Insper

Supercomputação

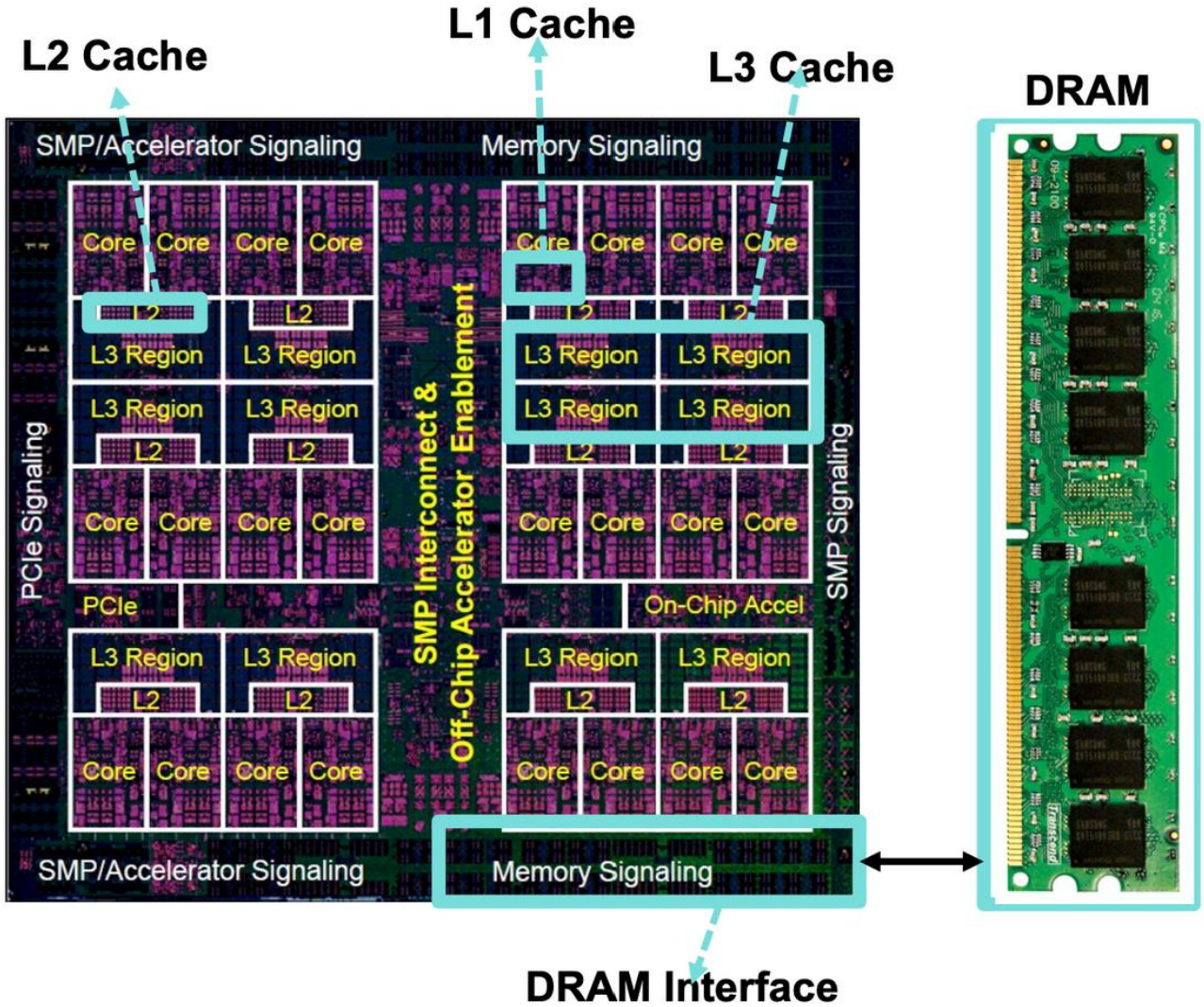
Paralelismo com OpenMP

Recaptulando...

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

Python x C++

Linguagem Importa!


L2 Cache
L1 Cache
L3 Cache
DRAM
SMP/Accelerator Signaling
Memory Signaling
Core
L2
L3 Region
PCIe Signaling
SMP Interconnect &
Off-Chip Accelerator Enablement
SMP Signaling
PCIe
On-Chip Accel
SMP/Accelerator Signaling
Memory Signaling
DRAM Interface

Hardware Importa!

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

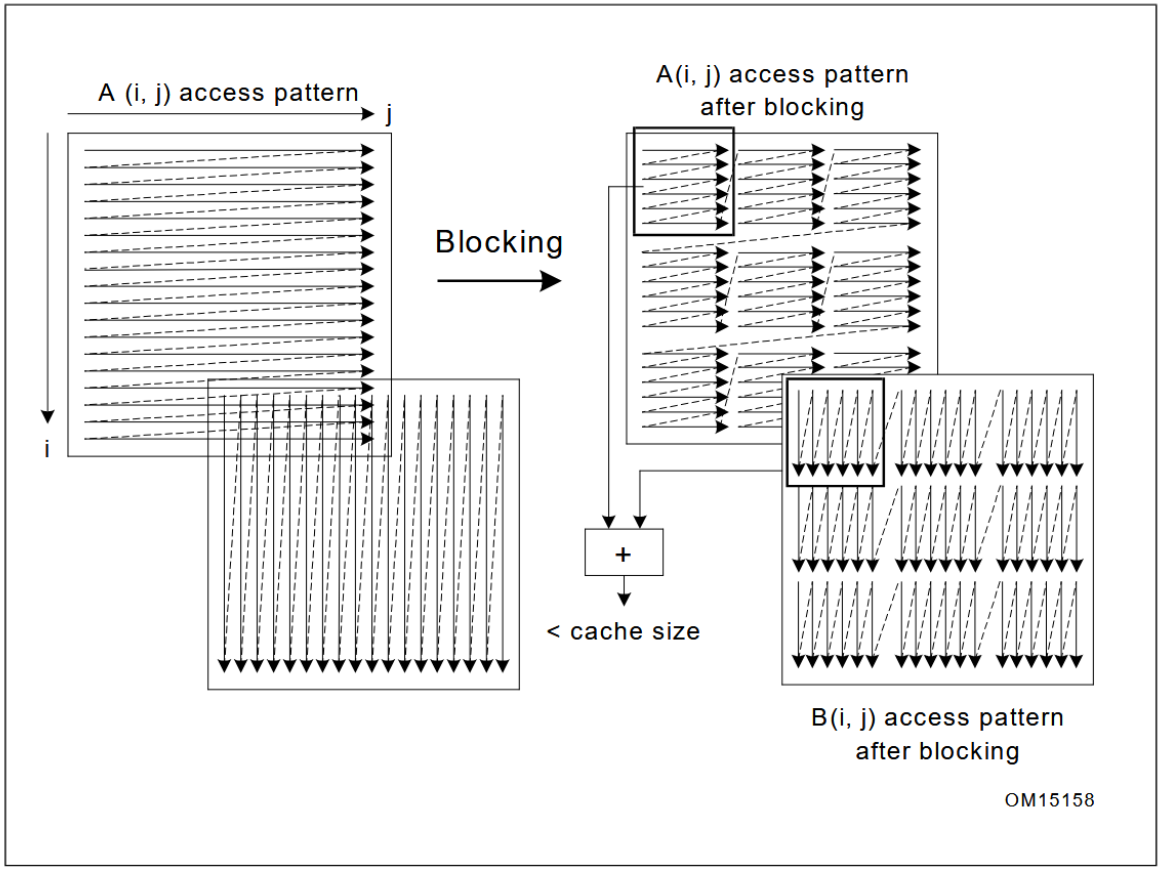
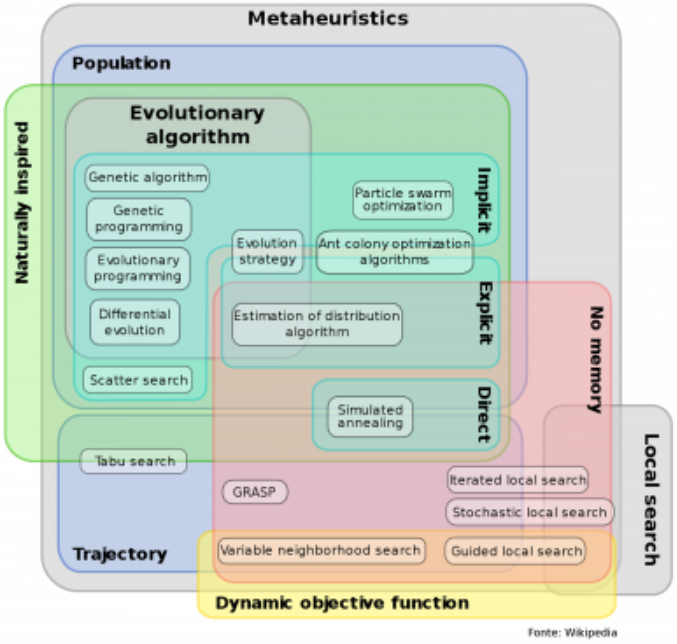


Figure 5-5. Loop Blocking Access Pattern

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!

Paralelismo

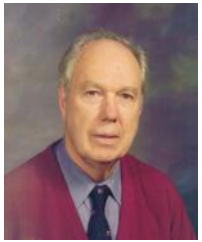
- Consiste no uso de múltiplos processadores, simultaneamente, para resolver um problema

- Tem por objetivo o aumento do desempenho, i.e., a redução do tempo necessário para resolver um problema

Paralelismo

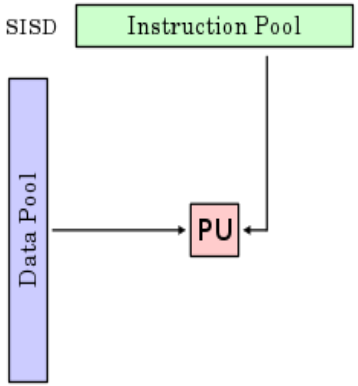
- Usamos paralelismo normalmente por 2 motivos
- (1) Problemas cada vez mais complexos e/ou maiores
- (2) Clock dos processadores se aproximando dos limites ditados pela física

Taxonomia de Flynn

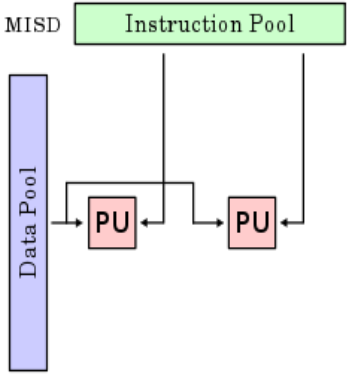


Michael J. Flynn

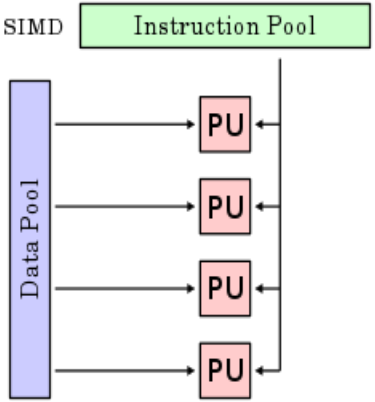
- É uma forma de classificar computadores paralelos
- Proposta por Flynn, em 1972



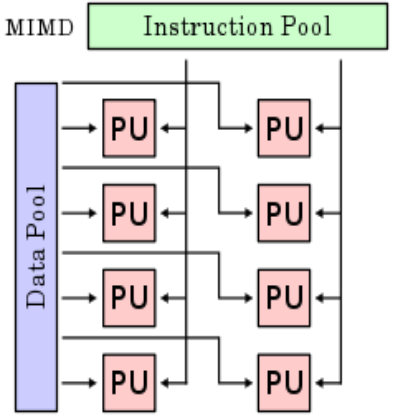
Von Newmann



Pipeline

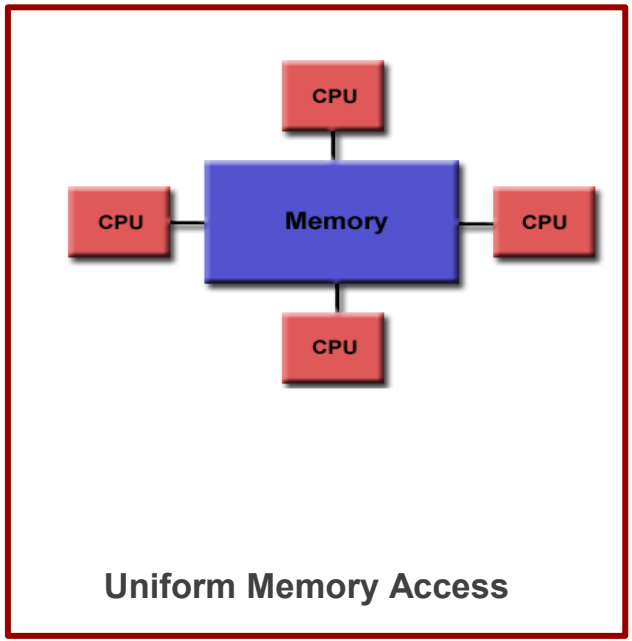


Array

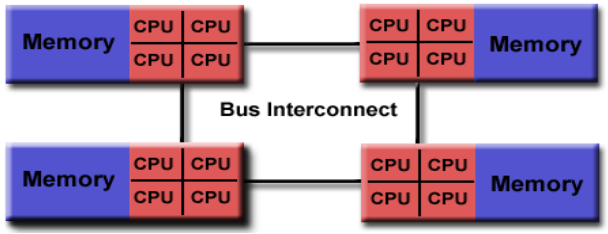


Máquinas Paralelas
(SMP, clusters e NUMA)

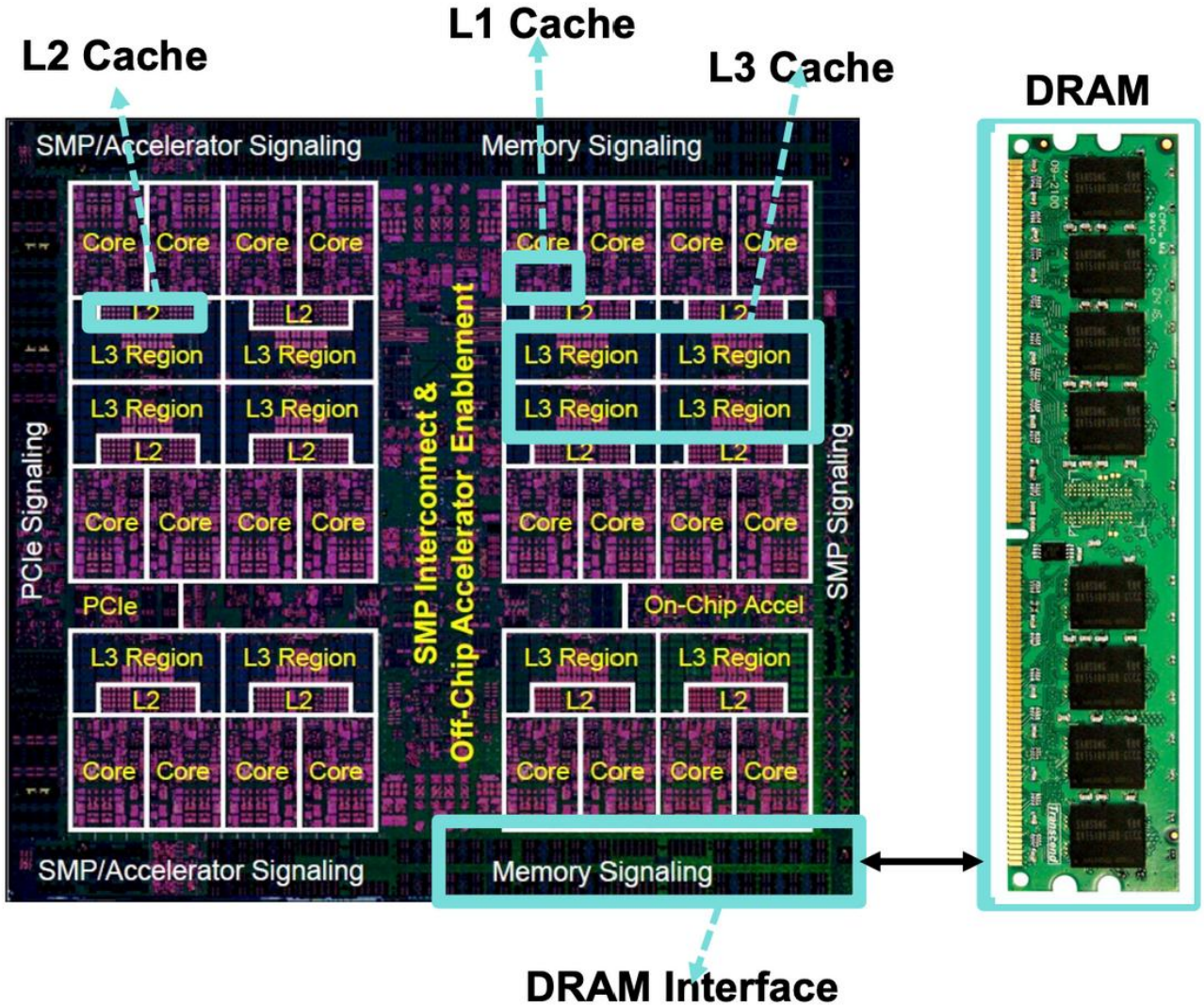
Sistemas Multi-core

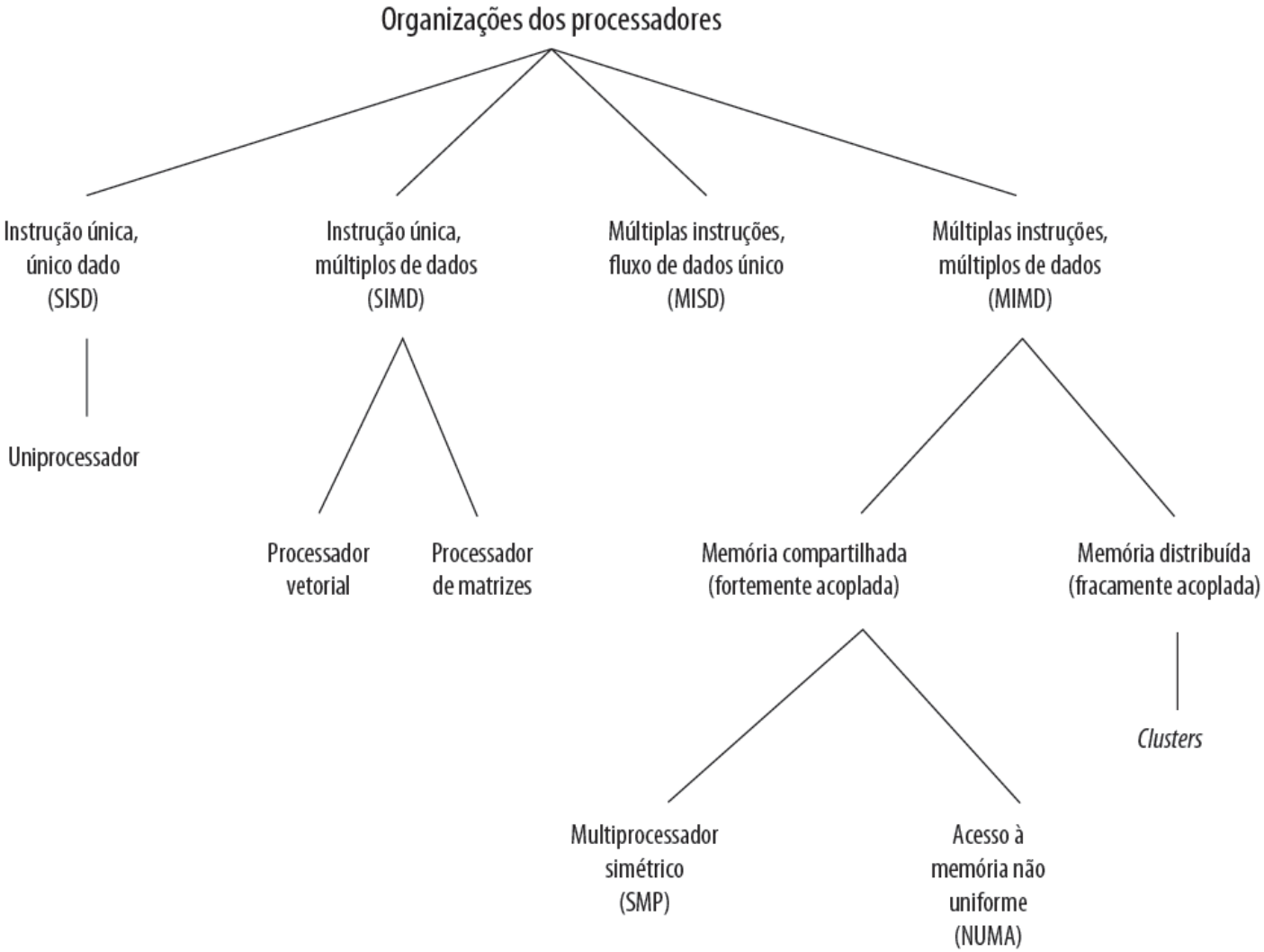


Uniform Memory Access



Non-Uniform Memory Access

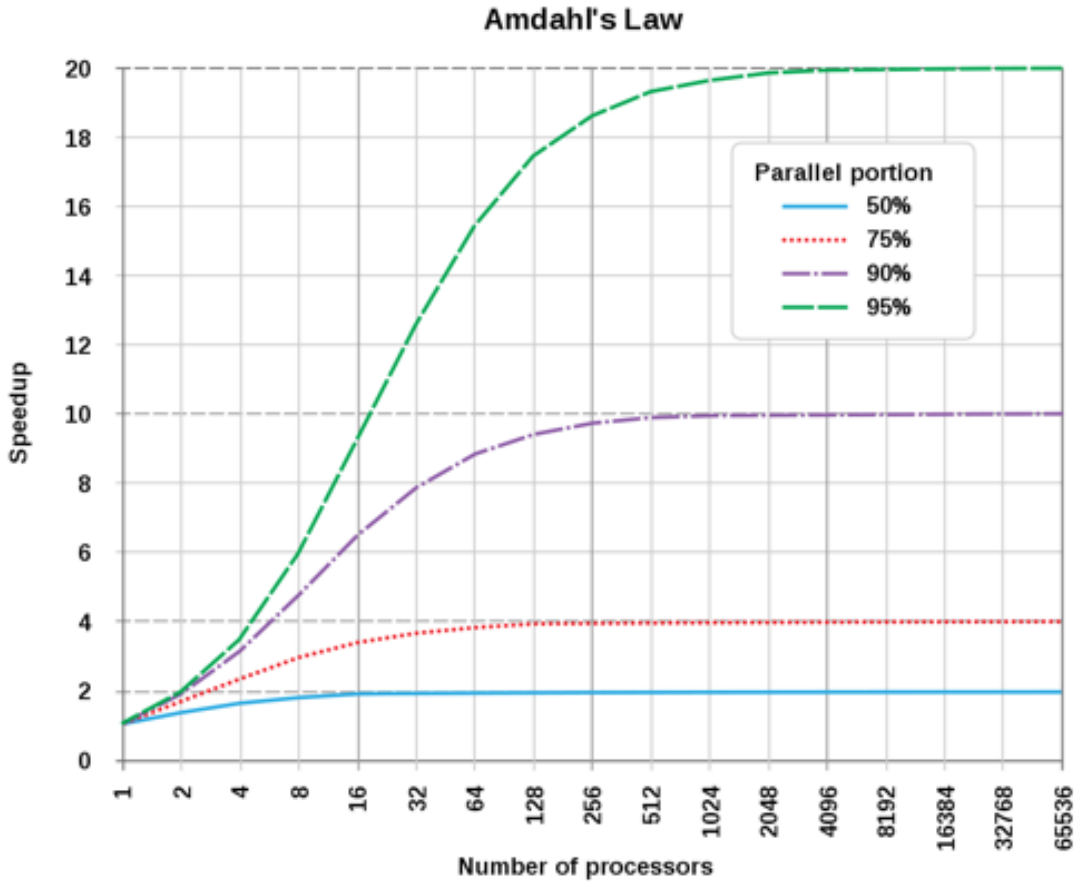

L2 Cache
L1 Cache
L3 Cache
DRAM
SMP/Accelerator Signaling
Memory Signaling
Core
L2
L3 Region
PCIe Signaling
SMP Interconnect &
Off-Chip Accelerator Enablement
SMP Signaling
PCIe
On-Chip Accel
SMP/Accelerator Signaling
Memory Signaling
DRAM Interface


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)

Discussão

- **Discussão 1: qual a expectativa da melhoria de velocidade com paralelismo?**

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]);
}
```

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]);
}
```

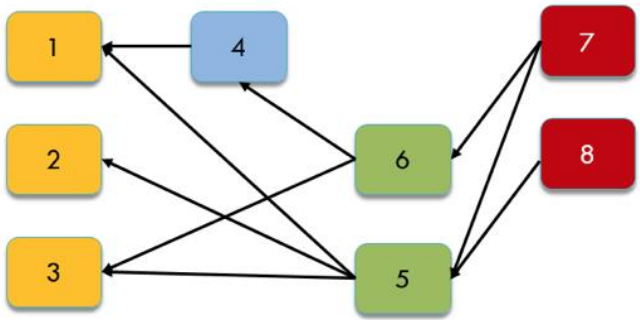
Discussão

- **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 :(

Dependência



- É quando uma iteração depende de resultados calculados em iterações anteriores

- Quando não existe nenhuma dependência em um loop, por exemplo, dizemos que ele é ingenuamente paralelizável

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]);
}
```

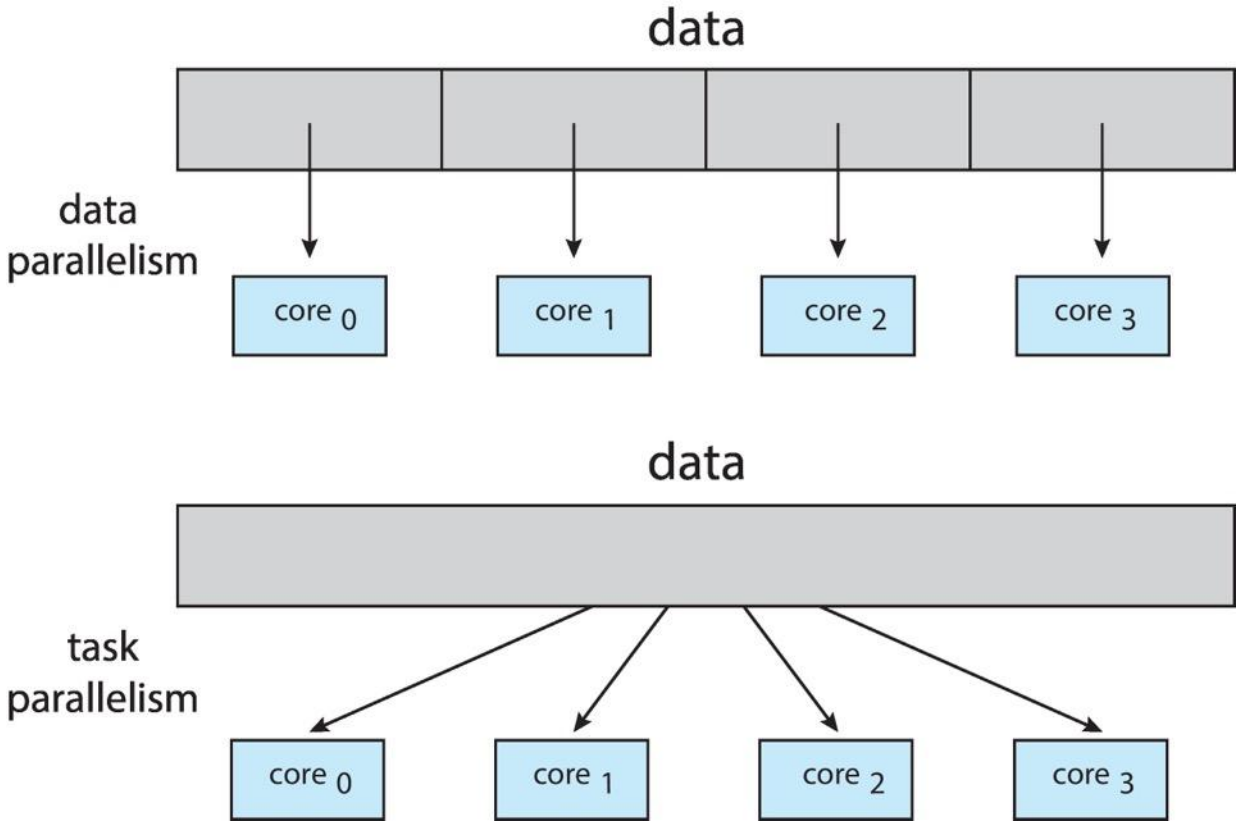
Podemos calcular resultados1 e resultados2 paralelamente

Paralelismo

- **Paralelismo de dados:** a mesma operação é executada para todos os elementos de um conjunto de dados

- **Paralelismo de tarefas:** duas ou mais tarefas independentes são executadas em paralelo. Se houver dependências, quebramos o problema em partes independents e rodamos na ordem adequada

Paralelismo



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

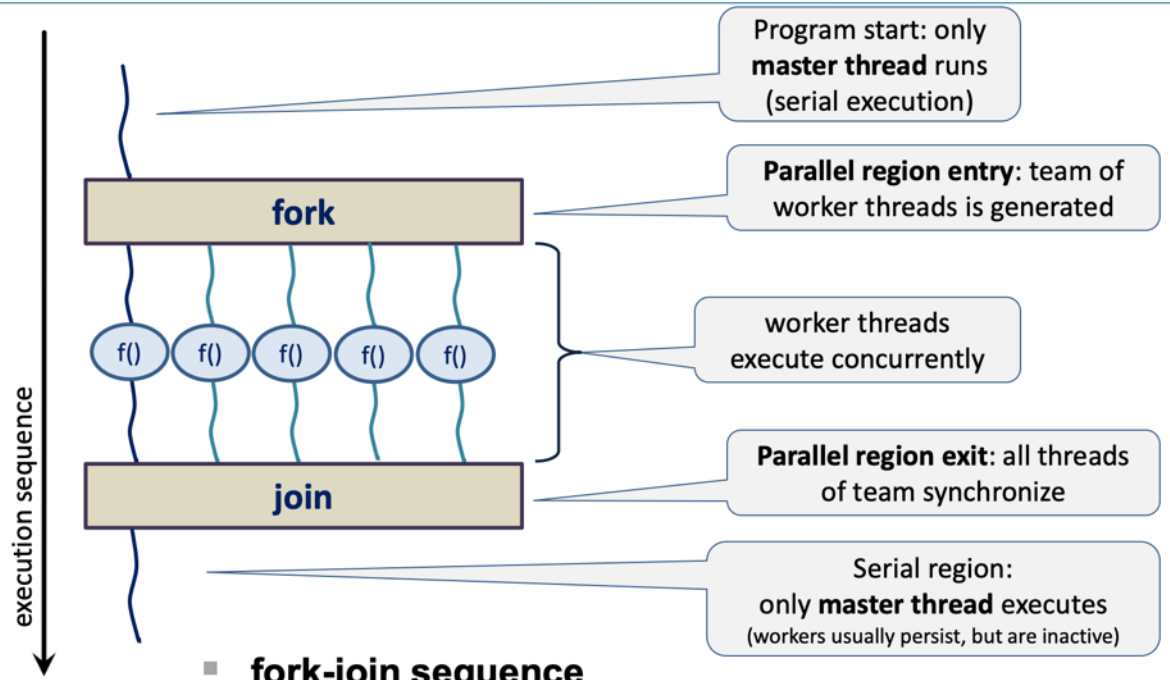
OpenMP

Sintaxe, Principais Conceitos

OpenMP

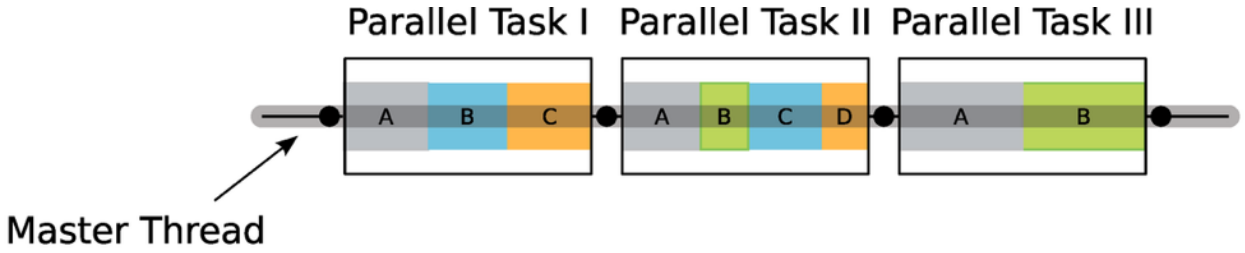
- Conjunto de extensões para C/C++ e Fortran
- Fornece construções que permitem paralelizar código em ambientes multi-core
- Idealmente funciona com o mínimo de modificações no código sequêncial

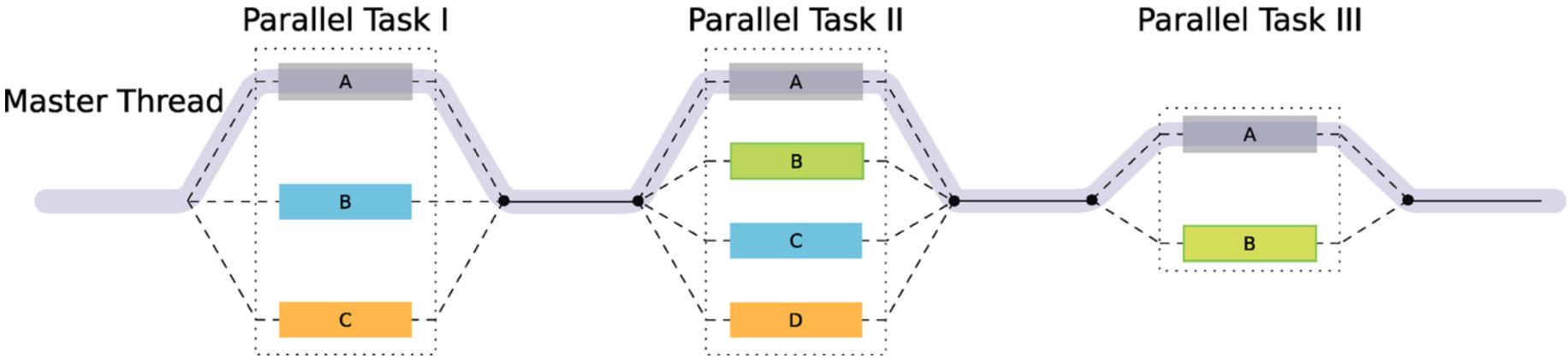
OpenMP – Fork/Join



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

OpenMP


Parallel Task I
Parallel Task II
Parallel Task III
A
B
C
A
B
C
D
A
B
Master Thread


Parallel Task I
Parallel Task II
Parallel Task III
Master Thread
A
B
C
A
B
C
D
A
B

OpenMP - Sintaxe

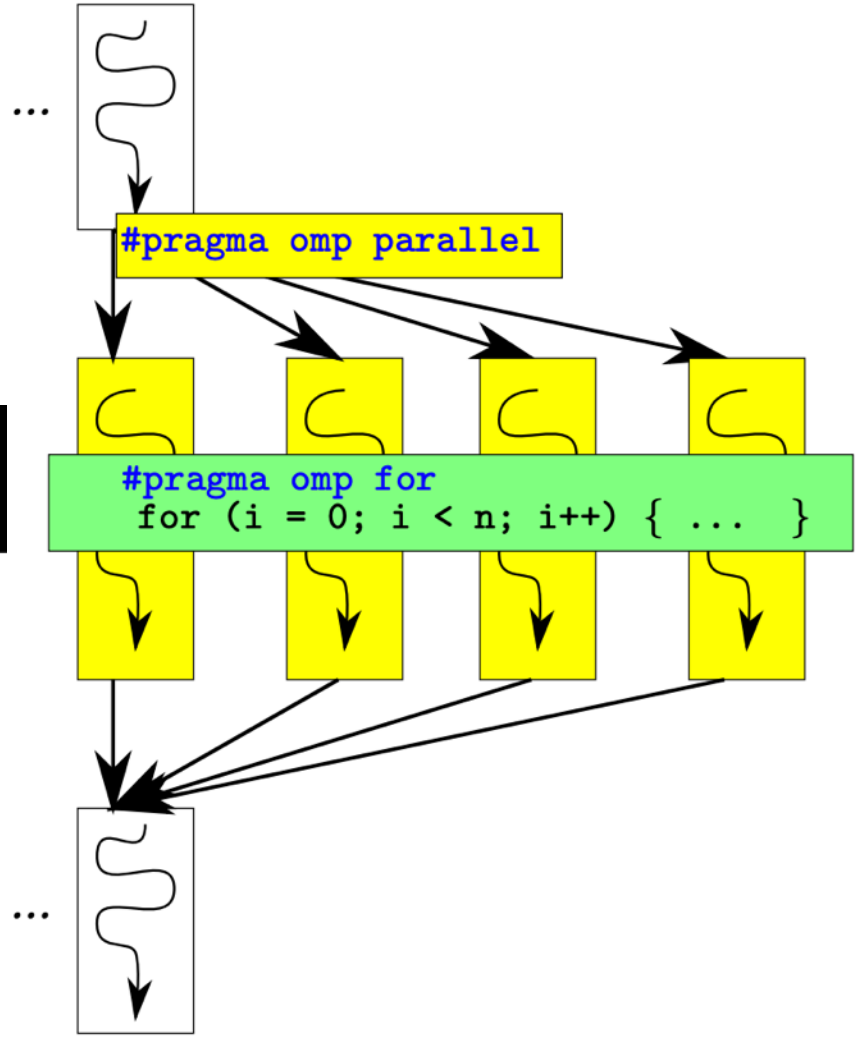
```
// Arquivo interface da biblioteca OpenMP para C/C++
#include <omp.h>

// 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();
```

OpenMP - Sintaxe

most commonly used subset

Name	Result type	Purpose
`omp_set_num_threads (int num_threads)`	none	number of threads to be created for subsequent parallel region
`omp_get_num_threads()`	int	number of threads in **currently executing** region
`omp_get_max_threads()`	int	maximum number of threads that can be created for a subsequent parallel region
`omp_get_thread_num()`	int	thread number of calling thread (zero based) in **currently executing** region
`omp_get_num_procs()`	int	number of processors available
`omp_get_wtime()`	double	return wall clock time in seconds since some (fixed) time in the past
`omp_get_wtick()`	double	resolution of timer in seconds

```
#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)

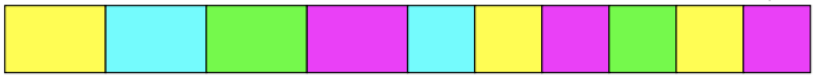
#pragma omp for schedule(static,3)

#pragma omp for schedule(dynamic)

#pragma omp for schedule(dynamic,2)

#pragma omp for schedule(guided)

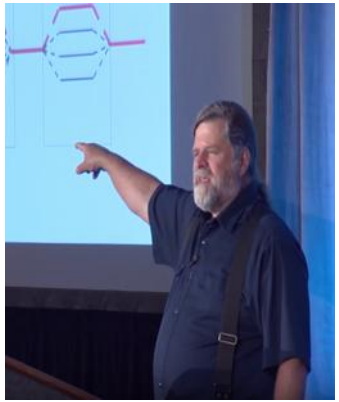
#pragma omp for schedule(guided,2)



OpenMP – onde aprender mais

A brief Introduction to parallel programming

Tim Mattson
Intel Corp.
timothy.g.mattson@ intel.com



Vídeos:

https://www.youtube.com/watch?v=pRtTIW9-Nr0
https://www.youtube.com/watch?v=LRsQHDAqPHA
https://www.youtube.com/watch?v=dK4PITrQtjY
https://www.youtube.com/watch?v=WvoMpG_QvBU

Slides:

https://extremecomputingtraining.anl.gov/wp-content/uploads/sites/96/2016/08/Mattson_830aug3_HandsOnIntro.pdf