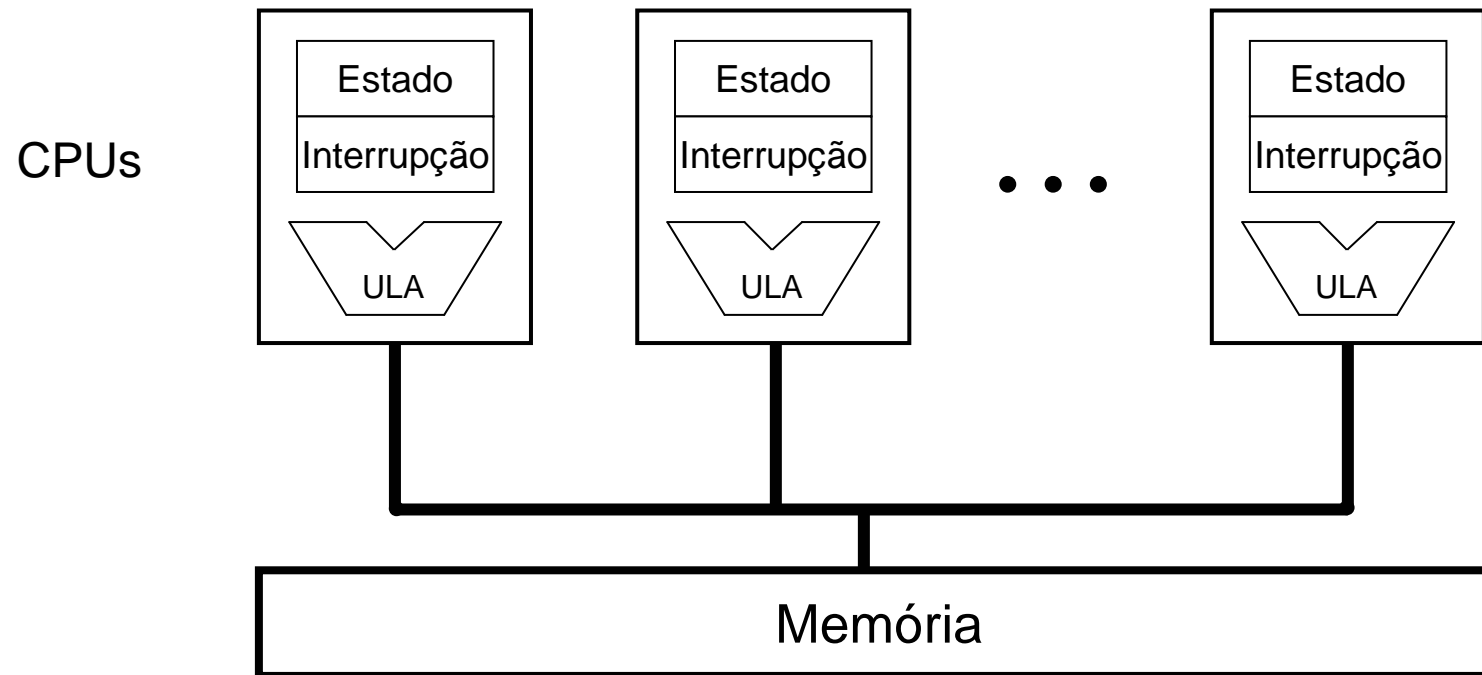
# Arquiteturas multi-core

- Arquitetura SMP
- Multiprocessamento em um chip
- O multi-core
- Produtos comerciais

# Arquitetura SMP

## ■ Symmetric MultiProcessor

- Arquitetura paralela dotada de um conjunto de processadores idênticos (simétricos) compartilhando acesso a uma área de memória comum





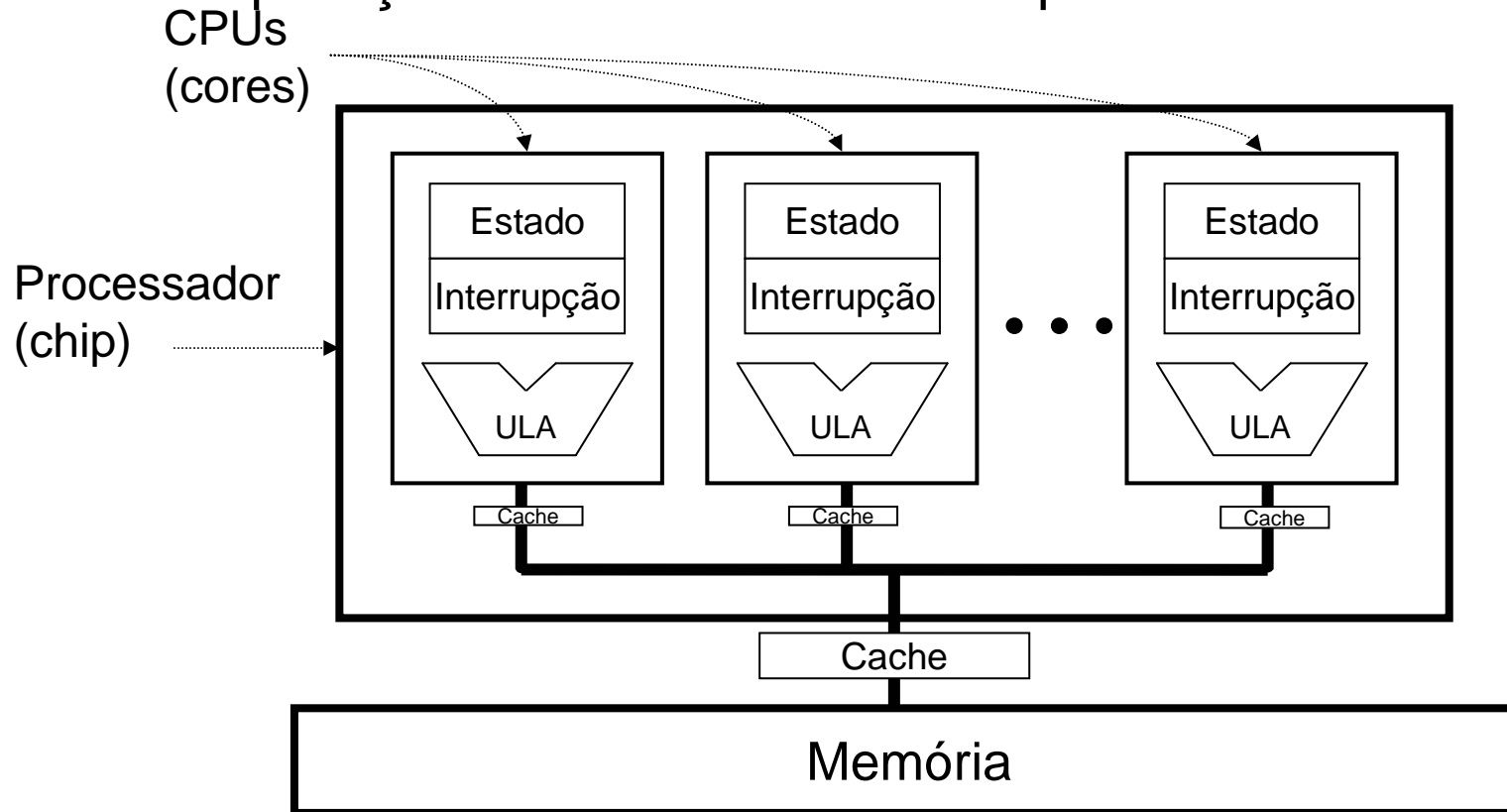
# Arquitetura SMP

## ■ Symmetric MultiProcessor

- ❑ Arquitetura paralela dotada de um conjunto de processadores idênticos (simétricos) compartilhando acesso a uma área de memória comum
- ❑ SO único responsável pelo escalonamento de atividades (fluxos de execução) entre os processadores
- ❑ Cada CPU replica todo conjunto de recursos necessários à execução de um fluxo.
- ❑ Fator limitante de desempenho: contenção no acesso à memória

# O multi-core

- Tecnologia em hardware na qual múltiplos cores são integrados em um único chip
- Multiplicação total dos recursos de processamento

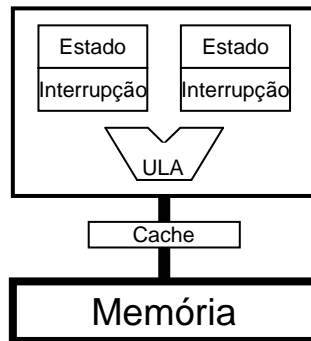


# O multi-core

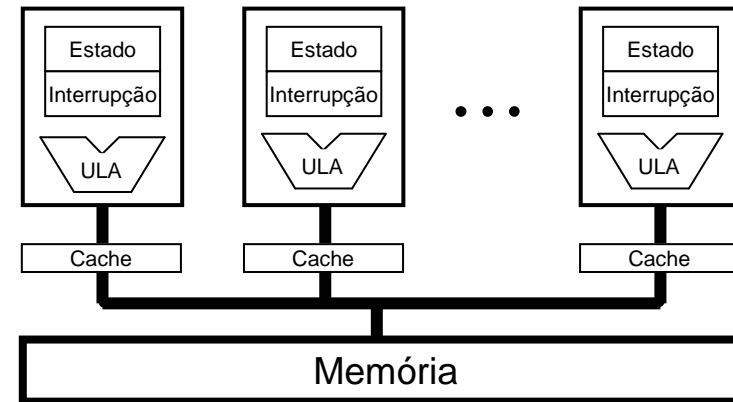
- Tecnologia em hardware na qual múltiplos cores são integrados em um único chip
- Multiplicação total dos recursos de processamento
- Grande vantagem: compatível com os códigos existentes!

# O multi-core

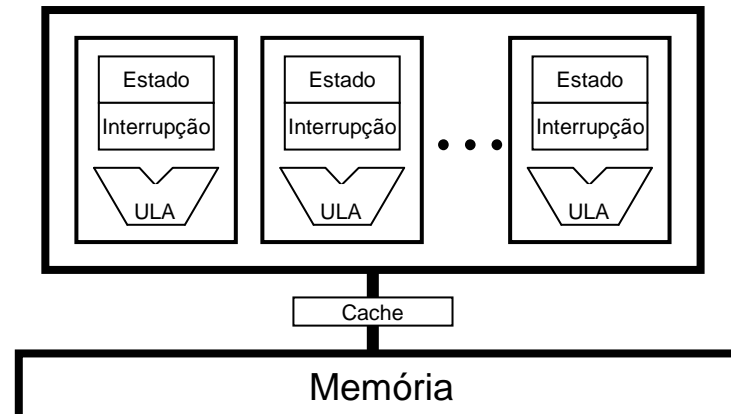
## ■ Comparativo – visão macro



Processador HyperThreading



Multiprocessador SMP



Processador multi-core

# Ilustração

## ■ Multi-core architecture

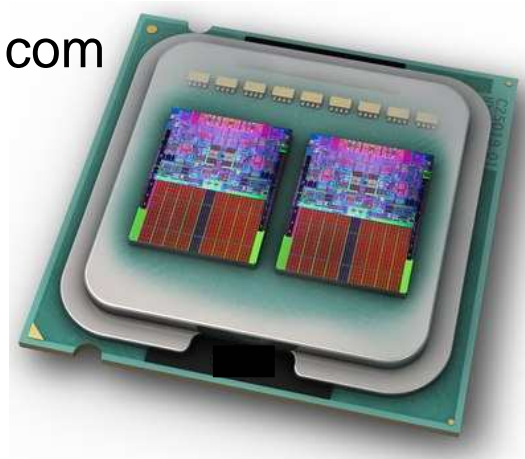
□ [http://cache-www.intel.com/cd/00/00/24/49/244978\\_244978.swf](http://cache-www.intel.com/cd/00/00/24/49/244978_244978.swf)

Fonte: intel.com

## ■ Explore a dual core Opteron™

□ [http://www.amd.com/us-en/assets/content\\_type/Additional/33635BDual-CoreDiagramFIN.swf](http://www.amd.com/us-en/assets/content_type/Additional/33635BDual-CoreDiagramFIN.swf)

Fonte: amd.com



Imagens ilustrativas

# Sumário

- ✓ Introdução
- ✓ Arquiteturas multi-core
- ▶ **Programação multithread**
- Ferramentas de programação
- Prática de programação
- Considerações finais

# Sumário

- ✓ Introdução
- ✓ Arquiteturas multi-core
- **Programação multithread**
- OpenMP
- Prática de programação
- Considerações finais

# Programação multithread

- Programação concorrente
- Thread
- Casos de estudo
  - Paralelismo de tarefas
  - Paralelismo de dados



# Programação concorrente

- Programação concorrente
  - Voltada às questões da aplicação
- Programação paralela
  - Voltada às questões do hardware disponível
- Execução concorrente
  - Competição no acesso a recursos de hardware
- Execução paralela
  - Replicação de recursos de hw permite simultaneidade

# Programação concorrente

- Programação concorrente, pois
  - A programação concorrente inclui a programação paralela
  - Permite focar nas questões da aplicação
    - Ou seja, não requer do programador conhecimentos sobre o hardware paralelo disponível
  - Idealmente: alto grau de portabilidade
    - Considerando que a ferramenta de programação possui um modelo de execução eficiente

# Programação concorrente

- Programação concorrente
  - Extensão ao paradigma Imperativo
    - Contexto de trabalho no curso
  - Execução segundo a arquitetura de von Neumann
    - Programa seqüencial é composto por:
      - Uma seqüência de instruções
      - Um conjunto de dados
    - Execução seqüencial
      - Instruções modificam estados de memória
      - Efeito colateral
        - A execução de uma instrução implica na alteração de um **estado** (posição de memória)

# Programação concorrente

- Programação concorrente
  - Extensão ao paradigma Imperativo
  - Execução segundo a arquitetura de von Neumann
    - Execução seqüencial:

The diagram illustrates the sequential execution of eight instructions (I1 to I8) within a rounded rectangular box. The instructions are listed vertically: I1: MOV DX, 0; I2: MOV AX, [313]; I3: ADD DX, AX; I4: DEC AX; I5: CMP AX, 0; I6: JNE I3; I7: MOV AX, [313]; I8: CALL RotinaImpressão. Blue arrows indicate the flow from I1 to I2, I2 to I3, I3 to I4, I4 to I5, I5 to I6, I6 to I7, and I7 to I8. A green box highlights instruction I3, and a green arrow points from I2 to I3. Another green arrow points from I3 to I8, bypassing instructions I4 through I7, which represents a jump or call operation.

```
I1: MOV DX, 0
I2: MOV AX, [313]
I3: ADD DX, AX
I4: DEC AX
I5: CMP AX, 0
I6: JNE I3
I7: MOV AX, [313]
I8: CALL RotinaImpressão
```

**Efeito da execução de  
uma instrução:**

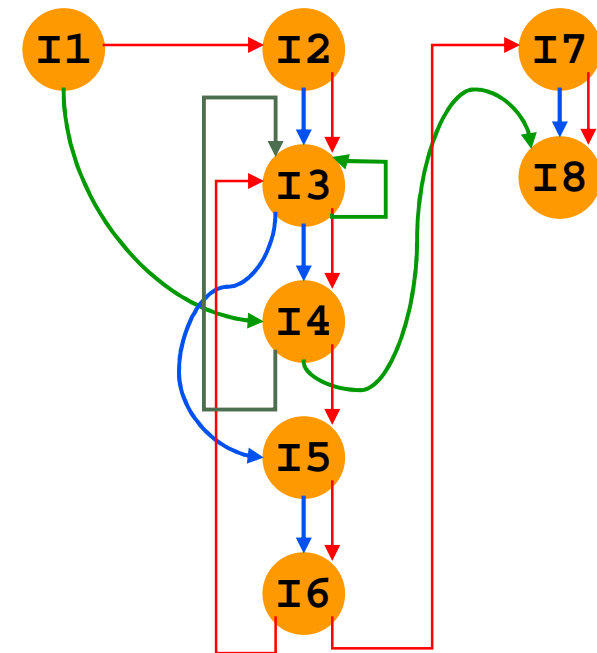
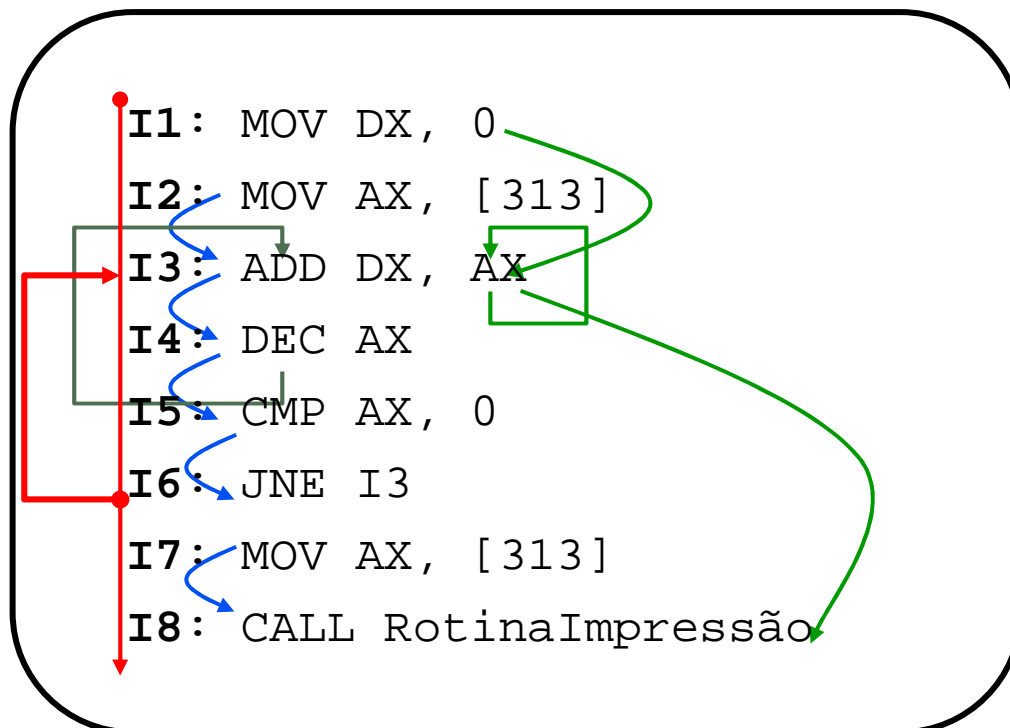
Escrita em memória

**Comunicação**

# Programação concorrente

## ■ Programação concorrente

- Extensão ao paradigma Imperativo
- Execução segundo a arquitetura de von Neumann
  - Execução seqüencial:



# Programação concorrente

- Programação concorrente
  - Extensão ao paradigma Imperativo
  - Execução segundo a arquitetura de von Neumann
    - Execução seqüencial:
      - **Unidade execução:** a instrução
      - **Comunicação:** alteração de um estado de memória
      - **Sincronização:** implícita, pois uma instrução não é executada antes que a anterior termine
    - Execução concorrente deve, igualmente, definir:
      - A unidade de execução elementar
      - O mecanismo de comunicação utilizado
      - A coordenação da execução para controle do acesso aos dados

# Programação concorrente

- Programação concorrente
  - Extensão ao paradigma Imperativo
  - Execução segundo a arquitetura de von Neumann
- Ferramentas de programação **multithread** oferecem recurso para:
  - Definir unidades de execução em termos de conjuntos de instruções ou procedimentos
  - Identificar regiões de memória (dados) compartilhados entre as unidades de execução
  - Controlar acesso aos dados compartilhados

# Programação concorrente

- Modelos de Decomposição Paralela

- **Paralelismo de tarefas**

- O paralelismo é definido em termos de atividades independentes, processando sobre conjunto de dados distintos.
    - Sincronização no início e no fim da execução concorrente e, eventualmente, durante a execução.

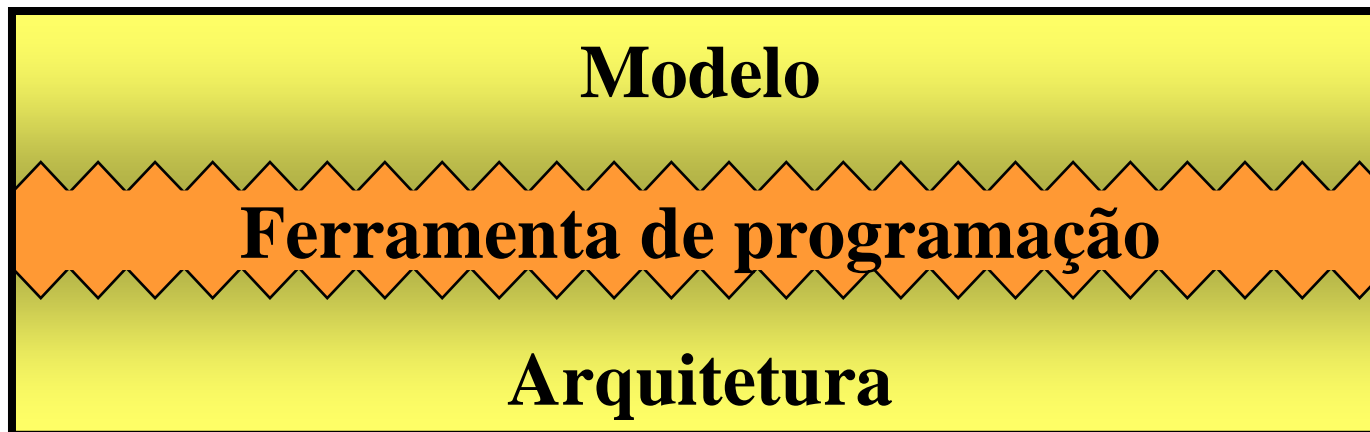
- **Paralelismo de dados**

- O paralelismo é definido em termos de dados que podem sofrer a execução da mesma operação de forma independente.
    - Sincronização no início e no final da execução concorrente.



# Programação concorrente

- Modelos de programação
  - Oferecem “níveis de abstração” dos recursos oferecidos pelas arquiteturas e pelas ferramentas
  - Ferramentas oferecem recursos para programação de arquiteturas reais



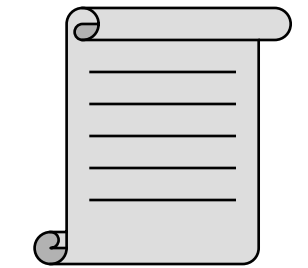
- É possível que uma ferramenta permita a exploração de mais de um tipo de paralelismo ou a utilização de diferentes modelos de concorrência. No entanto, cada ferramenta é mais conveniente para um determinado esquema

# Thread

- Processo leve
- Noção de processo
  - Um programa define uma seqüência de instruções e um conjunto de dados
  - Um processo representa uma instância de um programa em execução. A seqüência de instruções do programa define um fluxo de execução a ser seguido e o espaço de endereçamento utilizado. O processo também possui associado o registro de recursos que utiliza e é manipulado pelo sistema operacional.

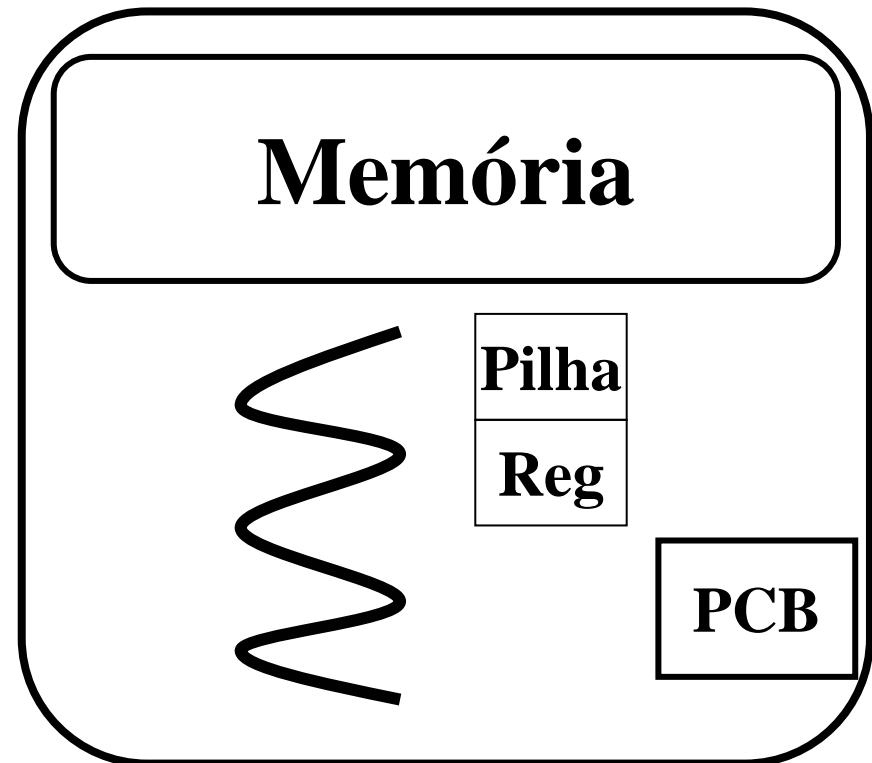
# Thread

- Processo leve
- Noção de processo



programa.exe

\$> programa.exe



# Thread

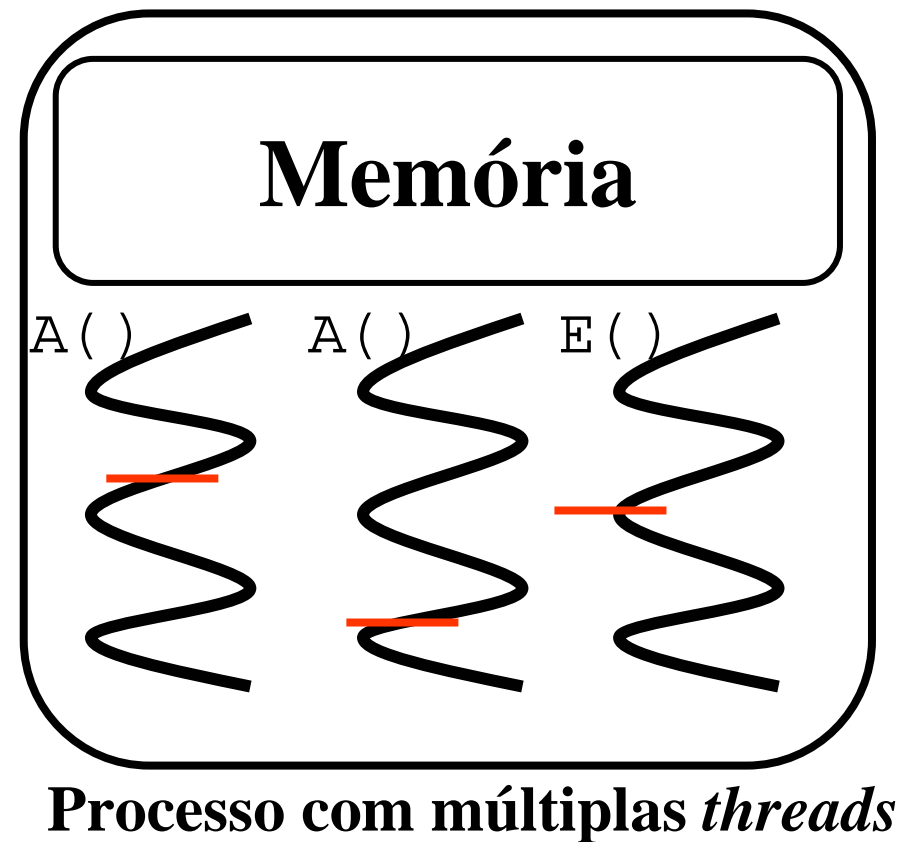
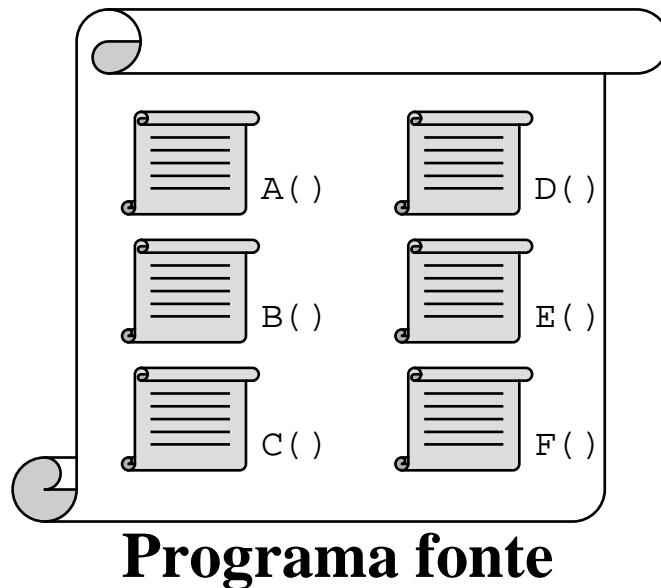
- Processo leve
- Noção de processo
  - Informações no PCB (Process Control Block)
    - Process ID
    - User and User Grup ID
    - Diretório de trabalho
    - Descritor de arquivos
    - Manipuladores de sinais
    - Bibliotecas de compartilhadas
    - Ferramentas de comunicação (pipes, semáforos etc)

# Thread

- Processo leve
- Noção de processo
- Processo multithread
  - Um programa define várias seqüência de instruções e um conjunto de dados
  - Uma thread representa uma instância de uma seqüência de instruções em execução. Cada seqüência de instruções do programa define um fluxo de execução a ser seguido e os dados o espaço de endereçamento utilizado. As threads compartilham o registro de recursos que utilizados pelo processo.

# Thread

- Processo leve
- Noção de processo
- Processo multithread

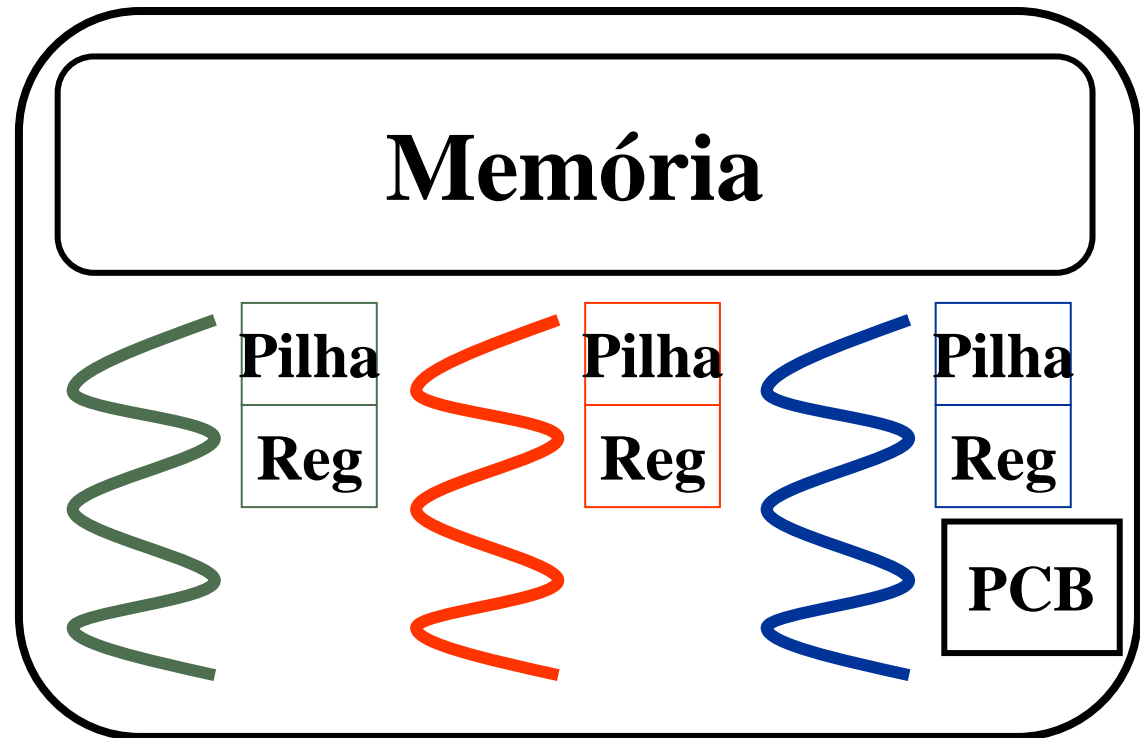


# Thread

- Processo leve
- Noção de processo
- Processo multithread
  - Os fluxos são independentes
    - Competição pelos recursos da arquitetura
  - Trocas de dados através de leituras e escritas em memória
    - Competição pelos dados na memória compartilhada

# Thread

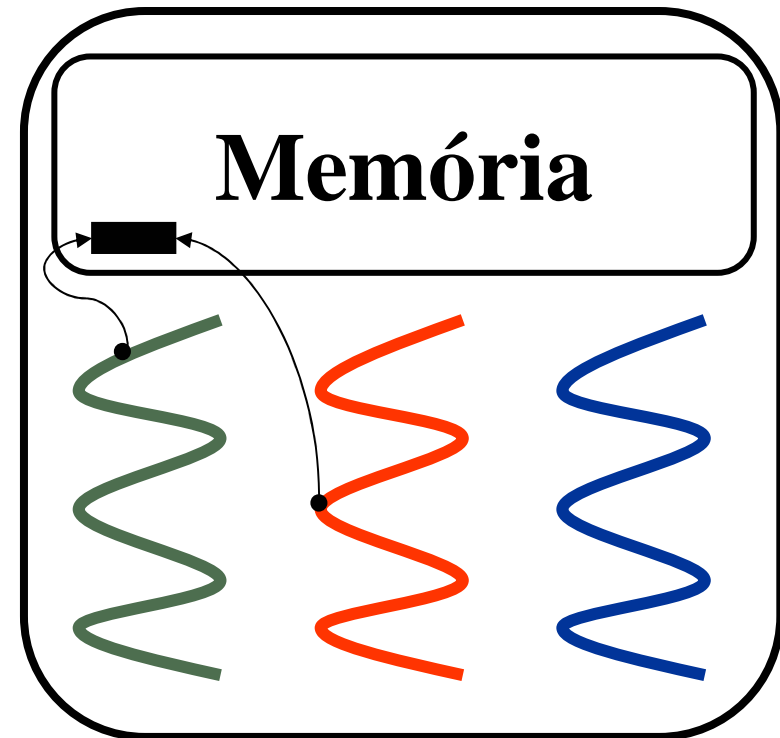
- Processo leve
- Noção de processo
- Processo multithread





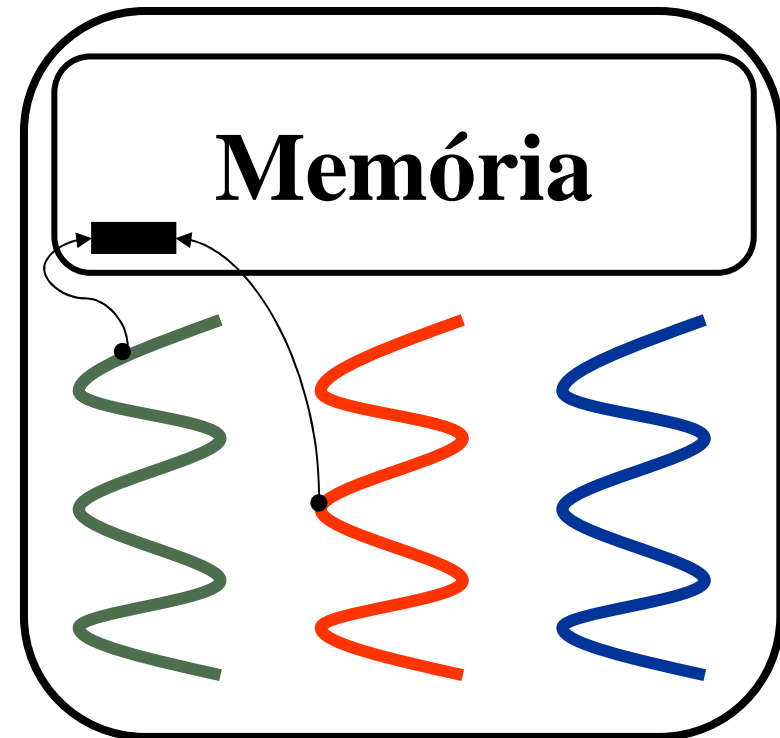
# Thread

- Processo leve
- Noção de processo
- Processo multithread
  - Acesso ao dado compartilhado através de operações de leitura e escrita convencionais
    - **Escrita:**  
`Dado = valor`
    - **Leitura:**  
`variável = Dado`



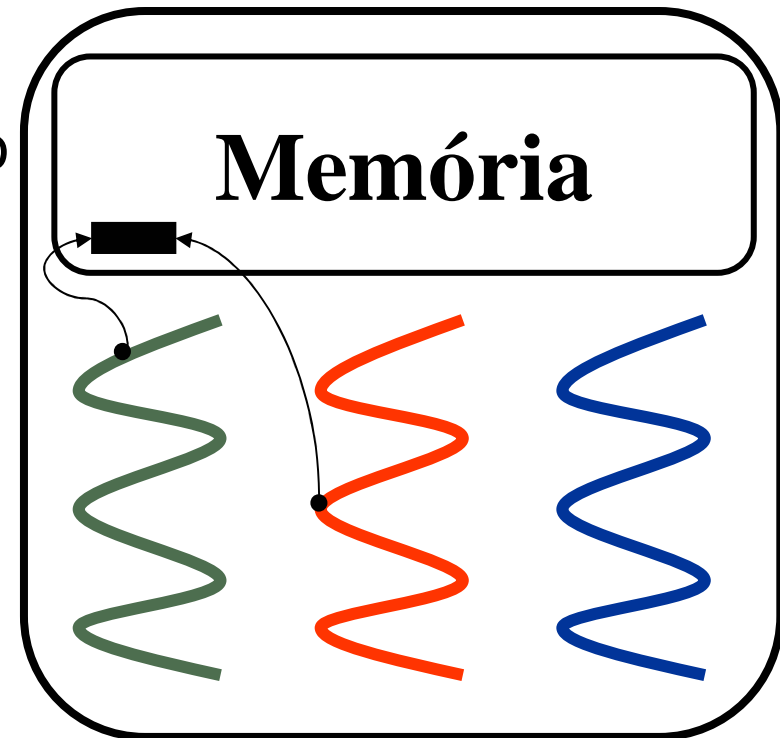
# Thread

- Processo leve
- Noção de processo
- Processo multithread
  - Acesso ao dado compartilhado através de operações de leitura e escrita convencionais
  - O programador é responsável por introduzir instruções para sincronização entre fluxos



# Thread

- Processo leve
- Noção de processo
- Processo multithread
  - Acesso a dado
  - Instruções de sincronização
  - **Seção crítica:** trecho de código com instruções que manipulam dado compartilhado entre fluxos de execução distintos



# Thread

## ■ Processo leve

□ Thread: custo de manipulação reduzido em relação ao processo

- Chaveamento de contexto em tempo menor
- Comunicação entre threads via memória compartilhada
- Criação/Destrução de threads mais eficiente que de processos

□ Informações de cada thread

- Pilha
- Registradores
- Propriedades de escalonamento (prioridade, política)
- Sinais pendentes ou bloqueados
- Dados específicos do thread (errno)

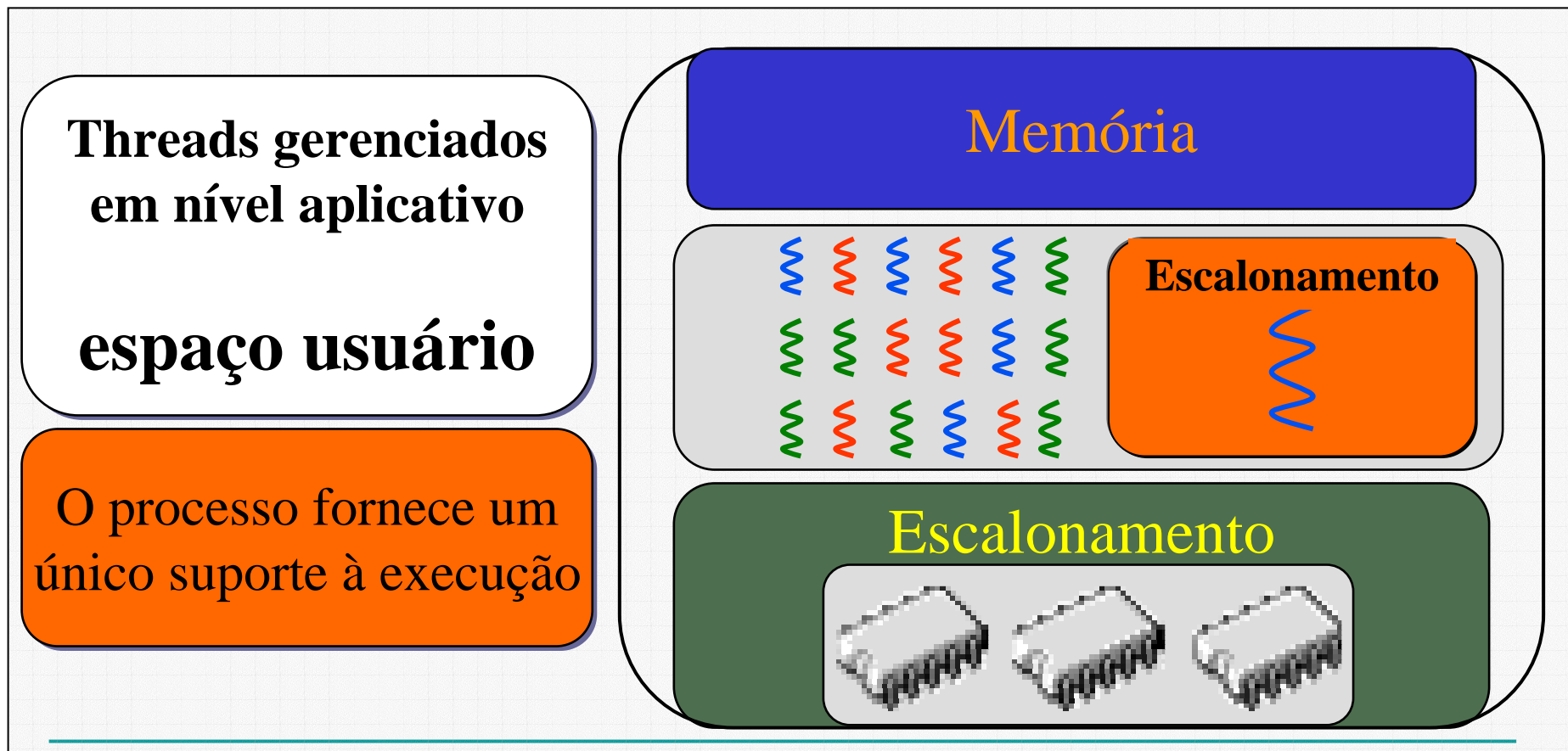
# Thread

## Implementações

$1 : 1$	$N : 1$	$M : N$
Thread sistema	Thread usuário	Misto
<b>SMP</b>	<b>Mais leves</b>	<b>Compromisso</b>

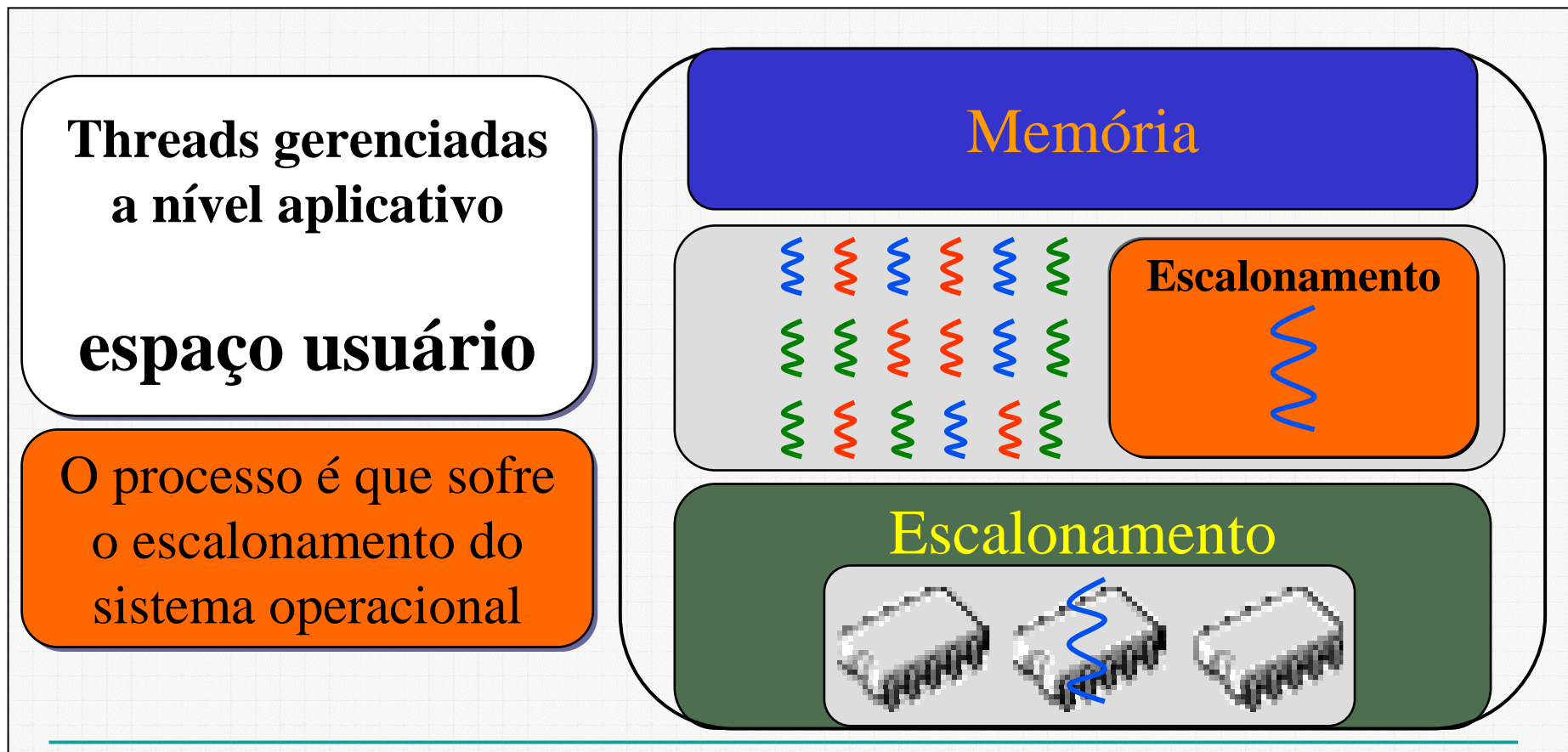
# Thread

## Modelos de Threads - $N : 1$



# Thread

## Modelos de Threads - $N : 1$



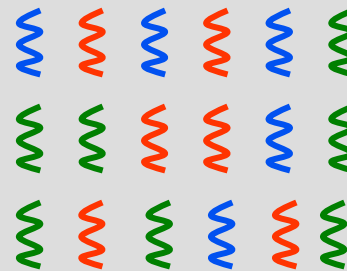
# Thread

## Modelos de Threads - $N : 1$

### Vantagens

- Aumento do nível de concorrência que pode ser expresso para a aplicação
- Os threads são bastante *leves*

Memória



Escalonamento



Escalonamento



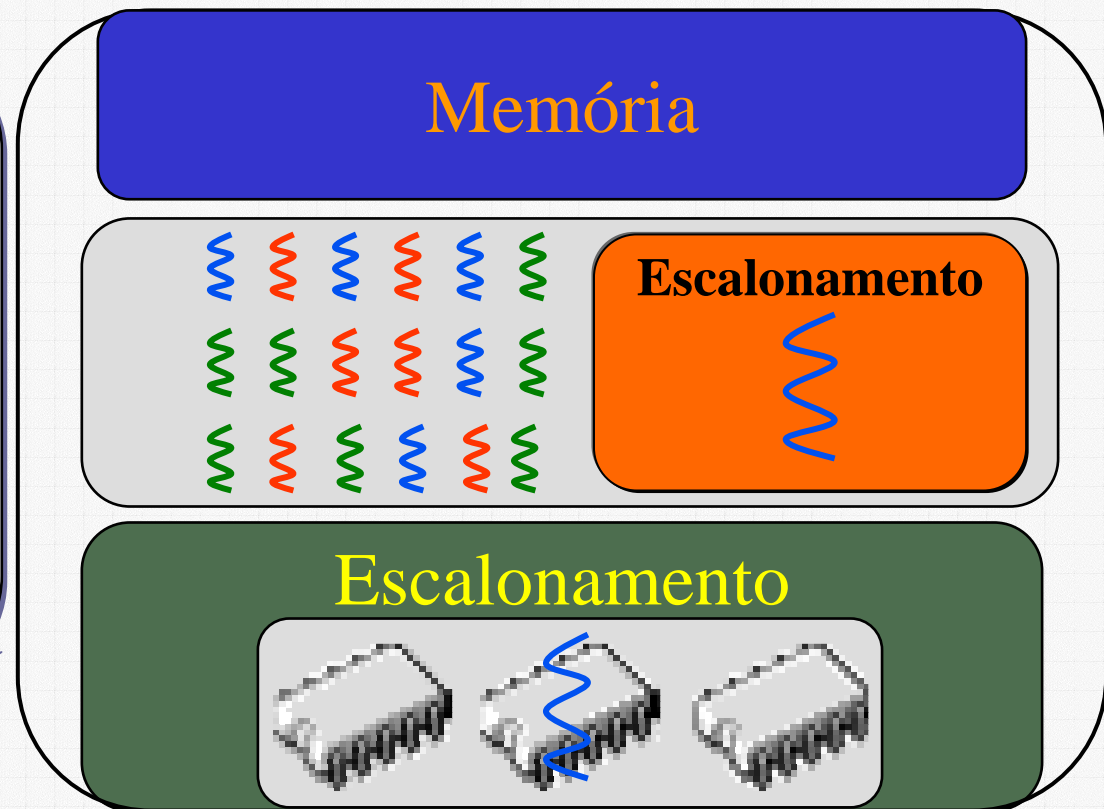


# Thread

## Modelos de Threads - $N : 1$

### Porém...

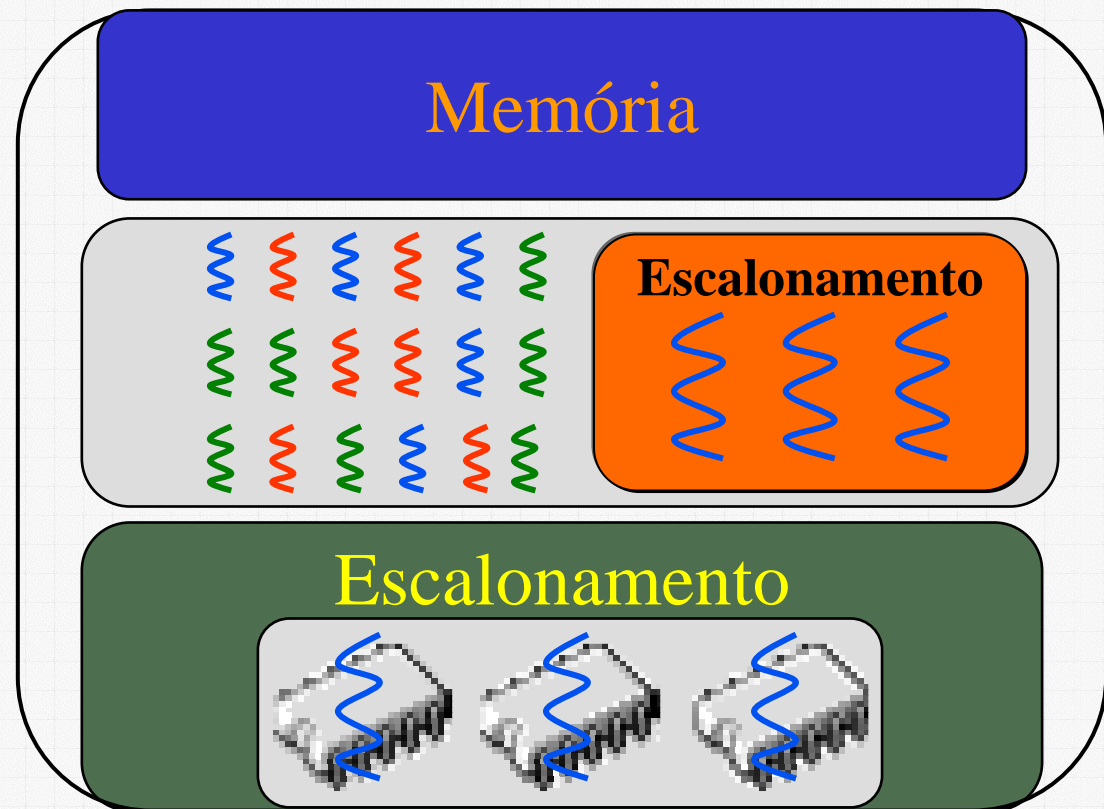
- Não explora o paralelismo natural de uma arquitetura SMP
- Um thread pode bloquear todo processo ao realizar uma chamada bloqueante (p.ex. `recv`, `scanf`) ao sistema



# Thread

## Modelos de Threads - $N : M$

Modelo misto  
gerenciamento tanto no  
**espaço usuário** quanto  
pelo sistema operacional



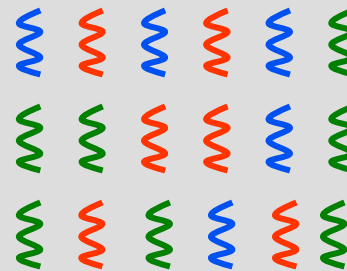
# Thread

## Modelos de Threads - $N : M$

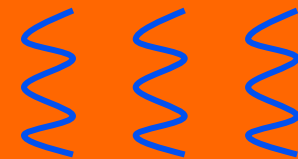
### Vantagens

- Separação entre a descrição da concorrência existente na aplicação do paralelismo real existente na arquitetura

### Memória



### Escalonamento

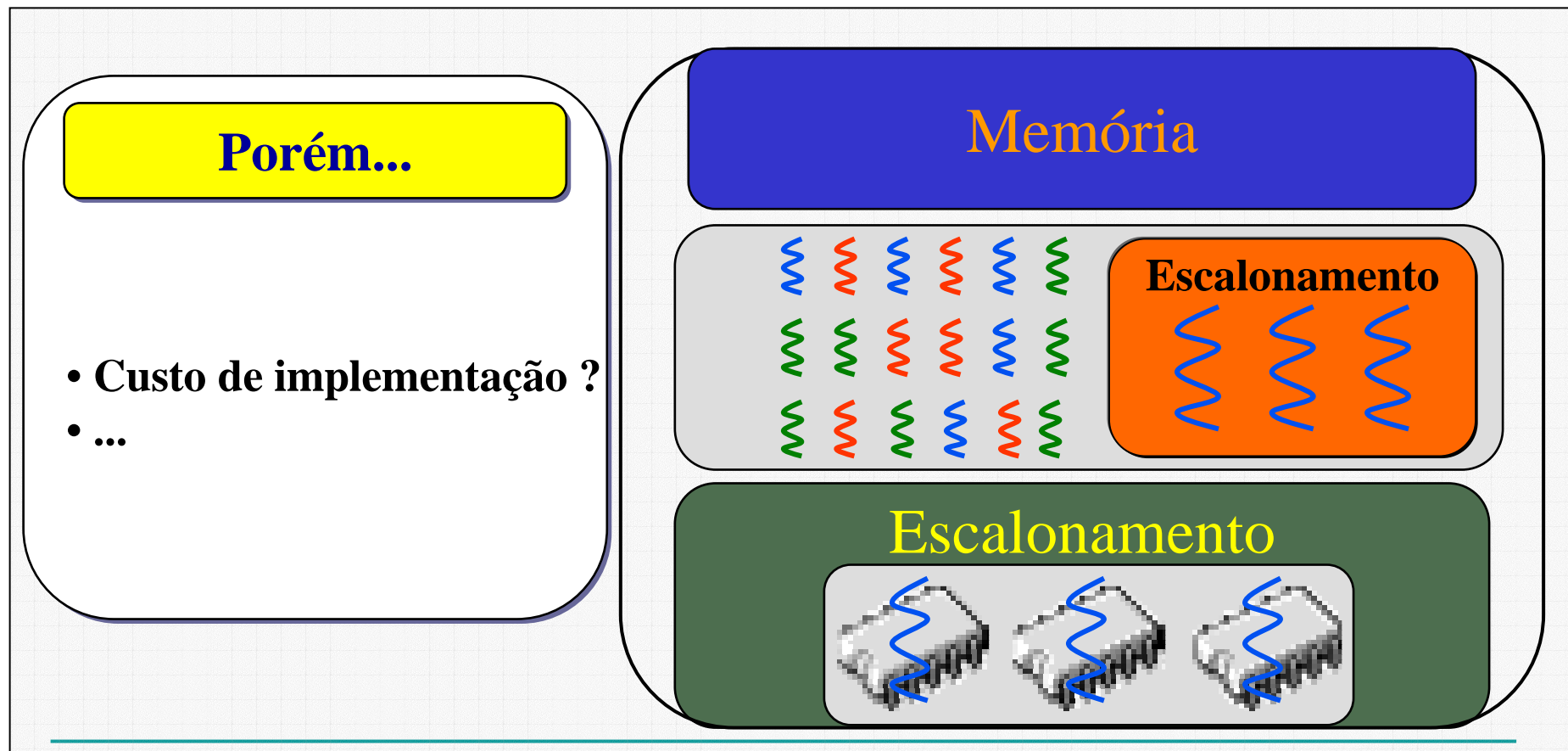


### Escalonamento



# Thread

## Modelos de Threads - $N : M$



# Thread

- Erros comuns na programação
  - Considerar uma ordem de execução dos threads
    - **Erro:**
      - Introduzir no algoritmo uma premissa que considere a ordem de execução dos threads
    - **Consequência:**
      - A ordem de execução é dada pela heurística de escalonamento, sendo estabelecida em tempo de execução. Normalmente esta heurística também considera fatores que são externos ao programa. Qualquer premissa no algoritmo que considere a ordem de execução fatalmente irá produzir um programa incorreto
    - **Solução:**
      - Caso seja necessário definir uma ordem de execução para os threads devem ser utilizadas as primitivas específicas de sincronização disponibilizadas pela ferramenta utilizada

# Thread

- Erros comuns na programação

- ✓ Considerar uma ordem de execução dos threads
- Acesso simultâneo a um dado por dois (ou mais) threads
  - *Data-race*
  - **Erro:**
    - Não controlar o acesso ao espaço de endereçamento compartilhado pelos threads
  - **Consequência:**
    - Incoerência do estado da execução pela manipulação simultânea e não controlada dos dados
  - **Solução:**
    - Identificar os dados passíveis de acesso simultâneo por dois (ou mais) threads e proteger sua manipulação pelo uso de mecanismos de sincronização

# Thread

- Erros comuns na programação
  - ✓ Considerar uma ordem de execução dos threads
  - ✓ Acesso simultâneo a um dado por dois (ou mais) threads
  - Postergação indefinida
    - *Deadlock*
    - **Erro:**
      - Não sinalizar uma condição de saída de uma sincronização
    - **Consequência:**
      - Um (ou mais) thread(s) permanecem bloqueados aguardando uma sincronização
    - **Solução:**
      - Garantir que todas as condições de saída de sincronizações sejam efetuadas

# Caso de estudo: OpenMP

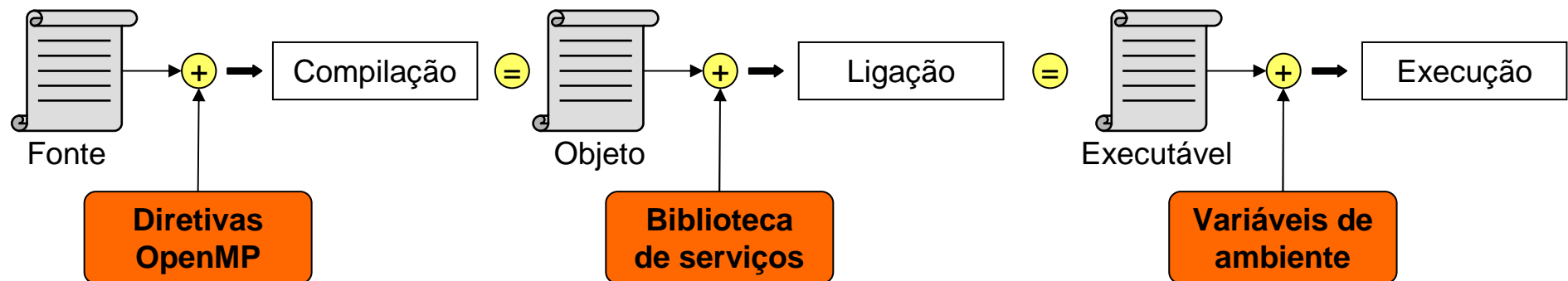
- Paralelismo explícito
  - O programador deve introduzir diretivas para gerar e controlar a execução paralela do programa
- Diferentes níveis de abstração
  - Paralelismo de dados
  - Paralelismo de tarefa
- **OpenMP**
  - Diretivas de compilação / Biblioteca de serviços
  - Paralelismo aninhado



# Sumário

- ✓ Introdução
- ✓ Arquiteturas multi-core
- ✓ Programação multithread
- ▶ **OpenMP**
  - Prática de programação
  - Considerações finais

- Interface de programação aplicativa (API) para desenvolvimento de aplicações multithread em C/C++ e Fortran
  - Define
    - Diretivas de compilação
    - Biblioteca de serviços
    - Variáveis de ambiente



# OpenMP

- Interface de programação aplicativa (API) para desenvolvimento de aplicações multithread em C/C++ e Fortran
  - Define
    - Diretivas de compilação
    - Biblioteca de serviços
    - Variáveis de ambiente
  - Busca estabelecer um padrão
    - AINSI X3H5 como primeira tentativa
  - Esforço iniciado de um grupo de fabricantes de hardware e desenvolvedores de software (Fujitsu, HP, IBM, Intel, Sun, DoE ASC, Compunity, EPCC, KSL, NASA Ames, NEC, SGI, ST Micro/PG, ST Micro/Portland Group, entre outros)

# OpenMP

- API para C/C++

- Variáveis de ambiente

`OMP_NOME`

- Diretivas de compilação

`#pragma omp diretiva [cláusula]`

- Biblioteca de serviços

`omp_serviço( ... );`

# OpenMP

## ■ API para C/C++

### □ Variáveis de ambiente

**OMP\_***NOME*

- Identifica o número de atividades que serão executadas em paralelo

**OMP\_NUM\_THREADS** (default: número de processadores)

- Indica se o número de atividades a serem executadas em paralelo deve ou não ser ajustado dinamicamente

**OMP\_DYNAMIC** (default: FALSE)

- Indica se deve ser contemplado ativação de paralelismo aninhado

**OMP\_NESTED** (default: FALSE)

- Define esquema de escalonamento das atividades paralelas

**OMP\_SCHEDULE** (default: estático)

# OpenMP

- API para C/C++

- Diretivas

- Formato:

#pragma omp diretiva [cláusula]

Opcionais aceitos pela diretiva OpenMP

Determinação da diretiva OpenMP

Sentinela de uma diretiva OpenMP (C/C++)

- Exemplo:

#pragma omp **parallel** default(shared) private(x,y)

# OpenMP

- API para C/C++

- Biblioteca de serviços

- Permitem controlar e interagir com o ambiente de execução.

- Formato:

- `retorno omp_serviço ( [parâmetros] );`

- Exemplos:

- `void omp_set_num_threads( 10 );`

- `int omp_get_num_threads();`

- `void omp_set_nested( 1 );`

- `int omp_get_num_procs ();`

# OpenMP

## ■ Modelo M:N

- ❑ Devem ser indicadas as atividades concorrentes da aplicação

Regiões paralelas

- ❑ Deve ser informado o paralelismo da arquitetura

Time

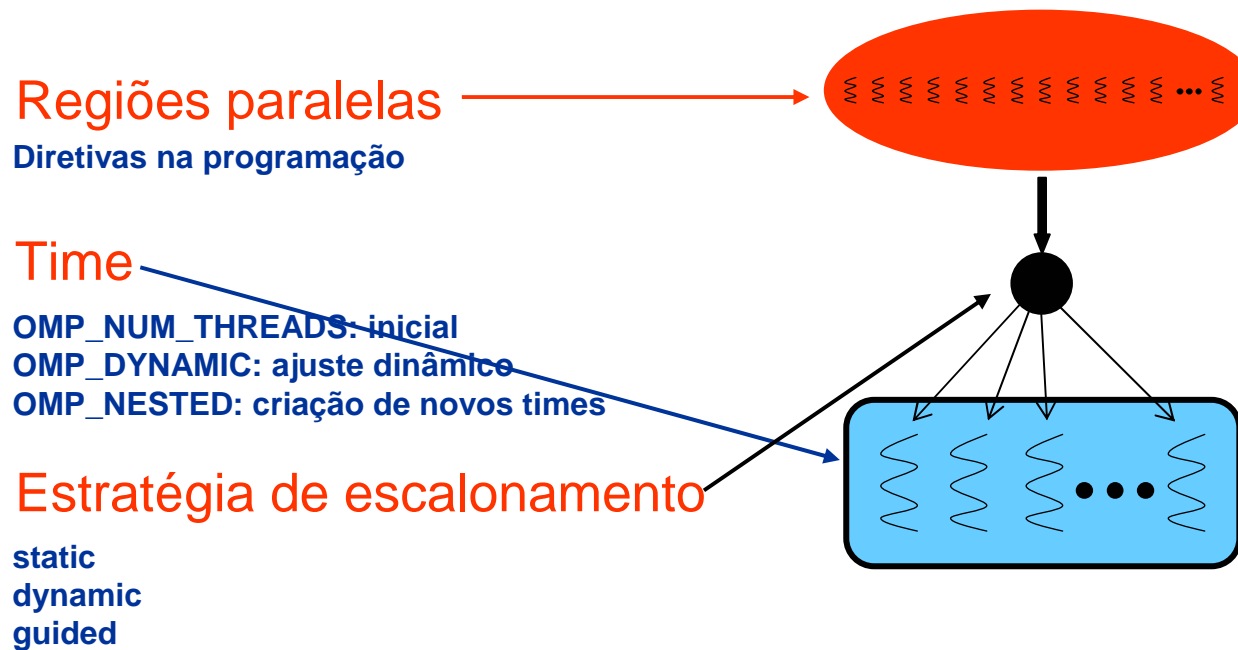
- ❑ Deve ser informado o esquema de escalonamento das atividades paralelas nos threads que compõem o time

Estratégia de escalonamento



# OpenMP

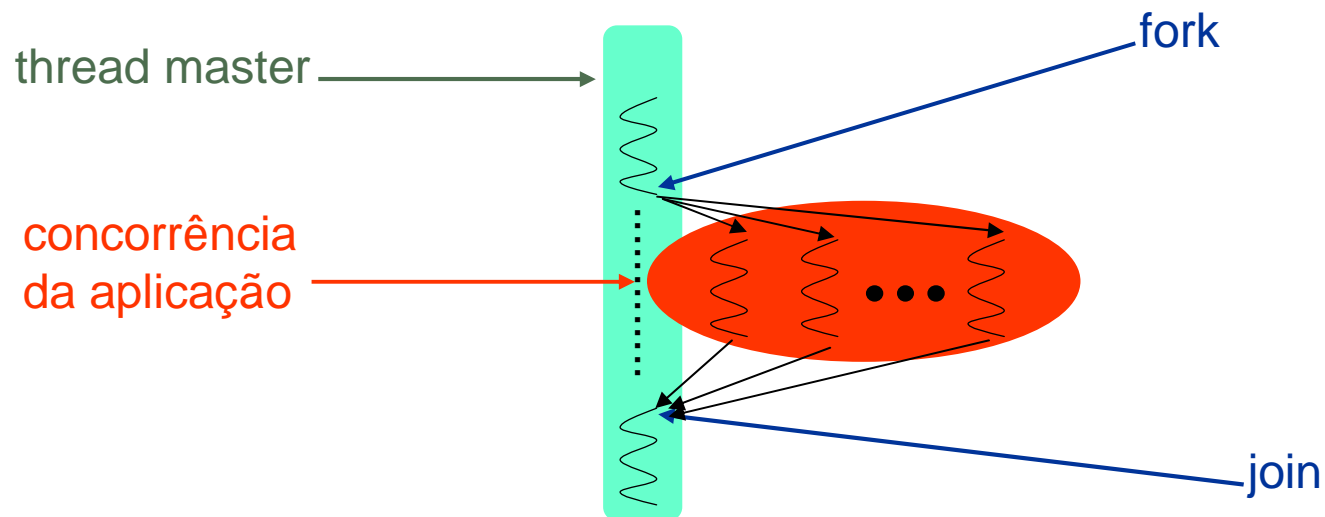
## ■ Modelo M:N



# OpenMP

## ■ Modelo básico de execução

### □ Fork / Join

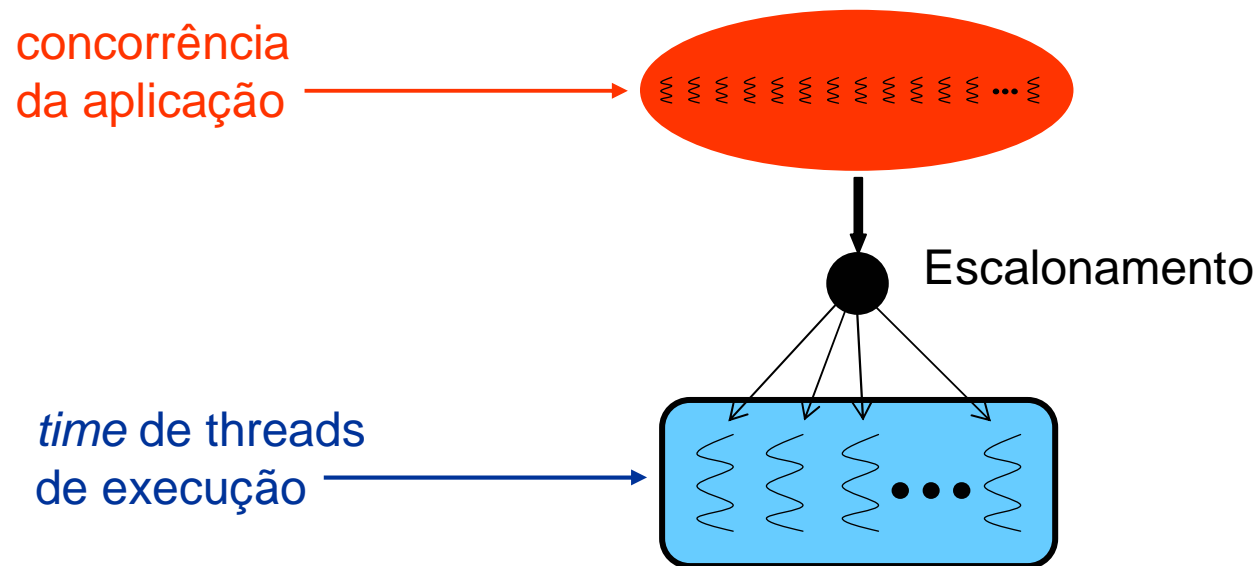


### □ O programador descreve a concorrência de sua aplicação

# OpenMP

## ■ Modelo básico de execução

- ❑ Fork / Join
- ❑ O programador descreve a concorrência de sua aplicação
- ❑ A execução se dá por um *time* de threads



# OpenMP

## ■ Primeiro programa

```
#include <omp.h>
main ( ) {
```

```
    // Código seqüencial (master)
```

```
    #pragma omp_parallel
```

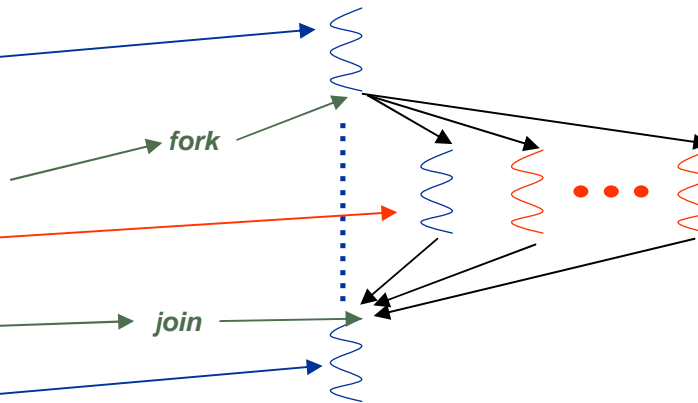
```
    { // Início do código concorrente
```

```
        // Código paralelo
```

```
    } // Fim do código concorrente
```

```
    // Código seqüencial (master)
```

```
}
```



*São executadas tantas instâncias do código paralelo quanto forem o número de threads no time.*

# OpenMP

- Primeiro programa
- Primeira sessão (Linux)

```
tcsh

$> icpc -openmp primeiro.c
$> setenv OMP_NUM_THREADS 4
$> ./a.out
$>
```

# OpenMP

## ■ Programa mais elaborado

```
#include <omp.h>
```

```
main () {
```

```
int nthreads, tid;
```

```
#pragma omp parallel private(tid)
```

```
{
```

```
    tid = omp_get_thread_num();
```

```
    printf("Oi mundo, sou o thread %d\n", tid);
```

```
    if( tid == 0 ) {
```

```
        nthreads = omp_get_num_threads();
```

```
        printf("Total de threads: %d\n", nthreads);
```

```
    }
```

```
}
```

### Lendo o programa:

- Dados globais
- Fork
- Join
- Cópia de dados privado ao thread
- Seção paralela
- Identificação do thread *master*

# OpenMP

## ■ Programa mais elaborado

```
#include <omp.h>

main () {
    int nthreads, tid;

    #pragma omp parallel private(tid)
    {
        tid = omp_get_thread_num();
        printf("Oi mundo, sou o thread %d\n", tid);

        if( tid == 0 ) {
            nthreads = omp_get_num_threads();
            printf("Total de threads: %d\n", nthreads);
        }
    }
}
```

```
tcsh

$> setenv OMP_NUM_THREADS 4
$> ./a.out
Oi mundo, sou o thread 3
Oi mundo, sou o thread 0
Total de Threads: 4
Oi mundo, sou o thread 1
Oi mundo, sou o thread 2
$>
$> setenv OMP_NUM_THREADS 5
$> ./a.out
Oi mundo, sou o thread 0
Oi mundo, sou o thread 1
Oi mundo, sou o thread 4
Total de Threads: 5
Oi mundo, sou o thread 3
Oi mundo, sou o thread 2
```

# OpenMP

## ■ Programação

### □ Diretiva: **sections**

*NO WAIT*

```
main () {  
  int x;
```

→ Dado compartilhado

```
  #pragma omp sections
```

→ Região paralela

```
{  
  #pragma omp section  
  {  
    foo(x);  
  }  
}
```

```
{  
  #pragma omp section  
  {  
    bar(x);  
  }  
}
```

```
}
```



# OpenMP

- Programação
  - Diretiva: **sections**
  - **Cláusulas aceitas**
    - **private**
    - **firstprivate**
    - **lastprivate**
    - **reduction**
    - **nowait**

# OpenMP

## ■ Programação

### ■ Diretiva: **parallel** Cláusula **private**

```
main () {  
  int x, y;
```

Dado compartilhado

```
#pragma omp parallel private(y)
```

Região paralela

```
{  
  {  
    y = x + 1;  
  }  
}
```

Número de instâncias

==

Número de threads no *time*

Cada execução manipula uma instância do dado.  
**As cópias locais não são inicializadas!**

# OpenMP

## ■ Programação

- Diretiva: **parallel** Cláusula **private**

- **Cláusulas alternativas:**

- **firstprivate**

- Cada cópia local é inicializada com o valor que o dado possui no escopo do thread master

- **lastprivate**

- O dado no escopo do thread master é atualizado com o valor da última avaliação da região paralela

- Exemplo

```
#pragma omp parallel lastprivate(y) firstprivate(y)
```

Pode ser *last* e *first* ao mesmo tempo!

# OpenMP

## ■ Programação

- Diretiva: **parallel** Serviço **omp\_set\_num\_threads**

```
main () {  
    int x, y;
```

```
    omp_set_num_threads( 5 );
```

```
    #pragma omp parallel private(y)  
    {  
        y = x + 1;  
    }  
}
```

Novo número de threads  
no *time*

# OpenMP


## ■ Programação

### ■ Diretiva: **parallel** Cláusula **reduction**

- Cada execução pode manipular sua cópia, como em `private`
- O valor local inicial é definido pela operação de *redução*
- No final uma operação de *redução* atualiza o dado no thread master

#### Exemplo:

```
int soma = 100;  
  
omp_set_num_threads( 4 );  
#pragma omp parallel reduction( + : soma )  
{  
    soma += 1;  
}  
// No retorno ao master: soma = 104
```



# OpenMP

## ■ Programação

### ■ Diretiva: **parallel** Cláusula **reduction**

- Cada execução pode manipular sua cópia, como em `private`
- O valor local inicial é definido pela operação de **redução**
- No final uma operação de **redução** atualiza o dado no thread master

Operações  
de redução:

Operador	Operação	Valor inicial
+	Soma	0
-	Subtração	0
*	Multiplicação	1
&	E aritmético	-1
&&	E lógico	1
	OU aritmético	0
	OU lógico	0
^	XOR aritmético	0

# OpenMP

## ■ Programação

### □ Diretiva: **parallel**

#### ■ Cláusulas aceitas

- **reduction**
- **private**
- **firstprivate**
- **shared**
- **default**
- **copyin**
- **if**
- **num\_threads**

# OpenMP

## ■ Programação

### □ Diretiva: **for** / **parallel for**

- Permite que os grupos de instruções definidas em um *loop* sejam paralelizadas
  - Note: **for** e **parallel for** não é exatamente igual
    - **for**: restringe o número de cláusulas aceitas
    - **parallel for**: aceita todas as cláusulas do **for** e do **parallel**



# OpenMP

## ■ Programação

### ■ Diretiva: **for** / **parallel for**

- Permite que os grupos de instruções definidas em um *loop* sejam paralelizadas
- Exemplo:

```
int vet[100], i, somatorio = 0;

#pragma omp parallel for private(i)
for( i = 0 ; i < 100 ; i++ ) vet[i] = i;

#pragma omp parallel for private(i) reduction(+:somatorio)
for( i = 0 ; i < 100 ; i++ ) somatorio += vet[i];

printf( "Somatorio: %d\n", somatorio );
```

# OpenMP

## ■ Programação

### ■ Diretiva: **for** / **parallel for**

- Permite que os grupos de instruções definidas em um *loop* sejam paralelizadas
- Exemplo:

```
int mat[TAM][TAM], aux[TAM][TAM], i, j;  
inicializa(mat);  
copia(mat, aux);  
#pragma omp parallel for  
  for( i = 0 ; i < TAM ; i++ )  
    #pragma omp parallel for  
      for( j = 0 ; j < TAM ; j++ )  
        mat[i][j] = func(aux, i, j, ...);  
copia(mat,aux);
```

# OpenMP

## ■ Programação

### □ Diretiva: **for** / **parallel for**

- Permite que os grupos de instruções definidas em um *loop* sejam paralelizadas
- Regras:

```
#pragma omp parallel for private(i)
for( i = 0 ; i < 100 ; i++ ) vet[i] = i;
```

- A variável de iteração é tornada local implicitamente no momento em que um loop é paralelizado
- A variável de iteração e o valor de teste necessitam ser inteiros e com sinal
- A variável de iteração deve ser incrementada ou decrementada, não são permitidas outras operações
- Os testes são: <, >, <=, >=
- Não é permitido: `for( i = 0 ; i < 100 ; vet[i++] = 0 );`

# OpenMP

## ■ Programação

### ■ Diretiva: **critical** [ (*nome*) ]

- Permite que trechos de código sejam executados em regime de exclusão mútua
- É possível associar *nomes* aos trechos

```
int prox_x, prox_y;  
#pragma omp parallel shared(x, y) private(prox_x, prox_y)  
{  
    #pragma omp critical(eixox)  
    prox_x = dequeue( x );  
    trataFoo( prox_x, x );  
  
    #pragma omp critical(eixoy)  
    prox_y = dequeue( y );  
    trataFoo( prox_y, y );  
}
```

# OpenMP

## ■ Programação

### ■ Diretiva: **single** [**nowait**]

- Indica, em uma região paralela, um trecho de código que deve ser executado apenas por um único thread
- Representa uma barreira
- Com **nowait** a barreira pode ser relaxada

```
#pragma omp parallel
{
    #pragma omp single
    printf("Serão %d threads\n", omp_get_num_threads() );

    foo();

    #pragma omp single nowait
    printf("0 threads %d terminou\n", omp_get_thread_num() );
}
```

# OpenMP

## ■ Programação

### ■ Diretiva: **master** [**nowait**]

- Indica, em uma região paralela, um trecho de código que deve ser executado apenas thread master
- Representa uma barreira
- Com **nowait** a barreira pode ser relaxada

```
#pragma omp parallel
{
    #pragma omp master
    printf("Serão %d threads\n", omp_get_num_threads() );

    foo();

    #pragma omp master nowait
    printf("O thread master terminou\n" );
}
```

# OpenMP

## ■ Programação

### □ Diretiva: **flush** [ (**dados**) ]

- Força a atualização dos dados compartilhados entre os threads em memória
- Representa uma barreira, garantindo que todas as operações realizadas por todos os threads até o momento do *flush* foram realizadas

# OpenMP

## ■ Programação

### □ Cláusula: **schedule(static | dynamic | guided)**

- Indica qual estratégia deve ser utilizada para distribuir trabalho entre os threads do time

#### □ **static**

- Implica na divisão do trabalho por igual entre cada thread
- Indicado quando a quantidade de trabalho em cada grupo de instruções (como em cada iteração em um `for`) é a a mesma

#### □ **dynamic**

- Cada thread busca uma nova quantidade de trabalho quando terminar uma etapa de processamento
- Indicado quando a quantidade de trabalho em cada grupo de instruções (como em cada iteração em um `for`) não for a mesma

#### □ **guided**

- Semelhante ao `dynamic`, mas com divisão não uniforme na divisão das tarefas
- Indicado quando existe execução assíncrona (cláusula `nowait` anterior)



# Sumário

- ✓ Introdução
- ✓ Arquiteturas multi-core
- ✓ Programação multithread
- ✓ Ferramentas de programação
- ✓ **Prática de programação**
- Considerações finais

# Modelagem da concorrência

## ■ Na implementação

- ❑ Traduzir as atividades concorrentes em termos de unidades de execução previstas pela ferramenta utilizada
  - Preocupar-se com a granularidade de execução
    - ❑ Relação entre a quantidade de cálculo efetuada a cada atividade independente e o número de interações entre estas atividades
    - ❑ Reflete o *overhead* de execução da implementação:
      - Quanto mais freqüente forem as interações, mais o ambiente de execução deverá se fazer presente, ocupando tempo de processamento

# Modelagem da concorrência

- Na implementação

- Tratar a questão da portabilidade

- **de código**

- Envolve a seleção da ferramenta através de critérios que facilitem a reutilização do código em diferentes ambientes

- **OpenMP**: busca por um padrão de programação simples e abrangente. Processo acompanhado por várias empresas de software e hardware. Disponibilizado em diversos compiladores

# Modelagem da concorrência

## ■ Na implementação

### □ Tratar a questão da portabilidade

✓ de código

### ■ de desempenho

- Relacionado com a escalabilidade da implementação realizada
- Deve ser considerada a grande variedade de arquiteturas paralelas disponíveis e a expectativa da ainda maior variedade de número de cores em processadores multi-core
- Dissociação da descrição da concorrência da aplicação da implementação paralela realizada
  - Modelos M:N nativos os introduzidos através de estratégias de programação

# Multithreading e bibliotecas de serviço

- Uso de bibliotecas de serviço em programas multithread

- **Thread safe**

- Garante a correção da execução dos serviços da biblioteca, pois sua implementação considera a possibilidade de que dois ou mais serviços podem estar ativos ao mesmo tempo

- **Thread aware**

- A biblioteca foi implementada de tal forma que não exista a possibilidade de alguma thread do programa aplicativo ser interrompida por necessidade de sincronização interna à biblioteca – ou, alternativamente, que a contenção seja a menor possível

# Hierarquia de memória

- Exploração da memória cache para minimizar o efeito do *Gargalo de von Neumann*
  - Alinhamento de dados em memória
    - Diz respeito a alocação de dados na memória
      - Um dado de  $n$  bytes é dito alinhado se  $n = 2^x$  e o endereço  $e$  de armazenamento seja  $n = k \times n$ 
        - Um dado `char` é alinhado em qualquer posição de memória
        - Um dado `int` é alinhado em um endereço múltiplo de 4
    - O alinhamento é especificado de forma a obter o melhor desempenho da arquitetura
      - Como utilização do barramento de dados
    - É possível alterar o comportamento padrão de alinhamento
      - O ônus é perder o desempenho ótimo da arquitetura
      - A aposta é reduzir a quantidade de memória utilizada, aumentando *virtualmente* a capacidade do cache

# Outros recursos

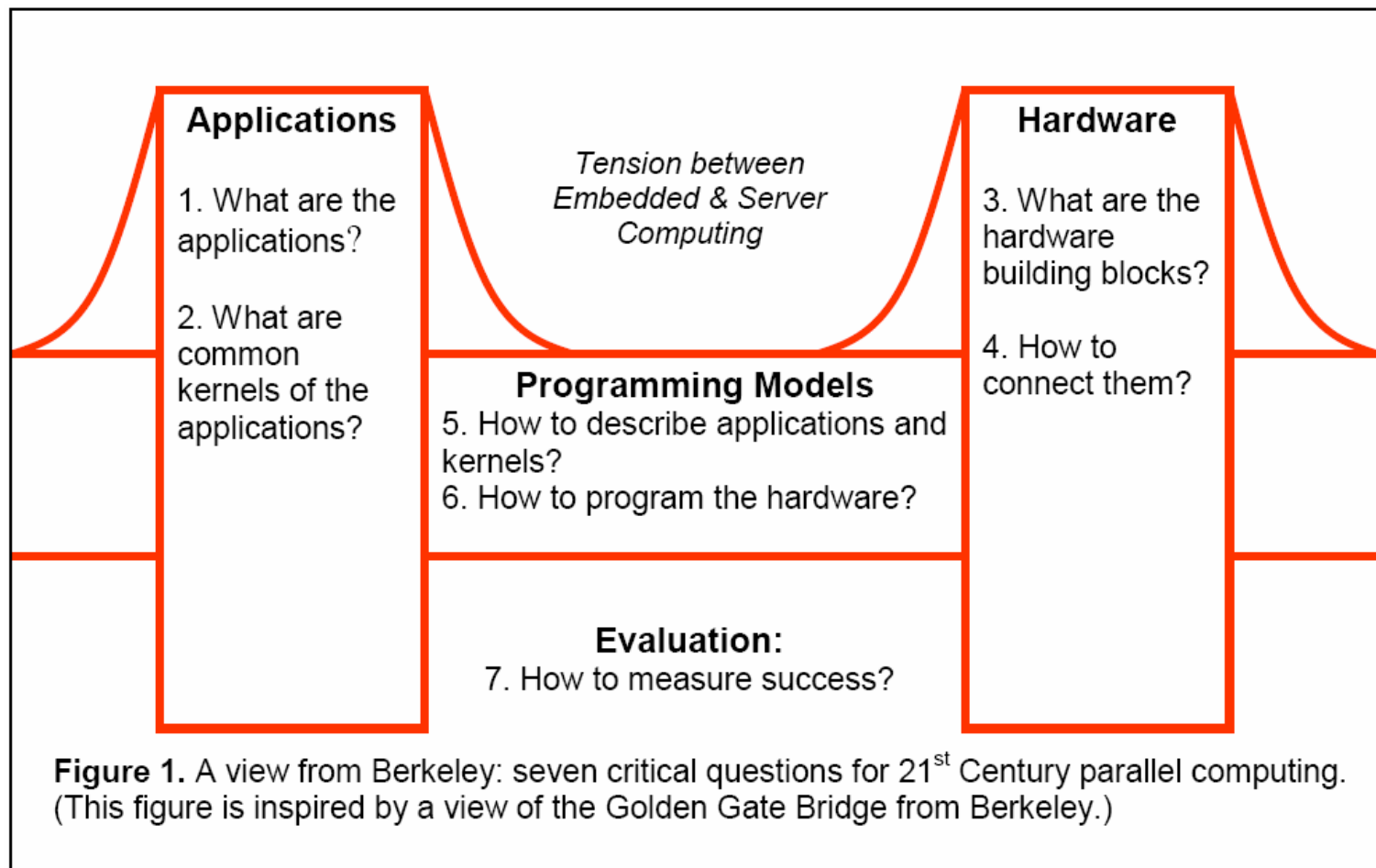
- Afinidade ao processador
  - *Processor affinity ( tid, mascara )*
- Frequência de processador
  - *Individual por core*
  - *Em escalas*
  - *Set frequency( core, frequencia)*

# Sumário

- ✓ Introdução
- ✓ Arquiteturas multi-core
- ✓ Programação multithread
- ✓ OpenMP
- ✓ Prática de programação
- ▶ **Considerações finais**



# Novas questões no desenvolvimento de software



Technical Report No. UCB/EECS-2006-183 <http://www.eecs.berkeley.edu/Pubs/TechRpts/2006/EECS-2006-183.html>

# Inovações tecnológicas

- A indústria de software sempre contou com o avanço da tecnologia CMOS para melhora de desempenho
- O software não tem acompanhado o avanço do hardware no que diz respeito a multi-core
  - Os efeitos em termos de desempenho da disponibilidade de vários cores ( $>4$ ) não serão percebidos pelo usuário sem uma modificação drástica no software desenvolvido
- O aumento de desempenho somente pode ser obtido com interferência direta dos recursos de programação utilizados

# Reação das empresas de hardware

- <http://sev.prnewswire.com/computer-electronics/20070626/NYTU01926062007-1.html>

NEW DELHI, India, **June 26, 2007** /PRNewswire/ -- Intel and NIIT today announced the launch of a Multi-Core training curriculum that has been developed jointly and will be deployed by NIIT globally. **As industry transitions from single-core processors to Multi-Core, software developers will need a new set of skills to design and develop software that harnesses the full potential of these processors.**

According to industry estimates, the global software developer population is expected to grow by 34% to 17.1 million, by 2009. NIIT, in association with its training partners, will offer the Intel Multi-Core training program to this developer community globally.

Speaking at the launch event, Mr. Scott Apeland, Director, Developer Network and Web Solutions, Intel Corporation said, **"The power of Multi-Core processors to increase software performance is incredible, but not automatic.** The full potential of Multi-Core processors is best unleashed when software is designed to take advantage of the power of multiple cores. This evolution of processing technologies has brought the software developer and architect community to a phase where they need to re-look at their existing skill-sets."

# Reação das empresas de hardware

- <http://www.hpcwire.com/hpc/1635195.html>

**DRESDEN, Germany, June 27, 2007 -- HP today introduced a program aimed at helping customers realize the full performance and cost benefits of using multi-core technology in high-performance computing (HPC) applications.**

The HPC market is where most disruptive technologies are created and evolved into broader commercial products. HP's new Multi-Core Optimization Program focuses on research for next-generation multi-core optimization technologies, as well as near-term developments and enhancements, in collaboration with key customers, technology partners and applications vendors.

"We're entering a phase where exploiting the power of multi-core processors is critical for customers to accelerate the innovation and discovery that is directly tied to their business outcomes," said Winston Prather, vice president and general manager, High Performance Computing, HP. "HP, a leader in the HPC market, is working with its partners to help customers capitalize on a new era of high-productivity computing."

"The emergence of multi-core technology allows for HPC technology to better help solve grand challenges; we've already seen an impact on large-scale numerical simulation in many fields of applications," said Prof. Dr. V. Heuveline, board member, Steinbuch Centre for Computing, KIT .

# Futuro

- Hoje arquiteturas multi-core são essencialmente SMPs e o número de cores relativamente baixo
- A expectativa é disponibilizar processadores com grande número de cores
  - 80 cores, segundo a Intel, em médio prazo
  - Existência de um número de cores tão grande que requer a utilização de um esquema de manipulação próximo a grades computacionais (G. Fox, CCGrid 2007, palestra)
- No futuro poderão (deverão) existir arquiteturas multi-core com cores especializados, tornando-se uma arquitetura heterogênea
- No Brasil, grupos ligados às áreas de processamento paralelo já questionam a necessidade de inserir disciplinas de programação concorrente nos primeiros semestres dos cursos de graduação

# E aqui?

- Diversos artigos técnicos: OpenMP e ...
  - Weather forecast
    - Google: 13.000
    - Scholar Google: 1.100
  - Forecast models
    - Google: 12.200
    - Scholar: 2.600
  - Modelos Meteorológicos
    - Google: 1.920
    - Scholar: 71

# Programação Paralela: Uma disciplina introdutória

*O paralelismo ao alcance de todos*

Prof. Dr. Gerson Geraldo H. Cavalheiro

*gerson.cavalheiro@inf.ufpel.edu.br*



Programa de Pós-Graduação em Computação  
Centro de Desenvolvimento Tecnológico  
Universidade Federal de Pelotas