Aula 06
Paralelismo em CPU

Supercomputação

Prof. Lícia Sales Costa Lima

Recaptulando...

- Soluções de alto desempenho
- (1) Linguagem eficiente
- (2) Conhecimento do Hardware
- (3) Heurística Inteligente
- (4) Paralelismo

Hardware Importa!

Conhecer o hardware ajuda a tomar decisões para usar da melhor forma possível o recurso disponível

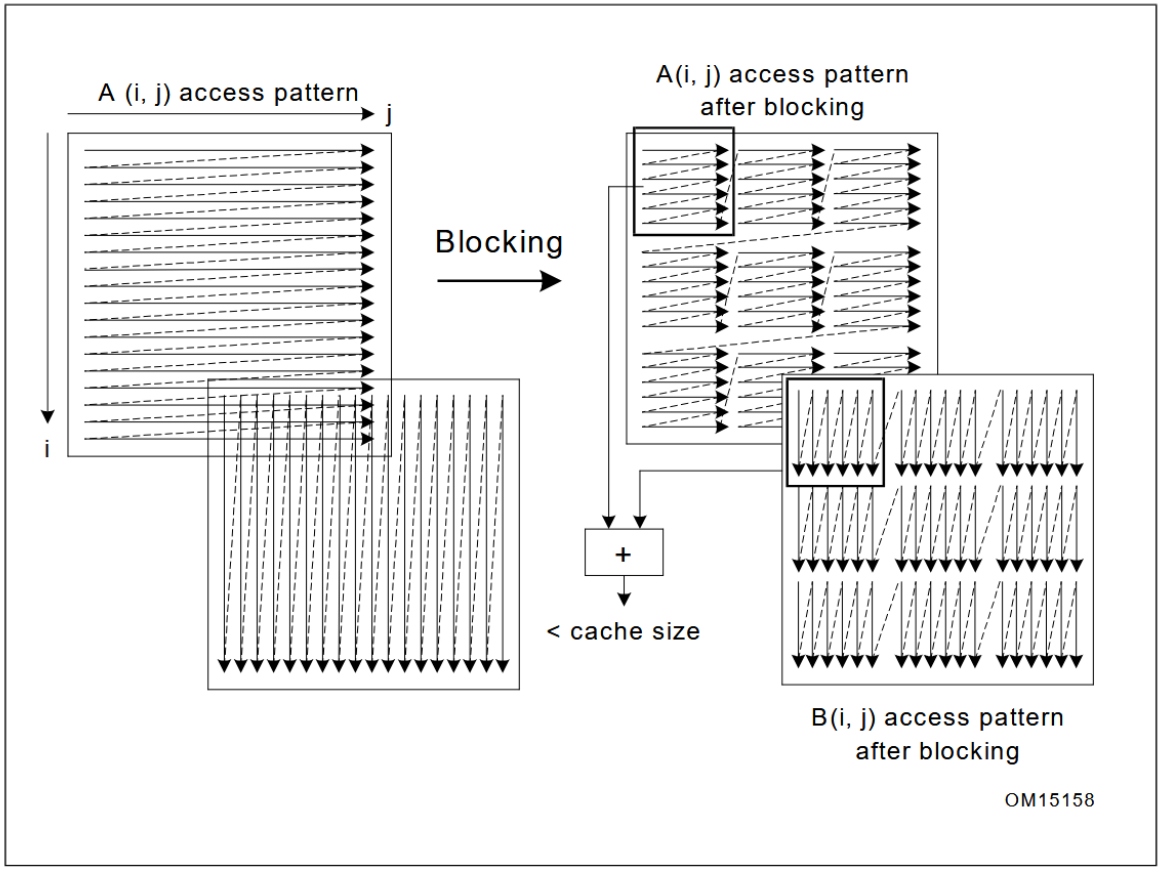
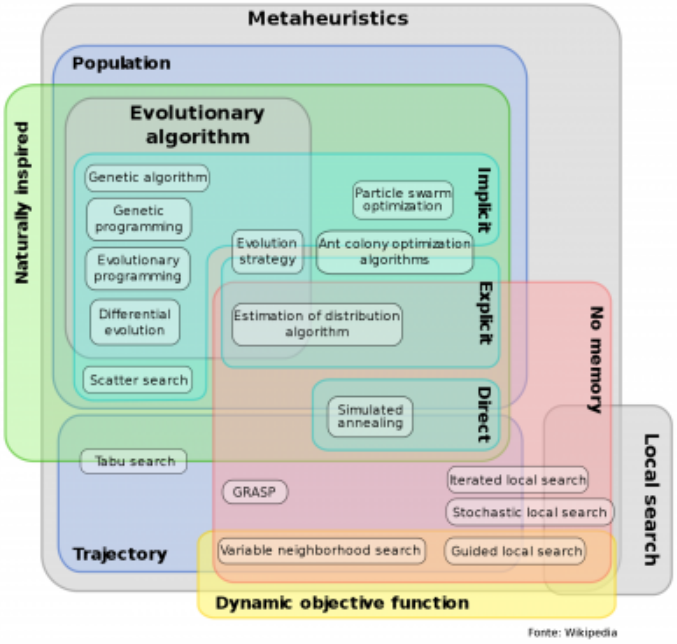


Figure 5-5. Loop Blocking Access Pattern

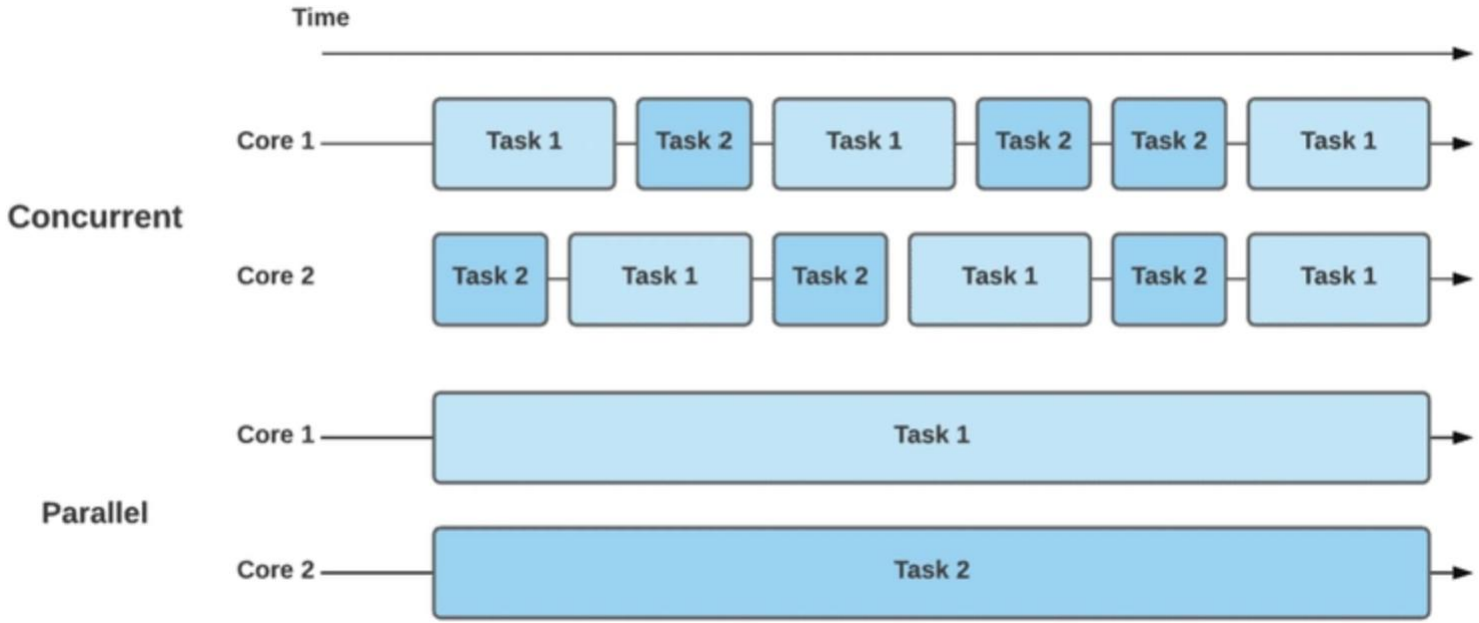
Fonte: Documentação Intel I9 pag 5-26

Heurística Inteligente



- Definir a estratégia para trabalhar com a natureza do problema
- Pegar atalhos e ganhar tempo!
- Lembre-se, em HPC, tempo é o nosso recurso mais escasso!

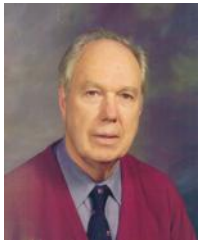
Concorrência x Paralelismo



Paralelismo

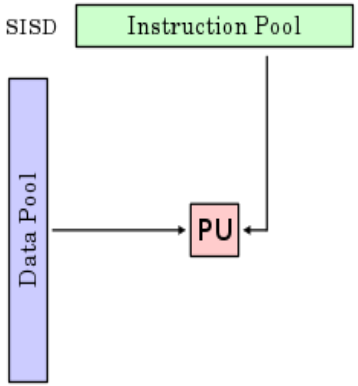
- Consiste no uso de múltiplos processadores, simultaneamente, para resolver um problema
- Tem por objetivo o aumento do desempenho, i.e., a redução do tempo necessário para resolver um problema

Taxonomia de Flynn

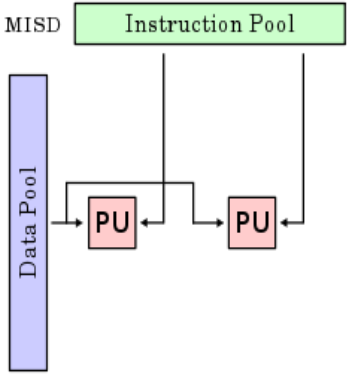


Michael J. Flynn

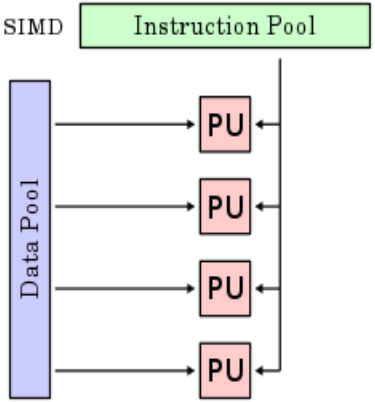
- É uma forma de classificar computadores paralelos
- Proposta por Flynn, em 1972



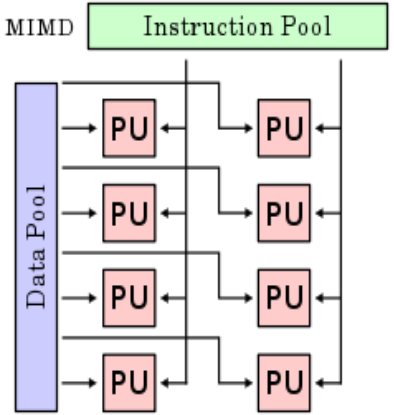
Von Newmann



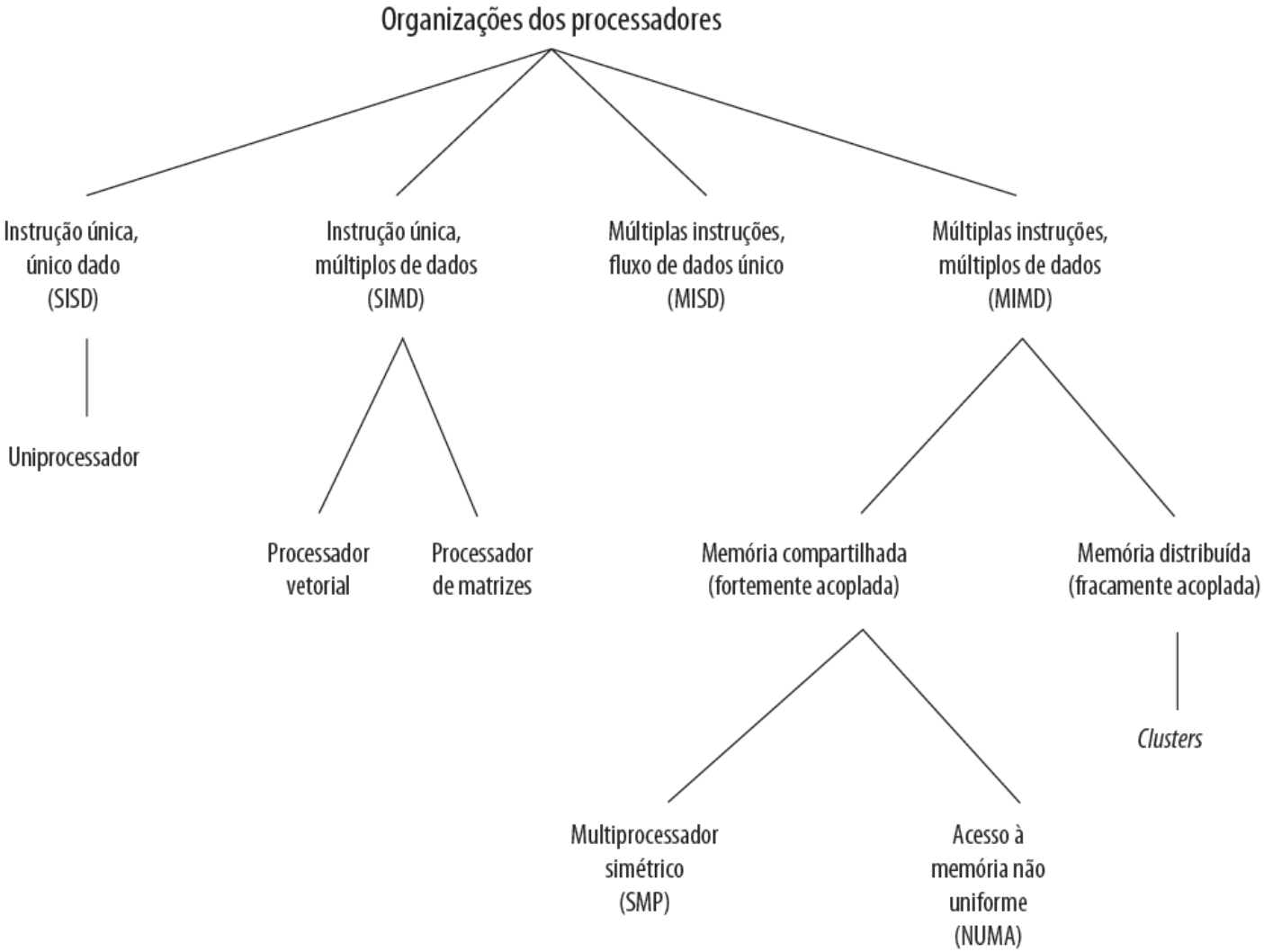
Pipeline



Array

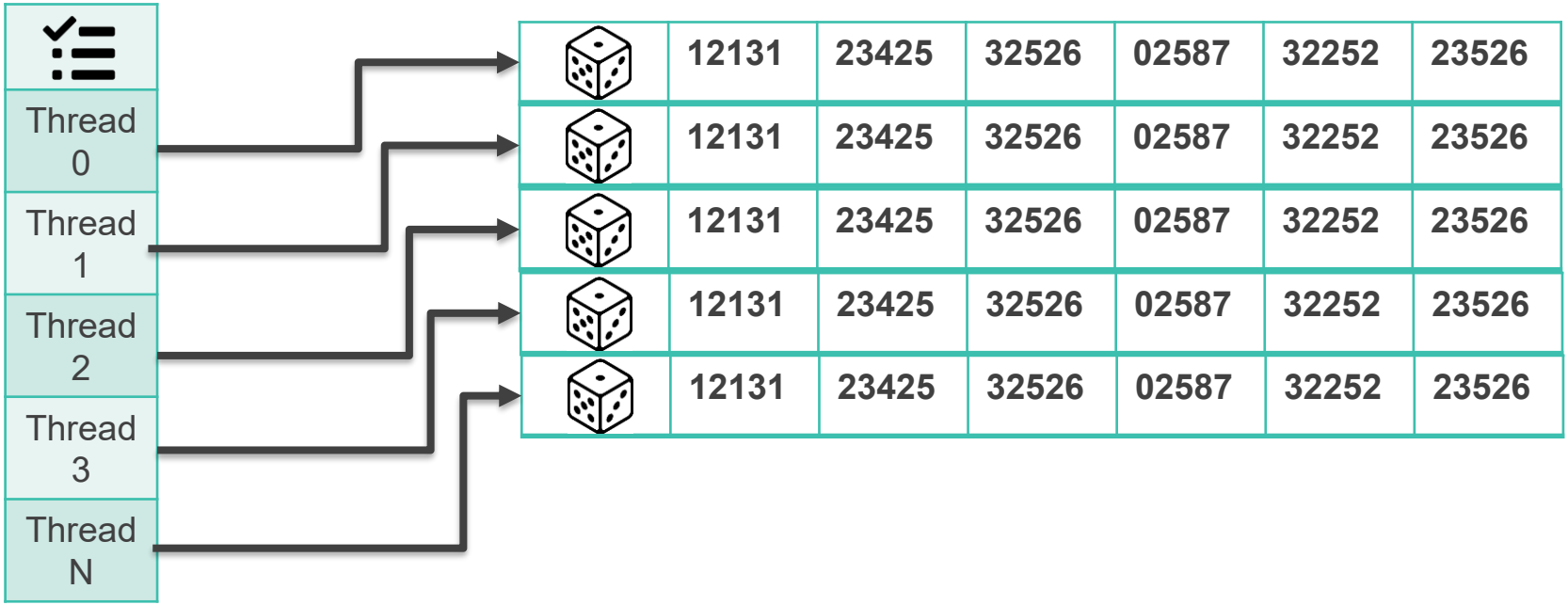


Máquinas Paralelas
(SMP, clusters e NUMA)


Organizações dos processadores
Instrução única, único dado (SISD)
Instrução única, múltiplos de dados (SIMD)
Múltiplas instruções, fluxo de dados único (MISD)
Múltiplas instruções, múltiplos de dados (MIMD)
Uniprocessador
Processador vetorial
Processador de matrizes
Memória compartilhada (fortemente acoplada)
Memória distribuída (fracamente acoplada)
Clusters
Multiprocessador simétrico (SMP)
Acesso à memória não uniforme (NUMA)

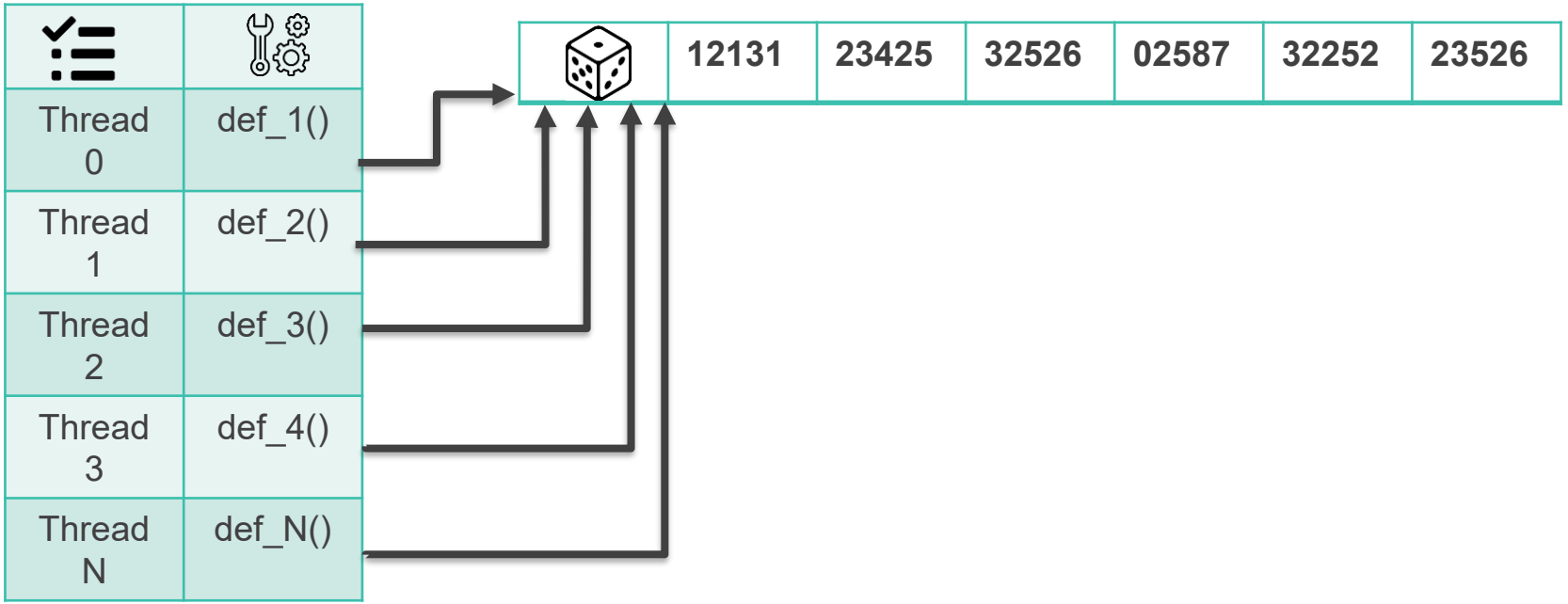
Paralelismo de dados

A mesma operação é executada para todos os elementos de um conjunto de dados

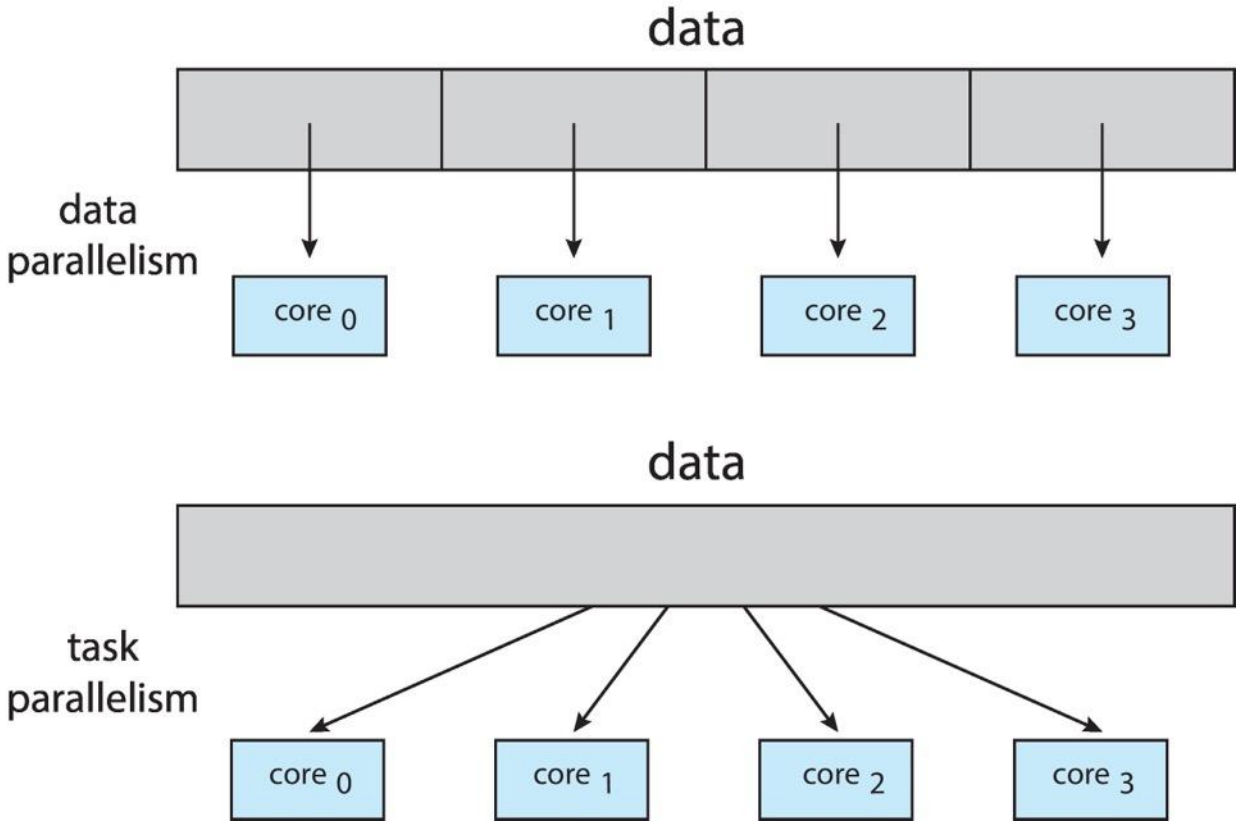


Paralelismo de tarefas

Tarefas independentes são executadas em paralelo



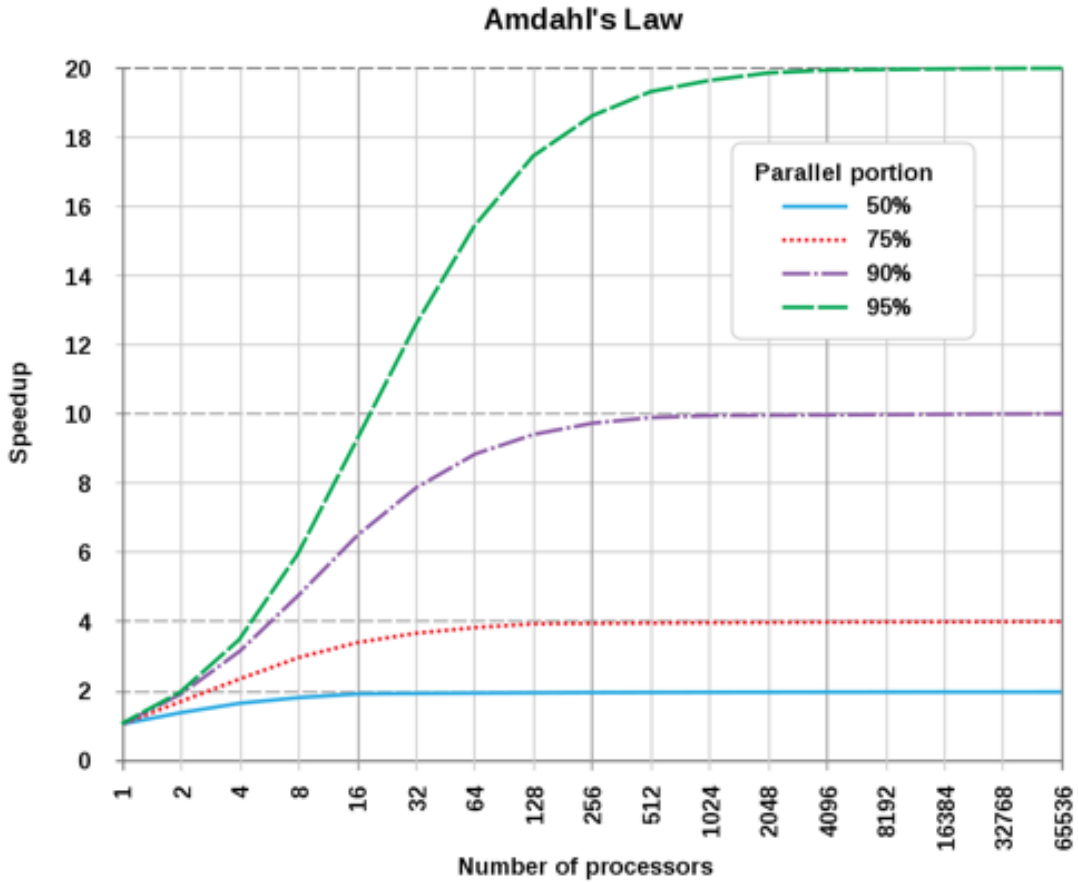
Paralelismo



Discussão

- **Quanto mais paralelo melhor?**

Discussão


Amdahl's Law
Speedup
Number of processors
Parallel portion
50%
75%
90%
95%
0
2
4
6
8
10
12
14
16
18
20
1
2
4
8
16
32
64
128
256
512
1024
2048
4096
8192
16384
32768
65536

Discussão

- **Exemplo 1 – Supondo 8 cores**

```
vector<double> dados;
vector<double> resultados;
for (int i = 0; i < dados.size(); i++) {
    resultados[i] = funcao_complexa(dados[i]);
}
```

– Tempo total / 8

Discussão

- **Exemplo 2 – Supondo 8 cores**

```
vector<double> dados;
vector<double> resultados;
resultados[0] = 0;
for (int i = 1; i < dados.size(); i++) {
    resultados[i] = funcao_complexa(dados[i], resultados[i-1]);
}
```

Complicou, Depende da iteração anterior </3

Discussão

- **Exemplo 3 – Supondo 8 cores**

```
vector<double> dados;
vector<double> resultados1;
vector<double> resultados2;
resultados1[0] = resultados2[0] 0;
for (int i = 1; i < dados.size(); i++) {
    resultados1[i] = funcao_complexa(dados[i], resultados1[i-1]);
    resultados2[i] = funcao_complexa2(dados[i], resultados2[i-1]);
}
```

Podemos calcular resultados1 e resultados2 paralelamente

Paralelismo - Resumo

1. Paralelizar significa rodar código sem dependências simultaneamente
2. Paralelismo de dados: mesma tarefas, dados diferentes
3. Paralelismo de tarefas: dividimos uma tarefa em tarefas menores
4. Existem tarefas inerentemente sequenciais
5. Ganhos são limitados a partes do programa

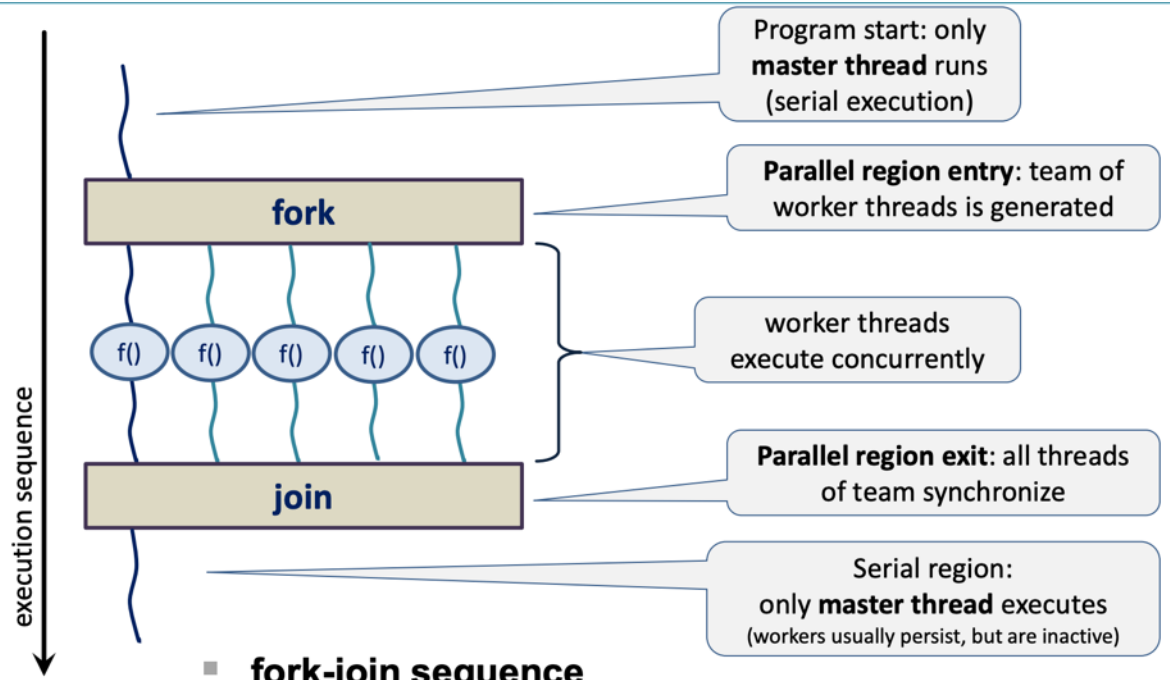
OpenMP

Sintaxe, Principais Conceitos

OpenMP

- Diretivas de compilação para paralelizar o código
- Fornece pragmas que permitem paralelizar código em ambientes multi-core
- Idealmente funciona com o mínimo de modificações no código sequencial.

OpenMP – Fork/Join



- **fork-join sequence**
  - can repeat, with differing thread counts

OpenMP - Sintaxe

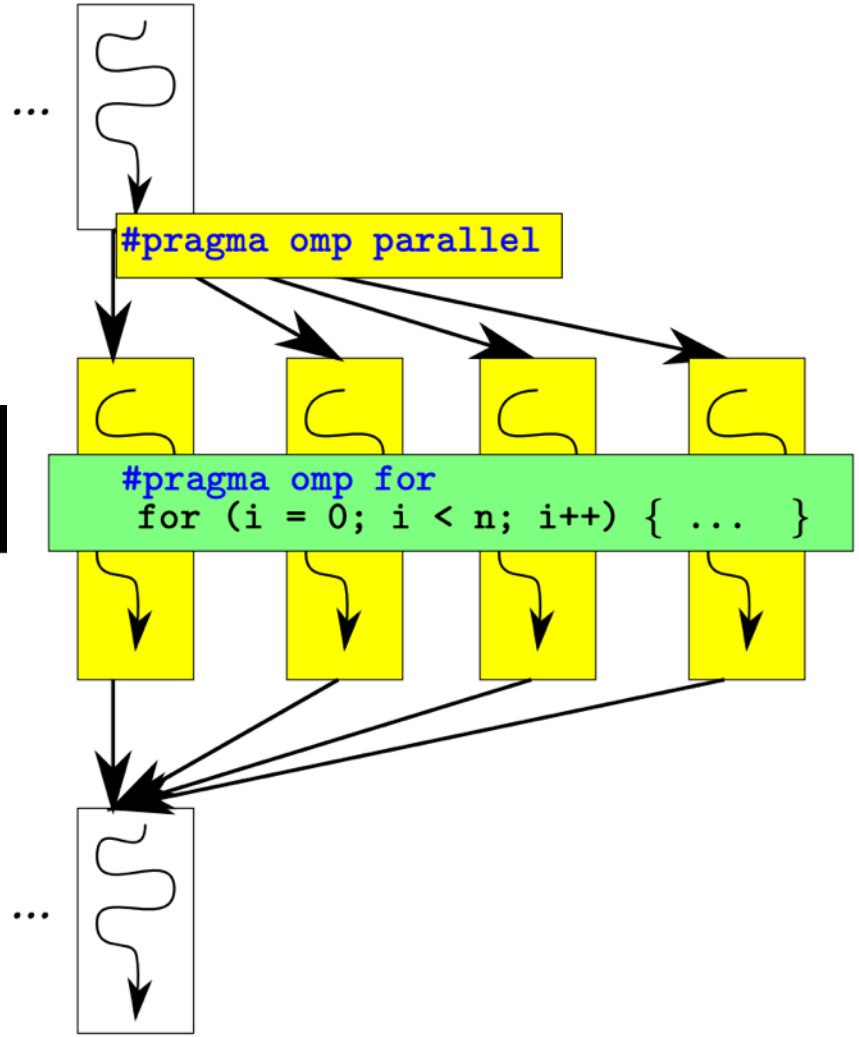
```
// Arquivo interface da biblioteca OpenMP para C/C++
#include <omp.h>

// retorna o identificador da thread.
int omp_get_thread_num();

// indica o número de threads a executar na região paralela.
void omp_set_num_threads(int num_threads);

// retorna o número de threads que estão executando no momento.
int omp_get_num_threads();
```

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



OpenMP - Sintaxe

Código sequencial

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

Região OpenMP parallel

```
#pragma omp parallel
{
  int id, i, Nthrds, istart, iend;
  id = omp_get_thread_num();
  Nthrds = omp_get_num_threads();
  istart = id * N / Nthrds;
  iend = (id+1) * N / Nthrds;
  if(id == Nthrds-1) iend = N;
  for(i = istart; i < iend; i++)
    a[i] = a[i] + b[i];
}
```

Região paralela OpenMP com uma construção de divisão de laço

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

OpenMP - Scheduling

#pragma omp for schedule(static)

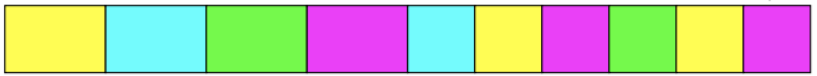
#pragma omp for schedule(static,3)

#pragma omp for schedule(dynamic)

#pragma omp for schedule(dynamic,2)

#pragma omp for schedule(guided)

#pragma omp for schedule(guided,2)



Pedindo Recursos ao SLURM

```
#!/bin/bash
#SBATCH --job-name=teste__openmp
#SBATCH --output=saida_%j.txt
#SBATCH --cpus-per-task=5 # 5 threads
#SBATCH --time=00:01:00
#SBATCH --partition=gpu
#SBATCH –mem=2G

# garante que o OpenMP use exatamente os recursos alocados pelo SLURM
export OMP_NUM_THREADS=${SLURM_CPUS_PER_TASK}

./seu_binario
```