INSTITUTO FEDERAL
SANTA CATARINA

Curso Superior de Tecnologia em
Análise e Desenvolvimento de Sistemas

Programação Concorrente e Distribuída

Professor:
Felipe Schneider Costa
felipe.costa@ifsc.edu.br

INSTITUTO FEDERAL
SANTA CATARINA

Tópicos

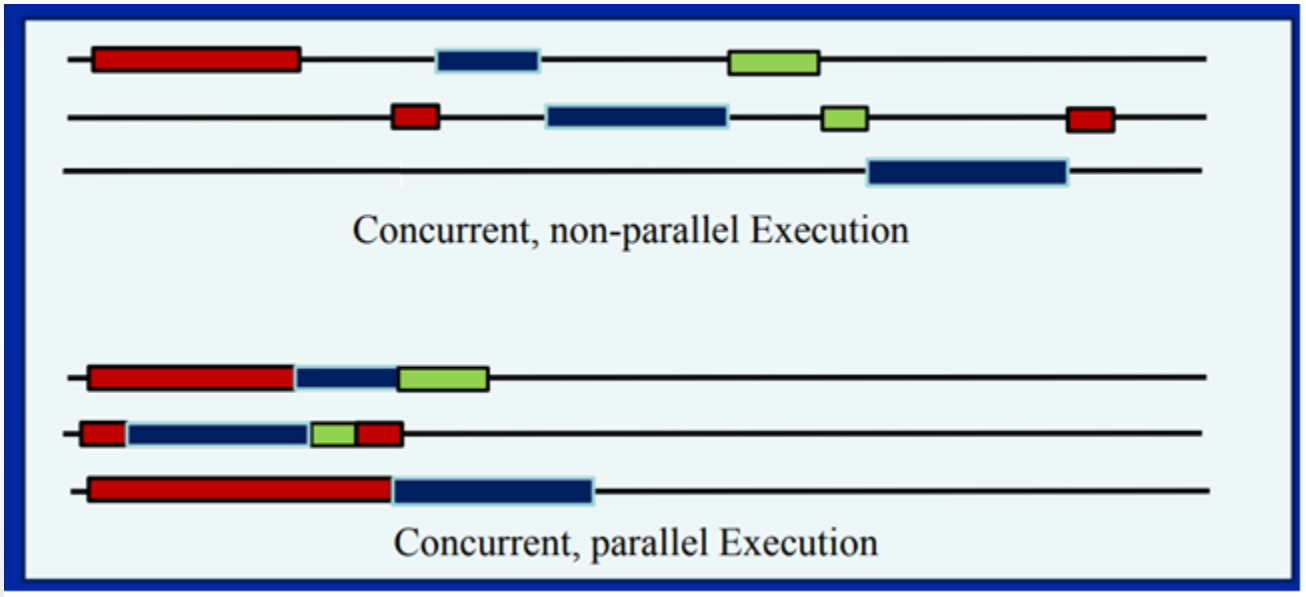
- Programação Paralela
- Atividade
- Exercícios

Programação Paralela

INSTITUTO FEDERAL
SANTA CATARINA

Concorrência x Paralelismo

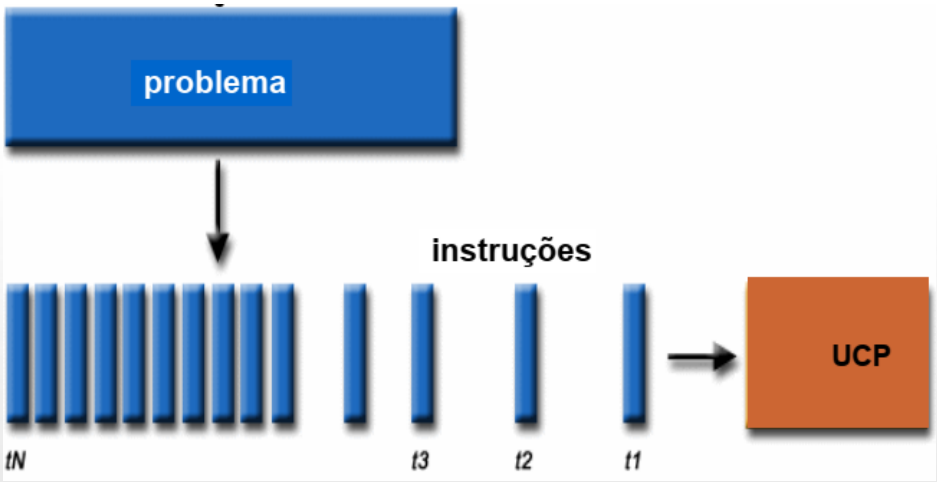
- Duas definições importantes:
  - ✓**Concorrência**: condição de um sistema em que várias tarefas estão logicamente ativas ao mesmo tempo.
  - ✓**Paralelismo**: condição de um sistema em que múltiplas tarefas estão realmente ativas ao mesmo tempo.



Programação Paralela

Programação sequencial

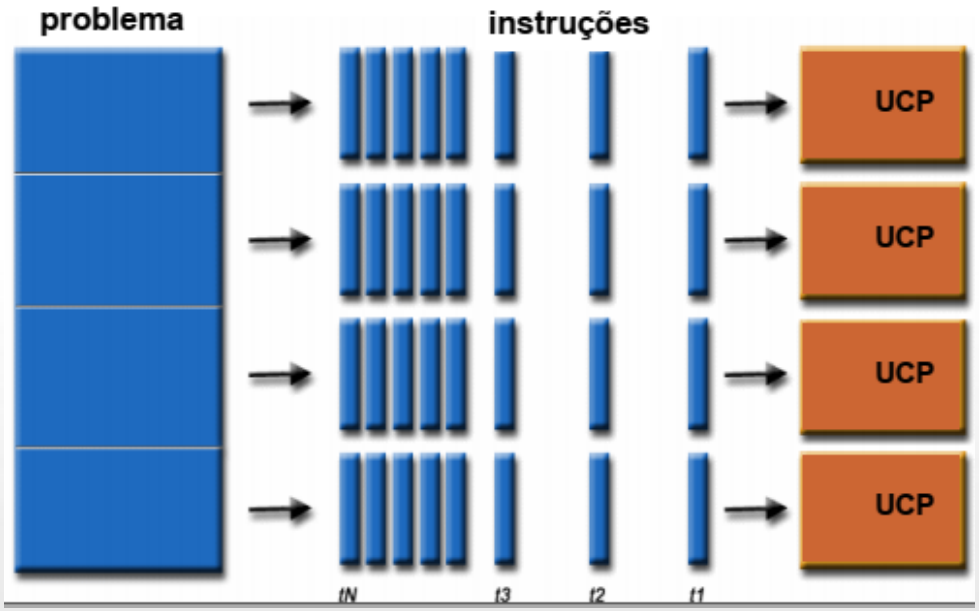
- O programa deve ser executado em um único computador com uma única Unidade Central de Processamento (UCP)
- Um programa é constituído de uma série de instruções que são executadas em sequência
- Somente uma instrução é executada de cada vez



Programação Paralela

Programação paralela

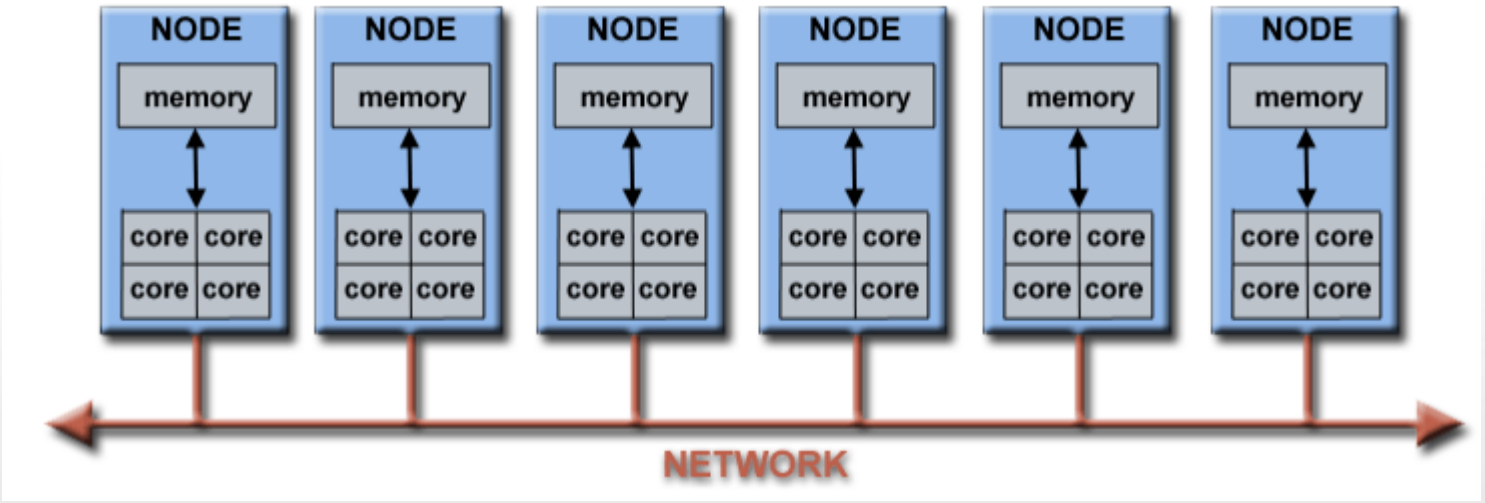
- O problema é decomposto em partes que podem ser executadas concorrentemente
- Cada parte é constituída de uma sequência de instruções que são executadas por UCPs diferentes simultaneamente



Programação Paralela

Computadores paralelos

- Os computadores hoje em dia são inerentemente paralelos:
  - ✓Múltiplas unidades funcionais (cache L1/L2, GPU, ...)
  - ✓Múltiplas unidades de execução (core's)
- Redes conectam computadores formando ***clusters***



INSTITUTO FEDERAL
SANTA CATARINA

Programação Paralela

- Características comuns dos problemas computacionais que podem ser resolvidos utilizando programação paralela:
  - ✓Podem ser decompostos em partes que podem ser resolvidas de forma simultânea.
  - ✓Necessitam executar muitas instruções ao mesmo tempo.
  - ✓São resolvidos em menor tempo utilizando múltiplas UCPs do que somente um.

Programação Paralela

Utilização de programação paralela

- Utilizada inicialmente para resolver problemas difíceis das áreas de ciência e engenharia:
  - ✓Meteorologia
  - ✓Física
  - ✓Biociências
  - ✓Geologia
  - ✓Microeletrônica
  - ✓Ciência da Computação

Programação Paralela

Utilização de programação paralela

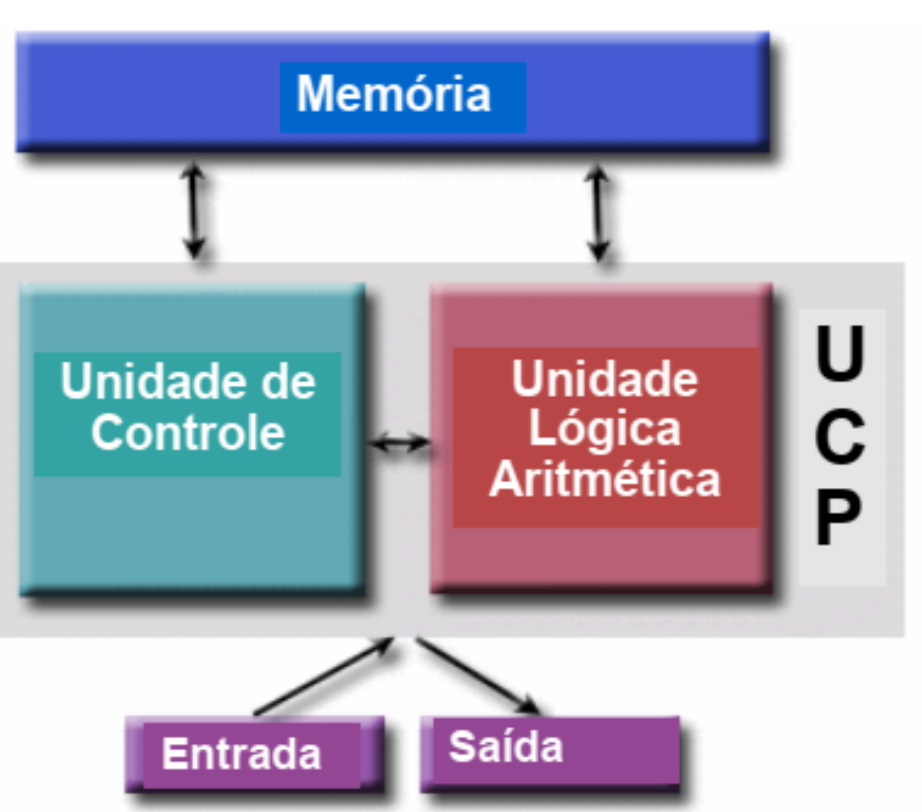
- Atualmente utilizada em aplicações comerciais para manipular grande quantidade de dados:
  - ✓Banco de dados, Mineração de dados
  - ✓Exploração de petróleo
  - ✓Busca na Web
  - ✓Computação gráfica e Realidade virtual
  - ✓Ambientes de trabalho cooperativos
  - ✓Tecnologia multimídia

Programação Paralela

Hardware

Programação Paralela

Arquitetura de von Neumann



Programação Paralela

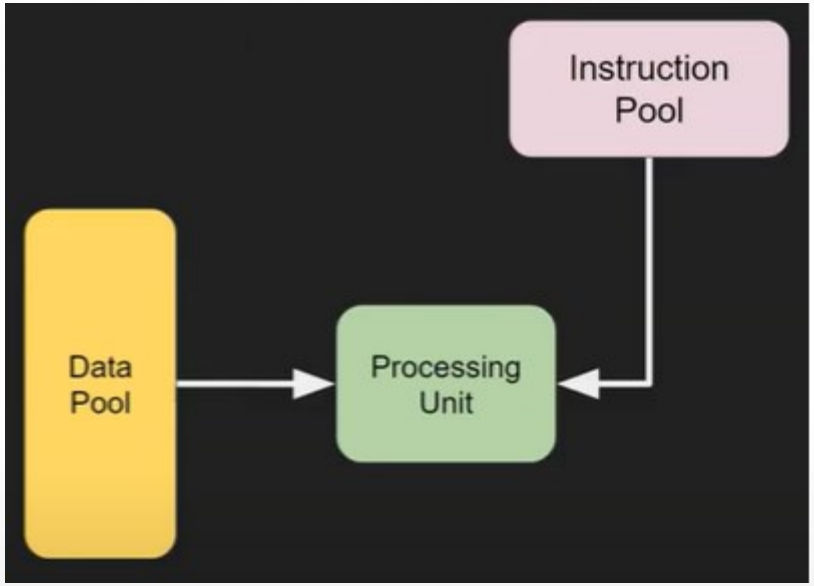
Classificação de Flynn para computadores paralelos

SISD Single Instruction, Single Data	**SIMD** Single Instruction, Multiple Data
MISD Multiple Instruction, Single Data	**MIMD** Multiple Instruction, Multiple Data

Programação Paralela

Single Instruction, Single Data (SISD)

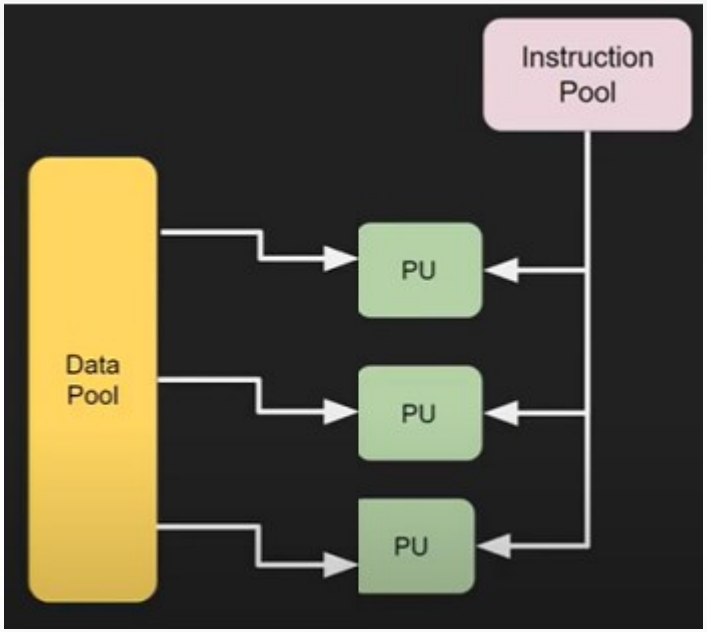
- Computador serial
- Tipo mais comum



Programação Paralela

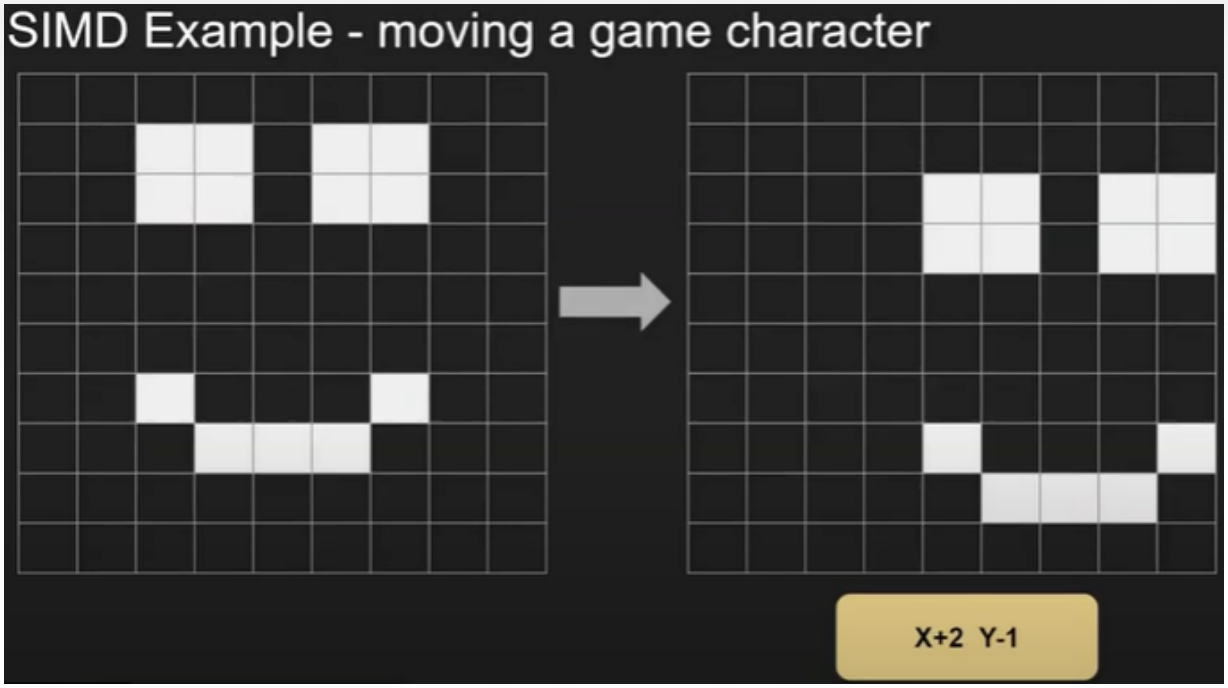
Single Instruction, Multiple Data (SIMD)

- Todas as UCPs executam as mesmas instruções sincronamente
- Cada UCP pode operar com diferentes dados
- GPUs utilizam processadores com esta arquitetura



Programação Paralela

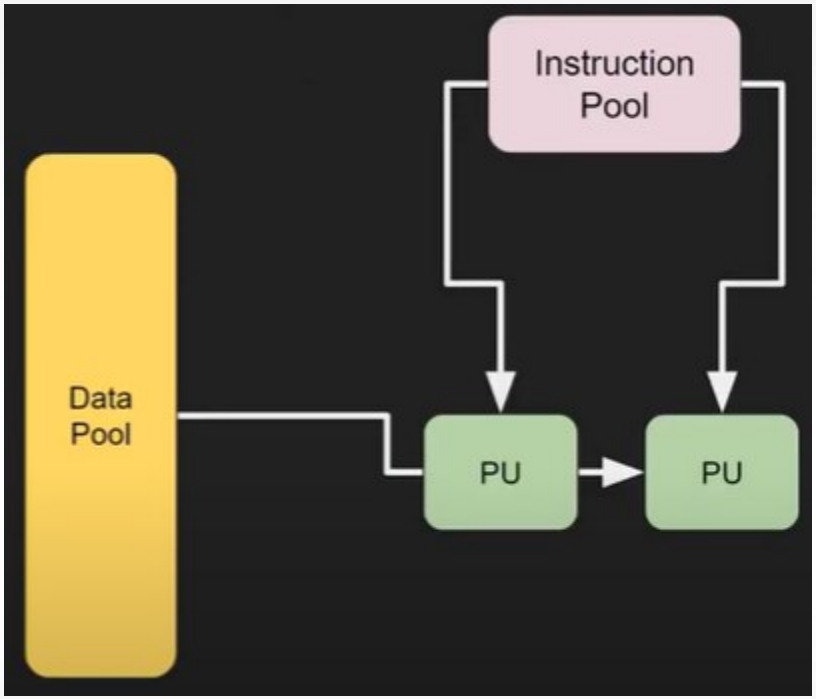
Single Instruction, Multiple Data (SIMD)



Programação Paralela

Multiple Instruction, Single Data (MISD)

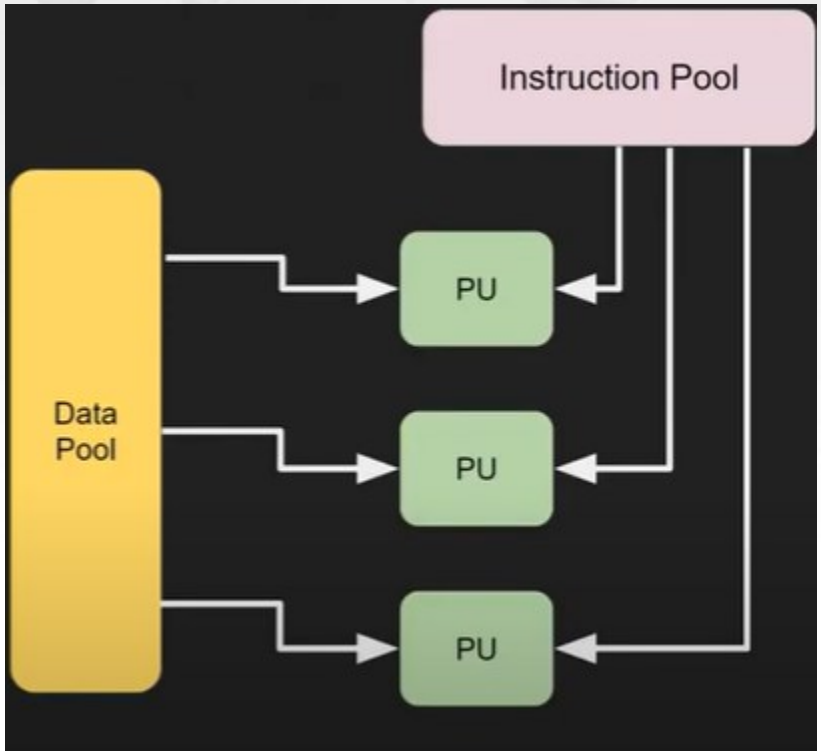
- Um único fluxo de dados é executado por várias UCPs com instruções diferentes
- Poucos exemplos reais de arquitetura deste tipo
- Exemplos:
  - ✓Tentar quebrar criptografia
  - ✓Aplicação de filtros em um sinal



Programação Paralela

Multiple Instruction, Multiple Data (MIMD)

- Tipo mais comum de computador paralelo
- Cada UCP pode executar instruções diferentes
- Cada UCP pode utilizar dados diferentes
- Super computadores, clusters, grids, multicore



INSTITUTO FEDERAL
SANTA CATARINA

Programação Paralela

- **Modelos de programação paralela** fornecem abstração da arquitetura do hardware e de memória.
- Os modelos de programação **não são** específicos para uma determinada arquitetura de computador.
- **Modelos de programação paralela:**
  - ✓Memória compartilhada
  - ✓Threads
  - ✓Passagem de mensagem
  - ✓Paralelismo de dados
  - ✓Modelos híbridos

INSTITUTO FEDERAL
SANTA CATARINA

Programação Paralela

- Qual modelo de programação paralela utilizar?
  - ✓Combinação de disponibilidade e escolha pessoal
  - ✓Qualidade da implementação do modelo é muito importante

Programação Paralela

Modelo de memória compartilhada

- Tarefas compartilham um espaço de endereçamento o qual pode ser lido e escrito assincronamente.
- Controle de acesso à memória realizado pelo programador.
- Não necessita de comunicação explícita entre as tarefas para compartilhar dados.

Programação Paralela

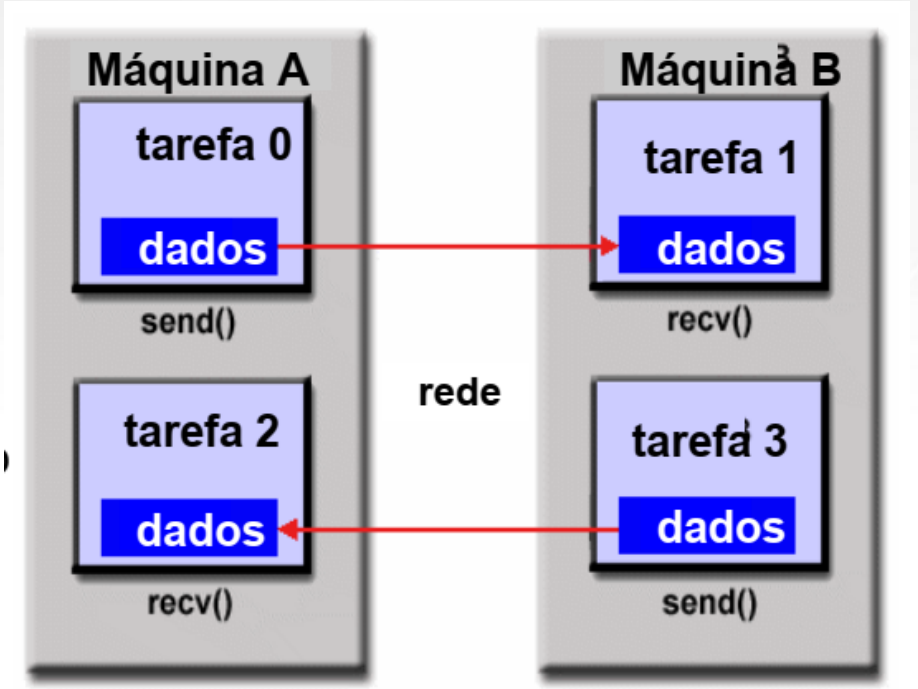
Modelo de threads

- Um único processo pode ter múltiplos e concorrentes caminhos
- Programa principal é escalonado para ser executado no computador.
- Programa principal executa serialmente e cria várias threads que podem ser executadas de forma concorrente.
- Threads possuem memória local e compartilham todos os recursos do Programa principal (memória global).

Programação Paralela

Passagem de mensagens

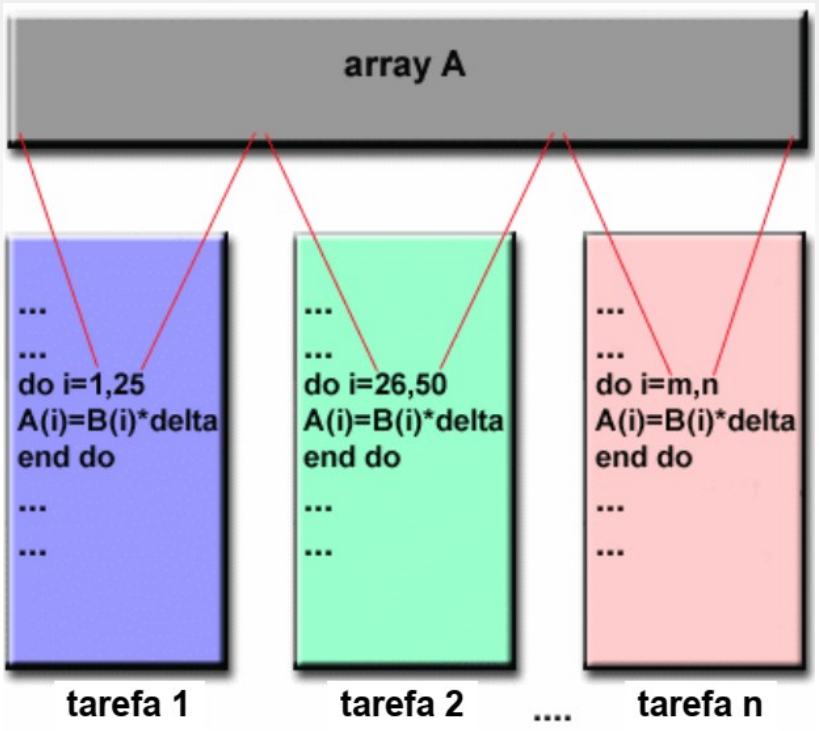
- Cada tarefa utiliza sua própria memória local
- Tarefas compartilham dados através da troca de mensagens
- Biblioteca de sub-rotinas que são inseridas no código
- Programador determina envio e recebimento de mensagens
- Padrão MPI



Programação Paralela

Paralelismo de dados

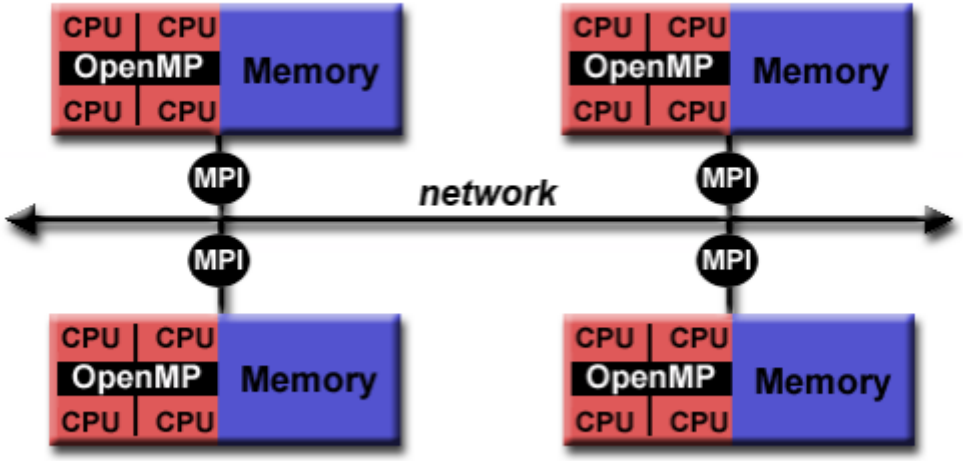
- Tarefas são executadas sobre o mesmo conjunto de dados organizados como um vetor ou cubo
- Tarefas trabalham sobre diferentes partes da mesma estrutura de dados
- Tarefas executam a mesma operação sobre dados diferentes
- Padrão OpenMP



Programação Paralela

Modelo híbrido

- Dois ou mais modelos de programação paralela são combinados
- Combinação de passagem de mensagens (MPI) com threads (OpenMP)



Programação Paralela

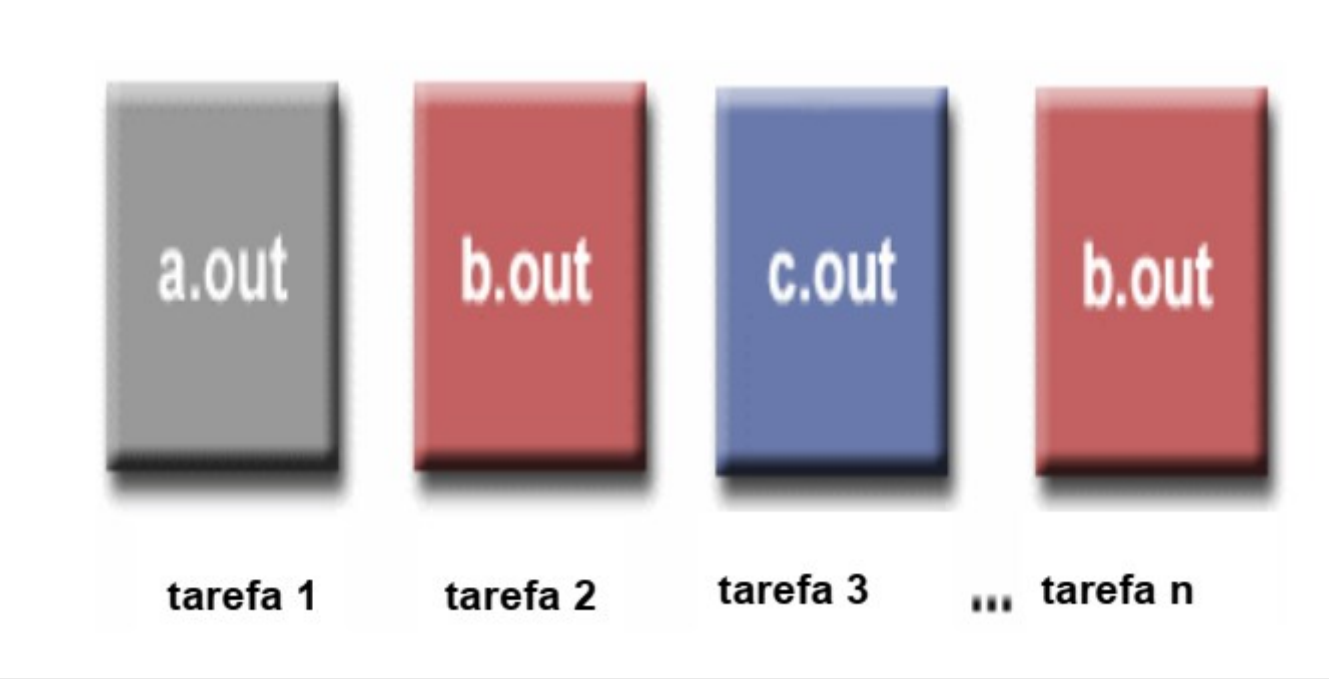
Single Program Multiple Data(SPMD)

- Um único programa executado por todas tarefas simultaneamente
- Tarefas podem estar executando instruções iguais ou diferentes de um mesmo programa
- Tarefas podem utilizar diferentes dados

Programação Paralela

Mulitple Program Multiple Data(MPMD)

- As tarefas podem estar executando programas iguais ou diferentes
- Todas as tarefas podem estar utilizando dados diferentes



Programação Paralela

Projeto de programas paralelos

- Paralelização automática ou manual
- Processo de paralelização realizado de forma manual
- Ferramentas para auxiliar o programador:
  - ✓Pré-processadores ou compiladores “paralelizadores”
  - ✓Compiladores
    - Totalmente automáticos
    - Compilador analisa o fonte de um programa sequencial e identifica possibilidades de paralelização.
    - Em geral “loops” são o alvo mais frequente de paralelização
  - ✓Diretivas de programação
    - O programador indica ao compilador o que paralelizar.

Programação Paralela

Problemas de paralelização automática

- Podem ser produzidos resultados errados.
- Queda de desempenho.
- Menor flexibilidade no desenvolvimento.
- Limitado a uma parte do código.
- Difícil de identificar paralelismo no código sequencial.
- Exemplo de um programa de implementação paralela difícil:
  - ✓Série de Fibonacci: F(k+2) = F(k+1) + F(k)

Programação Paralela

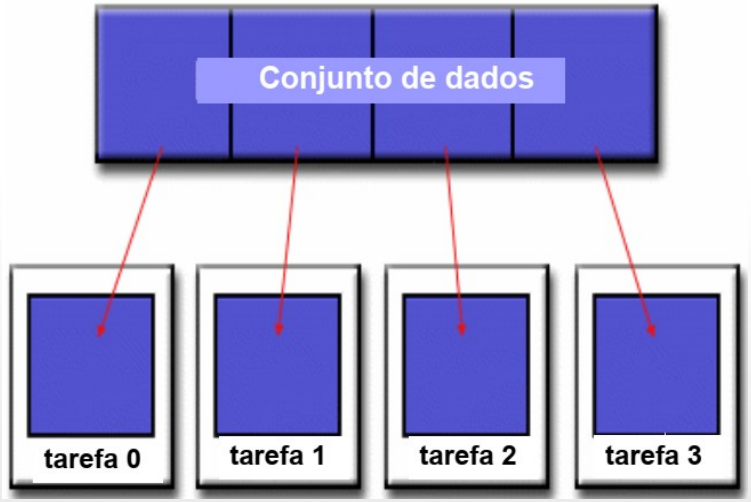
Entendimento do problema e do código sequencial

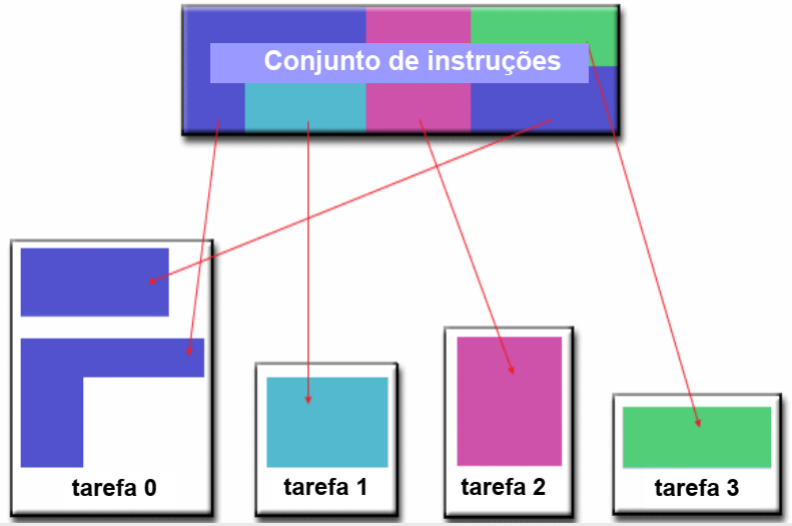
- Identificação das partes do programa que requerem muito processamento da UCP
- Identificação de gargalos do programa (Entrada/Saída)
- Identificação de inibidores de paralelismo (dependência de dados)
- Investigar diferentes algoritmos:
  - ✓Algoritmo utilizado para desenvolvimento sequencial pode não ser o melhor para desenvolvimento paralelo.

Programação Paralela

Particionamento

- Dividir o problema em partes discretas de trabalho que podem ser distribuídas a múltiplas tarefas
- Existem dois tipos básicos de particionamento:
  - ✓Particionamento de dados
  - ✓Particionamento funcional





Programação Paralela

Comunicação entre tarefas

- Aplicações que não necessitam de comunicação:
  - ✓Inverter cor de uma imagem preto e branco?
    - Basta inverter cada pixel independentemente.
    - São denominados embaraçosamente paralelos porque paralelização é trivial.
- Aplicações que necessitam de comunicação:
  - ✓Difusão de calor em uma superfície?
    - Uma tarefa necessita saber a temperatura dos seus vizinhos para calcular a temperatura da parte da superfície que ela está processando.

Programação Paralela

Fatores a considerar sobre a comunicação

- Custo de comunicação
  - ✓Comunicação entre tarefas implica em custo extra de processamento.
  - ✓Ciclos de máquina e recursos que poderiam ser utilizados no processamento do problema são utilizados para empacotar e desempacotar dados.
  - ✓Requer sincronização entre tarefas que pode levar as tarefas a ficarem esperando ao invés de estarem processando.
  - ✓Saturação da banda passante do meio de comunicação.

Programação Paralela

Fatores a considerar sobre a comunicação

- Latência *vs* Banda Passante
  - ✓Latência é o tempo gasto para enviar uma mensagem de tamanho mínimo (0 bytes) de um ponto A a B.
  - ✓Banda passante é a quantidade de dados que pode ser transmitida por unidade de tempo.
  - ✓O envio de muitas mensagens pequenas pode implicar em muito tempo gasto em latência.
  - ✓Geralmente é mais eficiente empacotar várias mensagens pequenas em uma grande.

Programação Paralela

Fatores a considerar sobre a comunicação

- Visibilidade da comunicação
  - ✓No modelo de passagem de mensagens a comunicação é explícita e visível e controlável pelo programador.
  - ✓No modelo de paralelismo de dados a comunicação é transparente para o programador.

Programação Paralela

Fatores a considerar sobre a comunicação

- Síncrona vs Assíncrona
  - ✓ **Síncrona:** requer algum tipo de protocolo de comunicação entre as tarefas que pode ser executado explicitamente pelo programador ou não.
    - Em geral são bloqueantes porque as tarefas necessitam que a comunicação termine para continuar o processamento.
  - ✓ **Assíncrona:** permite que as tarefas transfiram dados independentemente umas das outras.
    - Em geral são não bloqueantes porque as tarefas podem executar processamento enquanto a comunicação ocorre.

Programação Paralela e Distribuída

Processos

Programação Paralela e Distribuída

Introdução a *threads*

- ✓ **Programa:** conjunto de instruções em uma linguagem de alto nível ou de máquina.
- ✓ **Processo:** resulta da execução do programa, tem um ambiente de execução autocontido e privado.
- ✓ ***Thread***: fluxo de execução no interior de um processo.

Troca de contexto

- **Contexto de *thread*:** informações que o thread precisa para continuar a execução sem interrupções (conjunto de registros de CPU, pilha do thread, ...).

- **Contexto do processo:** todas as informações necessárias para que o S.O. possa restaurar a execução do processo a partir do ponto em que ele foi interrompido.

Troca de contexto

- **Observações**
  - ✓ *Threads* compartilham o mesmo espaço de endereçamento. A troca de contexto de *thread* pode ser feita totalmente independente do sistema operacional.
  - ✓ A troca de processos é geralmente (um pouco) mais cara, pois envolve colocar o sistema operacional no loop, ou seja, prender o kernel.
  - ✓ Criar e destruir *threads* é muito mais barato do que fazer isso para processos.

Programação Paralela e Distribuída

Por que usar *threads:* algumas razões simples

- **Evite bloqueios desnecessários**: um processo de *thread* único bloqueará ao fazer E/S; em um processo *multithread*, o sistema operacional pode alternar a CPU para outro *thread* nesse processo.
- **Explorar o paralelismo**: os threads em um processo *multithread* podem ser programados para serem executados em paralelo em um multiprocessador ou processador *multicore*.
- **Evite a troca de processos**: estruture grandes aplicativos não como uma coleção de processos, mas por meio de vários *threads*.

OpenMP

Histórico

- O OpenMP Architecture Review Board (ARB) publicou suas primeiras especificações da API do OpenMP em 1997.

- É possível utilizar o OpenMP em arquiteturas como:
  - ✓Arquiteturas com memória compartilhada centralizada
  - ✓Arquiteturas com memória compartilhada distribuída

- Na API OpenMP o programador especifica explicitamente as ações a serem tomadas pelo compilador e pelo sistema de tempo de execução para executar o programa em paralelo.

INSTITUTO FEDERAL
SANTA CATARINA

OpenMP

- O paralelismo no OpenMP é obtido pela execução simultânea de diversas *threads* dentro das regiões paralelas.

- Haverá ganho real de desempenho se houver processadores disponíveis na arquitetura para efetivamente executar essas regiões em paralelo.

- Por exemplo, as diversas iterações de um laço podem ser compartilhadas entre as diversas threads e, se não houver dependências de dados entre as iterações do laço, poderão também ser executadas em paralelo.

INSTITUTO FEDERAL
SANTA CATARINA

OpenMP

- Os laços são a principal fonte de paralelismo em muitas aplicações.
- Se as iterações de um laço são independentes (podem ser executadas em qualquer ordem), então podemos compartilhar as iterações entre threads diferentes.
- Por exemplo, se tivermos duas threads e o seguinte laço:

```
for (i = 0; i<100; i++)
    a[i] = a[i] + b[i];
```

- As iterações podem ser divididas entre as threads.
- **De que forma?**

OpenMP

Modelo de programação

- Paralelismo baseado em threads:
  - ✓Se baseia na existência de processos consistindo de várias threads
- Paralelismo explícito:
  - ✓Modelo de programação explícito e não automático, permitindo total controle da paralelização ao programador.

OpenMP

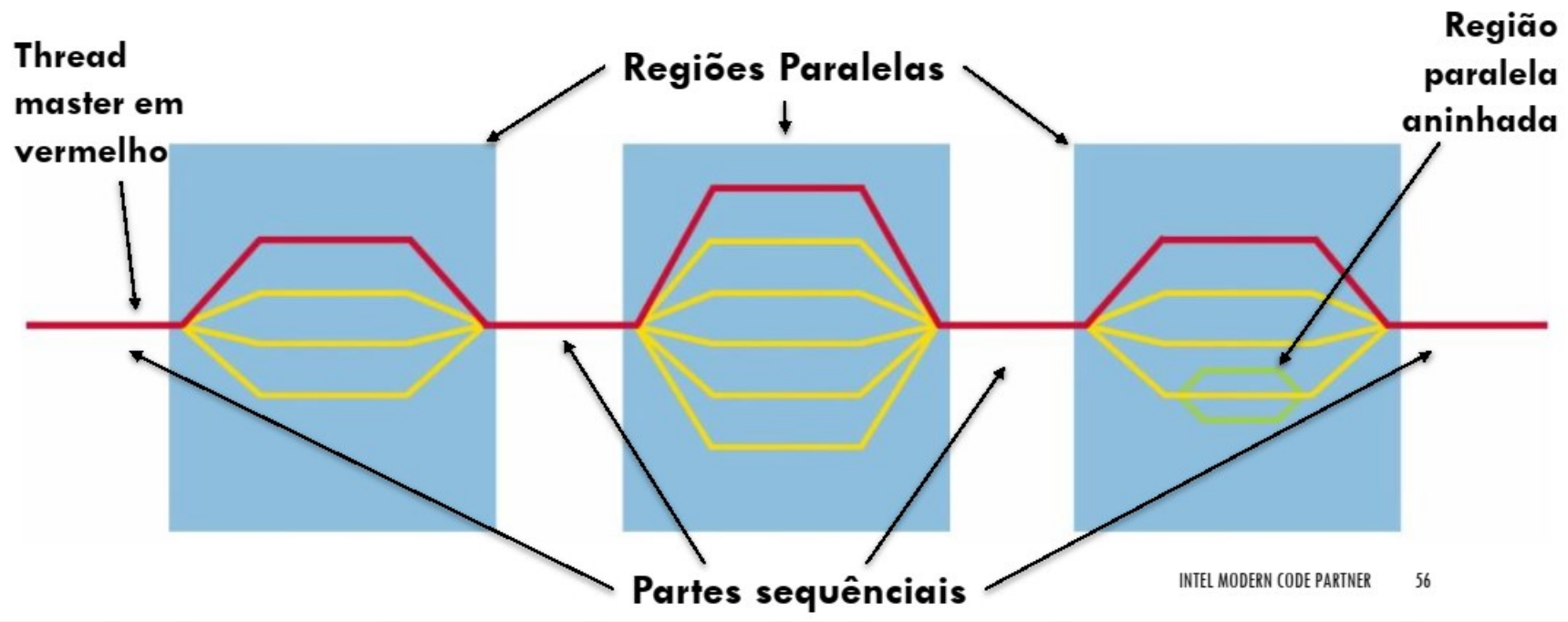
Modelo de programação (cont.)

- Modelo ***fork-join***
  - ✓Os programas OpenMP começam como um único processo denominado ***master thread***, que executa sequencialmente até encontrar a primeira construção para uma região paralela.
  - ✓**FORK**: a ***master thread*** cria um time de *threads* paralelos:
    - As instruções que estão dentro da construção da região paralela são executadas em paralelo pelas diversas threads do time.
  - ✓**JOIN**: quando as threads finalizam a execução das instruções dentro da região paralela, elas sincronizam e terminam, ficando somente ativa a ***master thread.***

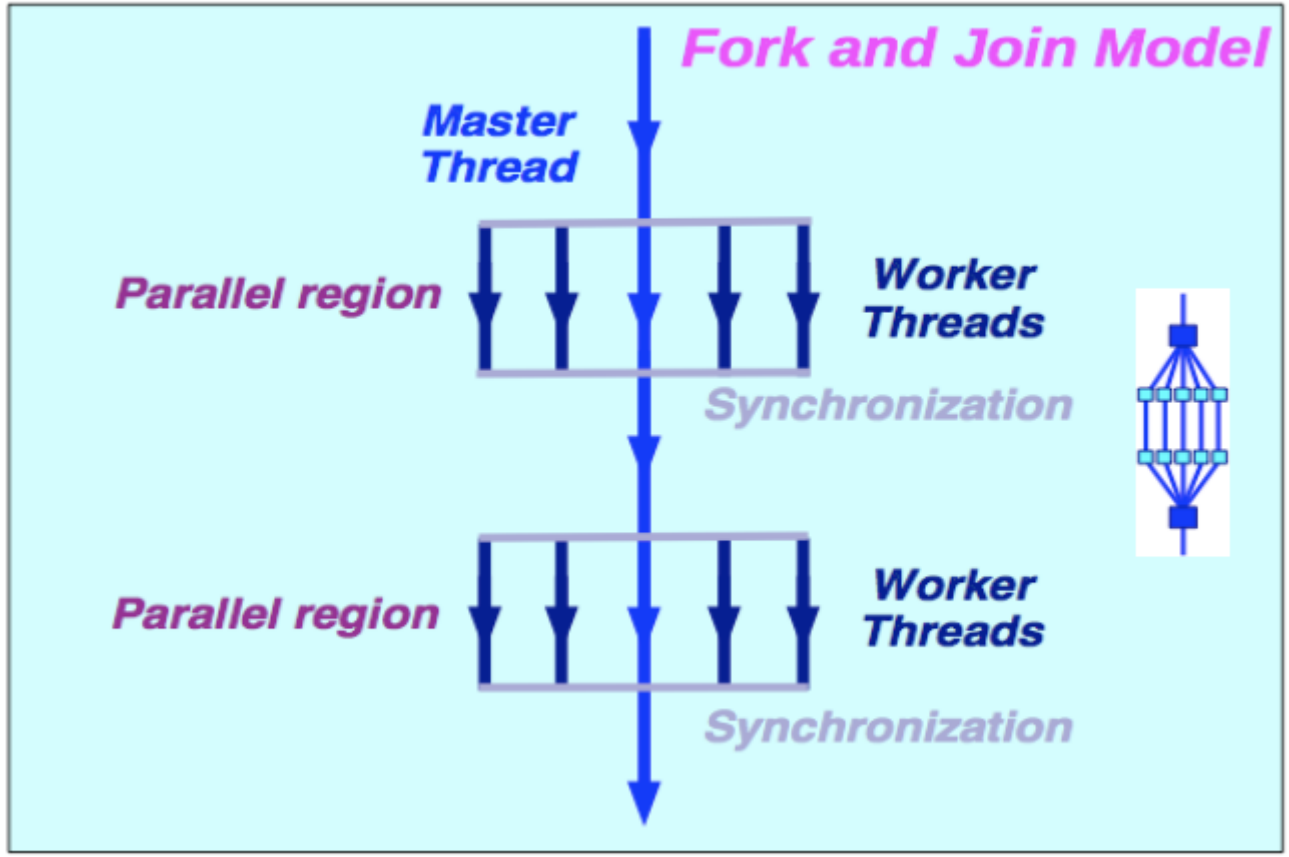
OpenMP

Modelo de programação (cont.)

- Modelo ***fork-join***



OpenMP

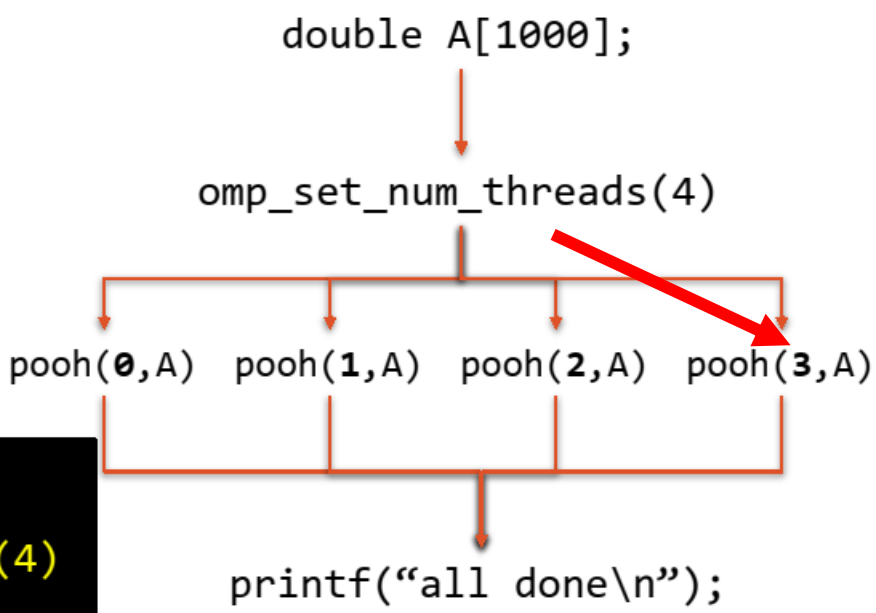

Fork and Join Model
Master Thread
Parallel region
Worker Threads
Synchronization
Parallel region
Worker Threads
Synchronization

OpenMP

Modelo de programação (cont.)

Cada thread executa o mesmo código de forma redundante.

As threads esperam para que todas as demais terminem antes de prosseguir (i.e. uma barreira)




```
double A[1000];
#pragma omp parallel num_threads(4)
{
  int ID = omp_get_thread_num();
  pooh(ID, A);
}
printf("all done\n");
```

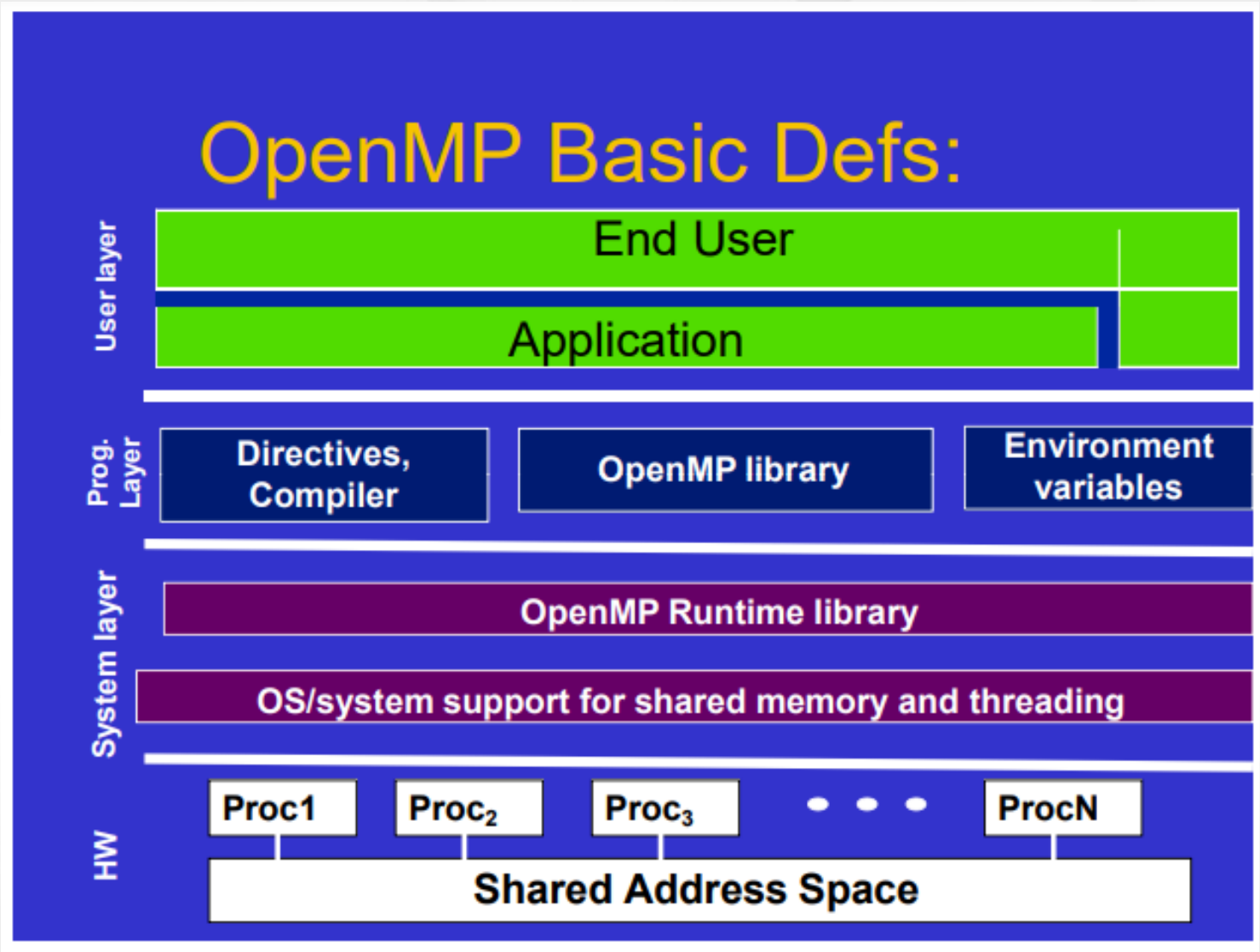
INTEL MODERN CODE PARTNER 59

OpenMP

Modelo de programação (cont.)

- Baseado em diretivas de compilação:
  - ✓o paralelismo é especificado através do uso de diretivas para o compilador que são inseridas em um código.
- Suporte a paralelismo aninhado:
  - ✓construções paralelas podem ser colocadas dentro de construções paralelas e as implementações podem ou não suportar essa característica.
- *Threads* dinâmicas:
  - ✓o número de *threads* a serem utilizadas para executar um região paralela pode ser dinamicamente alterado.

OpenMP


OpenMP Basic Defs:
User layer
End User
Application
Prog. Layer
Directives, Compiler
OpenMP library
Environment variables
System layer
OpenMP Runtime library
OS/system support for shared memory and threading
HW
Proc1
Proc2
Proc3
ProcN
Shared Address Space

INSTITUTO FEDERAL
SANTA CATARINA

OpenMP - Sintaxe principal

- Maior parte das construções são diretivas para o compilador:

  #pragma omp construct [cláusula [cláusula]...]

  Exemplo: ***#pragma omp parallel num_threads(4)***

- Protótipos e tipos de funções no arquivo:

  #include <omp.h>

- Maior parte das construções utiliza um "**bloco estruturado**":
  - ✓bloco de uma ou mais instruções com um **ponto de entrada** no topo e um **ponto de saída** na parte inferior.
    - Não há problema em ter um **exit()** dentro do bloco estruturado.

INSTITUTO FEDERAL
SANTA CATARINA

OpenMP – Opções do compilador

- Linux com **gcc**:

  - ✓ `gcc -fopenmp foo.c -o foo`
  - ✓ `export OMP_NUM_THREADS=4`
  - ✓ `./foo`