IMD1116 COMPUTAÇÃO DE ALTO DESEMPENHO

Prof. Samuel Xavier de Souza

HISTÓRICO E EVOLUÇÃO DA PROGRAMAÇÃO

- Objetivos desse tópico
  - Reconhecer a necessidade de maior conhecimento do hardware para alcançar alto desempenho [Tarefas 1 e 2]
  - Reconhecer a necessidade da programação paralela para alcançar alto desempenho [Tarefa 3]

HISTÓRICO E EVOLUÇÃO DA PROGRAMAÇÃO

Tarefa 1:
Implemente duas versões da multiplicação de matriz por vetor (MxV) em C: uma com acesso à matriz por linhas (linha externa, coluna interna) e outra por colunas (coluna externa, linha interna). Meça o tempo de execução de cada versão com uma função apropriada e execute testes com diferentes tamanhos de matriz. Identifique a partir de que tamanho os tempos passam a divergir significativamente e explique por que isso ocorre, relacionando suas observações com o uso da memória cache e o padrão de acesso à memória.

- O gargalo de von Neumman e a memória cache
  - Localidade temporal e espacial
  - Row-major vs. column-major

HISTÓRICO E EVOLUÇÃO DA PROGRAMAÇÃO

Tarefa 2:
Implemente três laços em C para investigar os efeitos do paralelismo ao nível de instrução (ILP): 1) inicialize um vetor com um cálculo simples; 2) some seus elementos de forma acumulativa, criando dependência entre as iterações; e 3) quebre essa dependência utilizando múltiplas variáveis. Compare o tempo de execução das versões compiladas com diferentes níveis de otimização (O0, O2, O3) e analise como o estilo do código e as dependências influenciam o desempenho.

- Paralelismo ao nível de instrução
  - Pipelining,
  - Vetorização

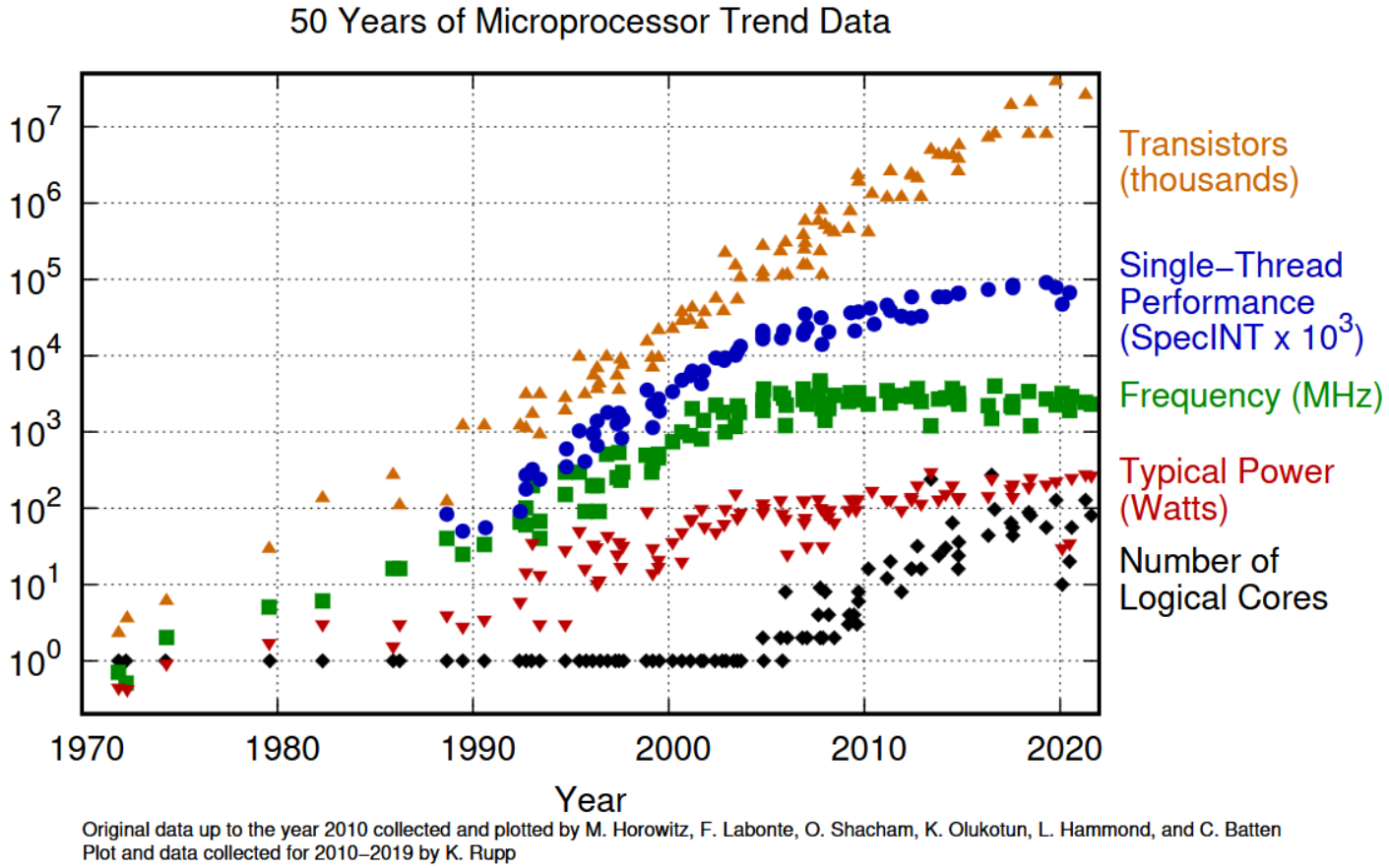
HISTÓRICO E EVOLUÇÃO DA PROGRAMAÇÃO

- Por que algumas aplicações requerem maior poder computacional continuamente?

HISTÓRICO E EVOLUÇÃO DA PROGRAMAÇÃO

Tarefa 3:
Implemente um programa em C que calcule uma aproximação de π usando uma série matemática, variando o número de iterações e medindo o tempo de execução. Compare os valores obtidos com o valor real de π e analise como a acurácia melhora com mais processamento. Reflita sobre como esse comportamento se repete em aplicações reais que demandam resultados cada vez mais precisos, como simulações físicas e inteligência artificial.

- As limitações físicas da fabricação de processadores sequenciais mais rápidos



PROGRAMAS MULTITAREFAS

- Objetivos desse tópico
  - Reconhecer a necessidade de maior conhecimento do software para alcançar alto desempenho
  - Reconhecer a maior dificuldade de se programar em paralelo

PROGRAMAS MULTITAREFAS

Tarefa 4:
Implemente dois programas paralelos em C com OpenMP: um limitado por memória, com somas simples em vetores, e outro limitado por CPU, com cálculos matemáticos intensivos. Paralelize com `#pragma omp parallel for` e meça o tempo de execução variando o número de threads. Analise quando o desempenho melhora, estabiliza ou piora, e reflita sobre como o multithreading de hardware pode ajudar em programas memory-bound, mas atrapalhar em programas compute-bound pela competição por recursos.

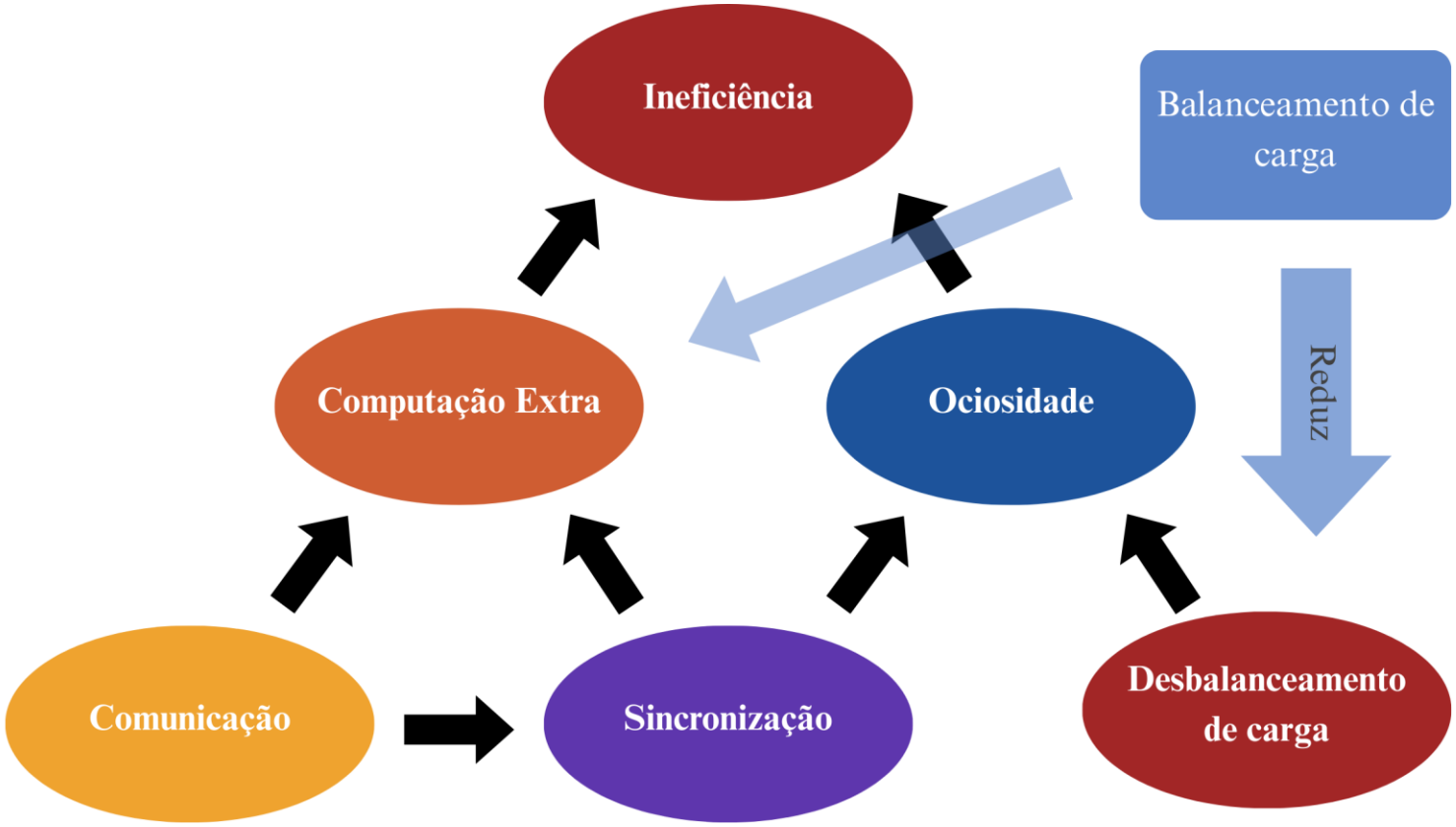
- Quais programas são limitados pela memória (gargalo de von Neuman) e quais não são?
  - Como o multi-thread de hardware pode ajudar (ou atrapalhar)

PROGRAMAS MULTITAREFAS

- Por que é mais difícil programar em paralelo?
  - Sincronização, comunicação e equilíbrio de carga

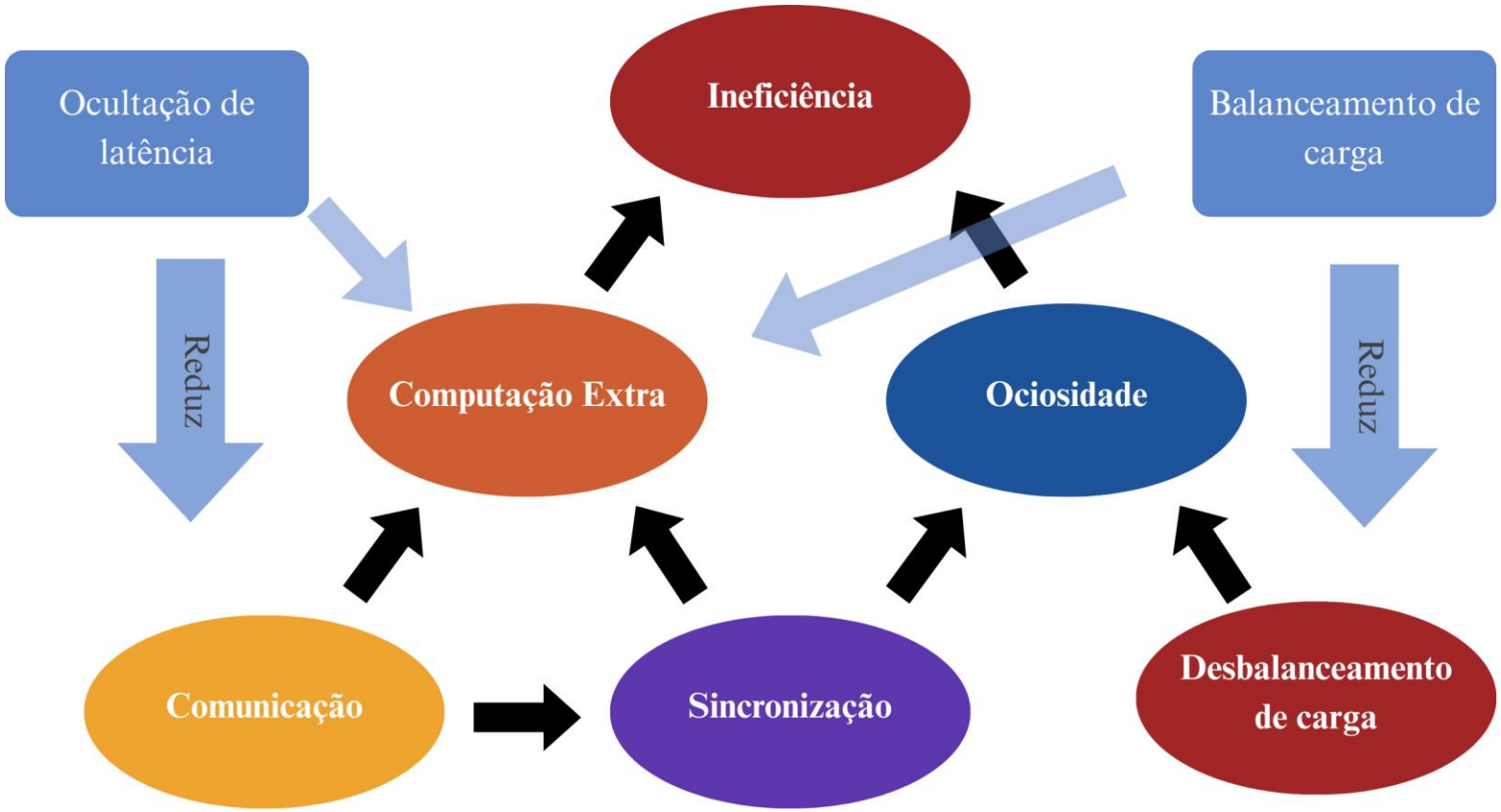
PROGRAMAS MULTITAREFAS

- Por que é mais difícil programar em paralelo?
  - Sincronização, comunicação e equilíbrio de carga



PROGRAMAS MULTITAREFAS

- Por que é mais difícil programar em paralelo?
  - Sincronização, comunicação e equilíbrio de carga



PROGRAMAS MULTITAREFAS

Tarefa 5:
Implemente um programa em C que conte quantos números primos existem entre 2 e um valor máximo *n*. depois, paralelize o laço principal usando a diretiva `#pragma omp parallel for` sem alterar a lógica original. compare o tempo de execução e os resultados das versões sequencial e paralela. observe possíveis diferenças no resultado e no desempenho, e reflita sobre os desafios iniciais da programação paralela, como correção e distribuição de carga.

- Por que é mais difícil programar em paralelo?
  - Sincronização, comunicação e equilíbrio de carga

PROGRAMAÇÃO EM MEMÓRIA COMPARTILHADA

- Objetivos desse tópico
  - Conhecer as ferramentas básicas de programação em memória compartilhada

PROGRAMAÇÃO EM MEMÓRIA COMPARTILHADA

- Parallel e parallel for
- Escopo de variáveis
- Condição de corrida e regiões críticas

PROGRAMAÇÃO EM MEMÓRIA COMPARTILHADA

- Parallel e parallel for
- Escopo de variáveis
- Condição de corrida e regiões críticas

Tarefa 6:
Implemente em C a estimativa estocástica de π. Paralelize com `#pragma omp parallel for` e explique o resultado incorreto. Corrija a condição de corrida utilizando o `#pragma omp critical` e reestruturando com `#pragma omp parallel` seguido de `#pragma omp for` e aplicando as cláusulas `private`, `firstprivate`, `lastprivate` e `shared`. Teste diferentes combinações e explique como cada cláusula afeta o comportamento do programa. Comente também como a cláusula `default(none)` pode ajudar a tornar o escopo mais claro em programas complexos.

PROGRAMAÇÃO EM MEMÓRIA COMPARTILHADA

- `#pragma omp single`
- `#pragma omp master`
- `#pragma omp barrier`
- Cláusula `nowait`
- Modelo de tarefas
  - `#pragma omp task`
  - `#pragma omp taskwait`

PROGRAMAÇÃO EM MEMÓRIA COMPARTILHADA

Tarefa 7:
Implemente um programa em C que cria uma lista encadeada com nós, cada um, contendo o nome de um arquivo fictício. Dentro de uma região paralela, percorra a lista e crie uma tarefa com `#pragma omp task` para processar cada nó. Cada tarefa deve imprimir o nome do arquivo e o identificador da thread que a executou. Após executar o programa, reflita: todos os nós foram processados? Algum foi processado mais de uma vez ou ignorado? O comportamento muda entre execuções? Como garantir que cada nó seja processado uma única vez e por apenas uma tarefa?

- `#pragma omp single`
- `#pragma omp master`
- `#pragma omp barrier`
- Cláusula `nowait`
- Modelo de tarefas
  - `#pragma omp task`
  - `#pragma omp taskwait`

COERÊNCIA DE CACHE E FALSO COMPARTILHAMENTO

- Objetivos desse tópico
  - Entender as vantagens do uso de variáveis compartilhadas e aprender a evitar as desvantagens
  - Entender o que é falso compartilhamento e aprender a evitá-lo

COERÊNCIA DE CACHE E FALSO COMPARTILHAMENTO

- Variáveis compartilhadas:
  - Vantagens para somente leitura ou escrita infrequente (evita cópias com `private` e `firstprivate`)
  - Desvantagens para escrita frequente: condição de corrida ou serialização devido a sincronização
- Coerência de cache com caches privadas
  - Write-through vs. Write-back
    - Bit de sujo (Dirty bit)
    - Bit de inválido (Invalid bit) em múltiplas caches
  - Protocolos de coerência de cache
    - Snooping
    - Baseados em diretórios
  - Variáveis compartilhadas: escrita frequente torna cache inútil e deteriora ainda mais o acesso a memória (dois acessos)
  - Falso compartilhamento: torna a cache inútil mesmo sem compartilhamento de variáveis

COERÊNCIA DE CACHE E FALSO COMPARTILHAMENTO

Tarefa 8:
Implemente estimativa estocástica de π usando `rand()` para gerar os pontos. Cada thread deve usar uma variável privada para contar os acertos e acumular o total em uma variável global com `#pragma omp critical`. Depois, implemente uma segunda versão em que cada thread escreve seus acertos em uma posição distinta de um vetor compartilhado. A acumulação deve ser feita em um laço serial após a região paralela. Compare o tempo de execução das duas versões. Em seguida, substitua `rand()` por `rand_r()` em ambas e compare novamente. Explique o comportamento dos quatro programas com base na coerência de cache e nos efeitos do falso compartilhamento.

- Variáveis compartilhadas:
  - Vantagens para somente leitura ou escrita infrequente (evita cópias com `private` e `firstprivate`)
  - Desvantagens para escrita frequente: condição de corrida ou serialização devido a sincronização
- Coerência de cache com caches privadas
  - Write-through vs. Write-back
    - Bit de sujo (Dirty bit)
    - Bit de inválido (Invalid bit) em múltiplas caches
  - Protocolos de coerência de cache
    - Snooping
    - Baseados em diretórios
  - Variáveis compartilhadas: escrita frequente torna cache inútil e deteriora ainda mais o acesso a memória (dois acessos)
  - Falso compartilhamento: torna a cache inútil mesmo sem compartilhamento de variáveis

MECANISMOS DE SINCRONIZAÇÃO

- Objetivos desse tópico
  - Compreender as diferentes aplicações para os vários mecanismos de sincronização existentes
  - Compreender que esses mecanismos têm sobrecargas diferentes no desempenho dos programas

MECANISMOS DE SINCRONIZAÇÃO

- `#pragma omp critical (nome)`
- Locks explícitos
  - `omp_lock_t lock`
  - `omp_init_lock(&lock)`
  - `omp_set_lock(&lock)`
  - `omp_unset_lock(&lock)`
  - `omp_destroy_lock(&lock)`

MECANISMOS DE SINCRONIZAÇÃO

Tarefa 9:
Escreva um programa que cria tarefas para realizar N inserções em duas listas encadeadas, cada uma associada a uma thread. Cada thread deve escolher aleatoriamente em qual lista inserir um número. Garanta a integridade das listas evitando condição de corrida e, sempre que possível, use regiões críticas nomeadas para que a inserção em uma lista não bloqueie a outra. Em seguida, generalize o programa para um número de listas definido pelo usuário. Explique por que, nesse caso, regiões críticas nomeadas não são suficientes e por que o uso de locks explícitos se torna necessário.

- `#pragma omp critical (nome)`
- Locks explícitos
  - `omp_lock_t lock`
  - `omp_init_lock(&lock)`
  - `omp_set_lock(&lock)`
  - `omp_unset_lock(&lock)`
  - `omp_destroy_lock(&lock)`

MECANISMOS DE SINCRONIZAÇÃO

- `#pragma omp atomic`
- A cláusula `reduction`

MECANISMOS DE SINCRONIZAÇÃO

Tarefa 10:
Implemente novamente o estimador da tarefa 8 que usa um contador compartilhado e o `rand_r` substituindo o `#pragma omp critical` pelo `#pragma omp atomic`. Compare essas duas implementações com suas versões que usam contadores privados. Agora, compare essas com uma 5ª versão que utiliza apenas a cláusula `reduction` ao invés das diretivas de sincronização. Reflita sobre a aplicabilidade de desses mecanismos em termos de desempenho e produtividade e proponha um roteiro para quando utilizar qual mecanismo de sincronização, incluindo `critical` nomeadas e *locks* explícitos.

- `#pragma omp atomic`
- A cláusula `reduction`

PARTICIONAMENTO DE DADOS E BALANCEAMENTO DE CARGA

- Objetivos desse tópico:
  - Conhecer os diversos métodos de particionamento de dados e balanceamento de carga

PARTICIONAMENTO DE DADOS E BALANCEAMENTO DE CARGA

- `#pragma omp sections`
- Cláusula schedule
  - `static`
  - `dynamic`
  - `guided`
  - `chunksize`
- A cláusula `collapse`
- `#pragma omp simd`

PARTICIONAMENTO DE DADOS E BALANCEAMENTO DE CARGA

- A equação de Navier-Stokes
  - Descreve como um fluido se move, acelera e se deforma por causa da pressão, da viscosidade e de forças externas.
- Aplicações práticas
  - Movimento de ar (vento, turbulência atmosférica).
  - Fluxo de água (rios, oceanos, tubulações).
  - Aerodinâmica (aviões, carros).
  - Medicina (fluxo sanguíneo).
  - Climatologia (modelagem do clima).
  - Engenharia (motores, bombas, ventilação).

PARTICIONAMENTO DE DADOS E BALANCEAMENTO DE CARGA

Tarefa 11:
Escreva um código que simule o movimento de um fluido ao longo do tempo usando a equação de Navier-Stokes, considerando apenas os efeitos da viscosidade. Desconsidere a pressão e quaisquer forças externas. Utilize diferenças finitas para discretizar o espaço e simule a evolução da velocidade do fluido no tempo. Inicialize o fluido parado ou com velocidade constante e verifique se o campo permanece estável. Em seguida, crie uma pequena perturbação e observe se ela se difunde suavemente. Após validar o código, paralelize-o com OpenMP e explore o impacto das cláusulas `schedule` e `collapse` no desempenho da execução paralela.

- A equação de Navier-Stokes
  - Descreve como um fluido se move, acelera e se deforma por causa da pressão, da viscosidade e de forças externas.
- Aplicações práticas
  - Movimento de ar (vento, turbulência atmosférica).
  - Fluxo de água (rios, oceanos, tubulações).
  - Aerodinâmica (aviões, carros).
  - Medicina (fluxo sanguíneo).
  - Climatologia (modelagem do clima).
  - Engenharia (motores, bombas, ventilação).

AVALIAÇÃO DE DESEMPENHO E ESCALABILIDADE PARALELA

- Objetivos desse tópico:
  - Entender a diferença entre desempenho e escalabilidade
  - Aprender a identificar pontos críticos e gargalos de escalabilidade

AVALIAÇÃO DE DESEMPENHO E ESCALABILIDADE PARALELA

- Desempenho x Escalabilidade



- O desempenho está relacionado à rapidez com que a tarefa é executada
- A escalabilidade está relacionada à eficiência com que a tarefa é executada

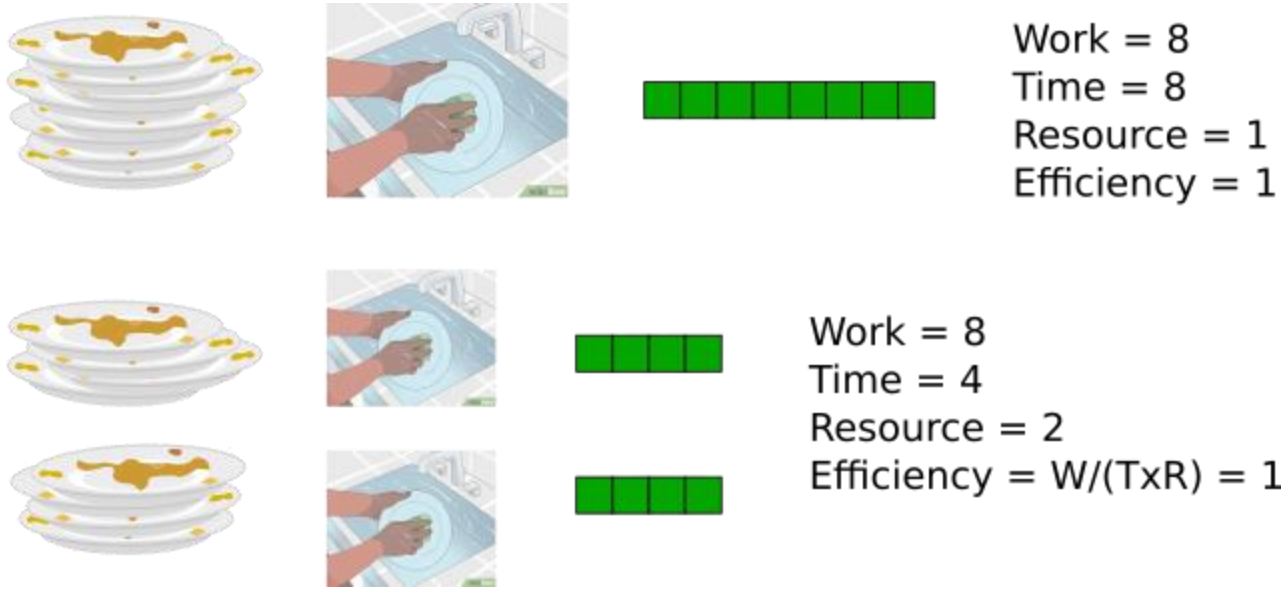
AVALIAÇÃO DE DESEMPENHO E ESCALABILIDADE PARALELA

- Desempenho é o inverso do Tempo, isto é:
  - Desempenho = 1/Tempo
- Eficiência é quanto Trabalho é realizado por Recurso, isto é:
  - Eficiência = Trabalho x Desempenho/Recursos; ou
  - Eficiência = Trabalho/(Tempo x Recursos)

Work = 1
Time = 1
Resource = 1
Efficiency = 1

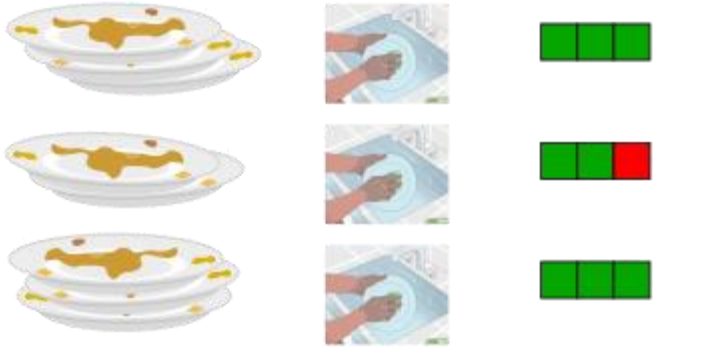
AVALIAÇÃO DE DESEMPENHO E ESCALABILIDADE PARALELA

- A primeira tarefa é sequencial e a segunda é paralela. Ambas performam 100% de eficiência.

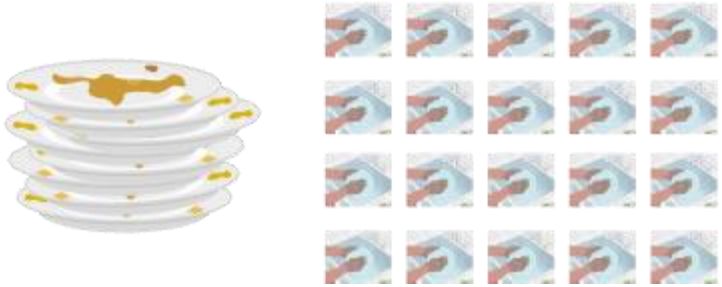


AVALIAÇÃO DE DESEMPENHO E ESCALABILIDADE PARALELA

- Ampliar os recursos por si só muitas vezes prejudica a eficiência.



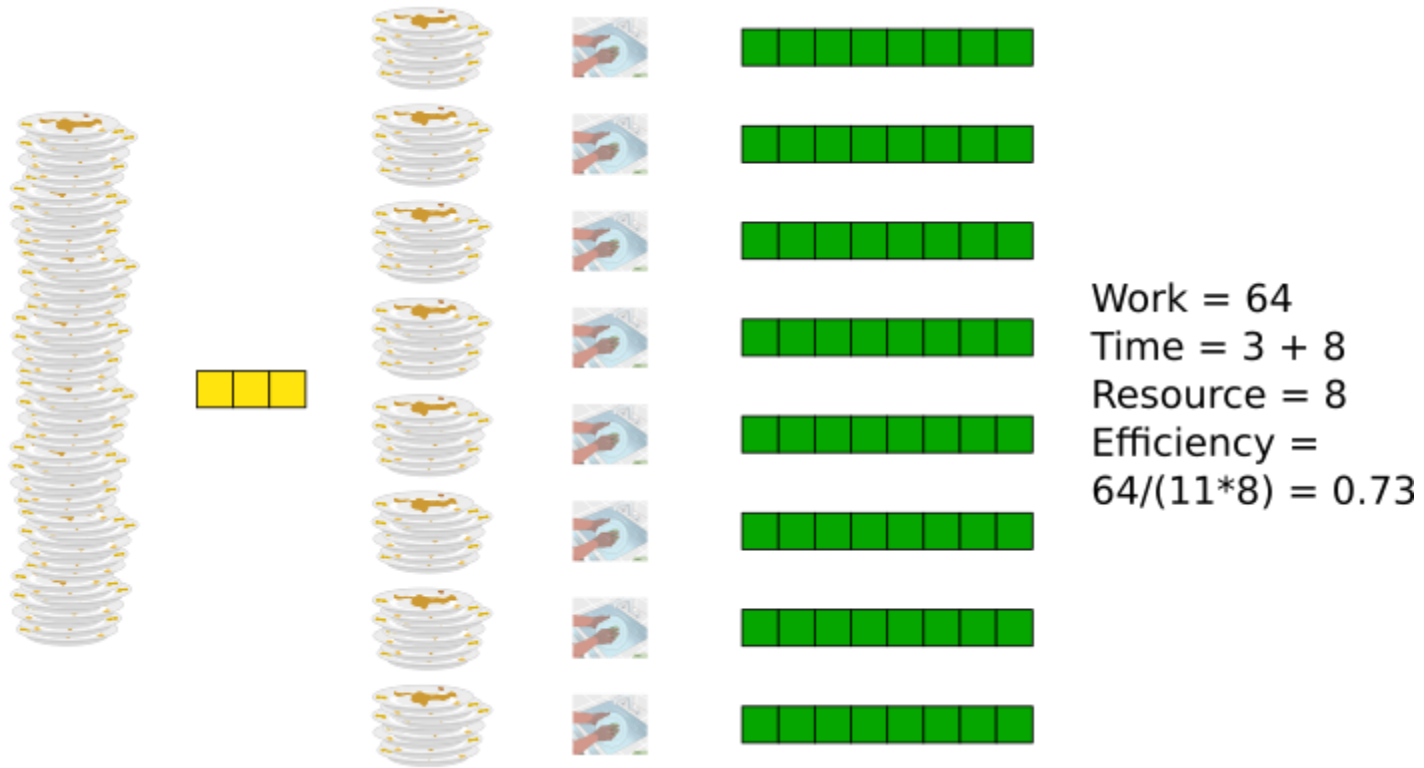
Work = 8
Time = 3
Resource = 3
Efficiency = 8/(3x3) = 0.89



Work = 8
Time = 1
Resource = 20
Efficiency = 8/(1x20) = 0.4

AVALIAÇÃO DE DESEMPENHO E ESCALABILIDADE PARALELA

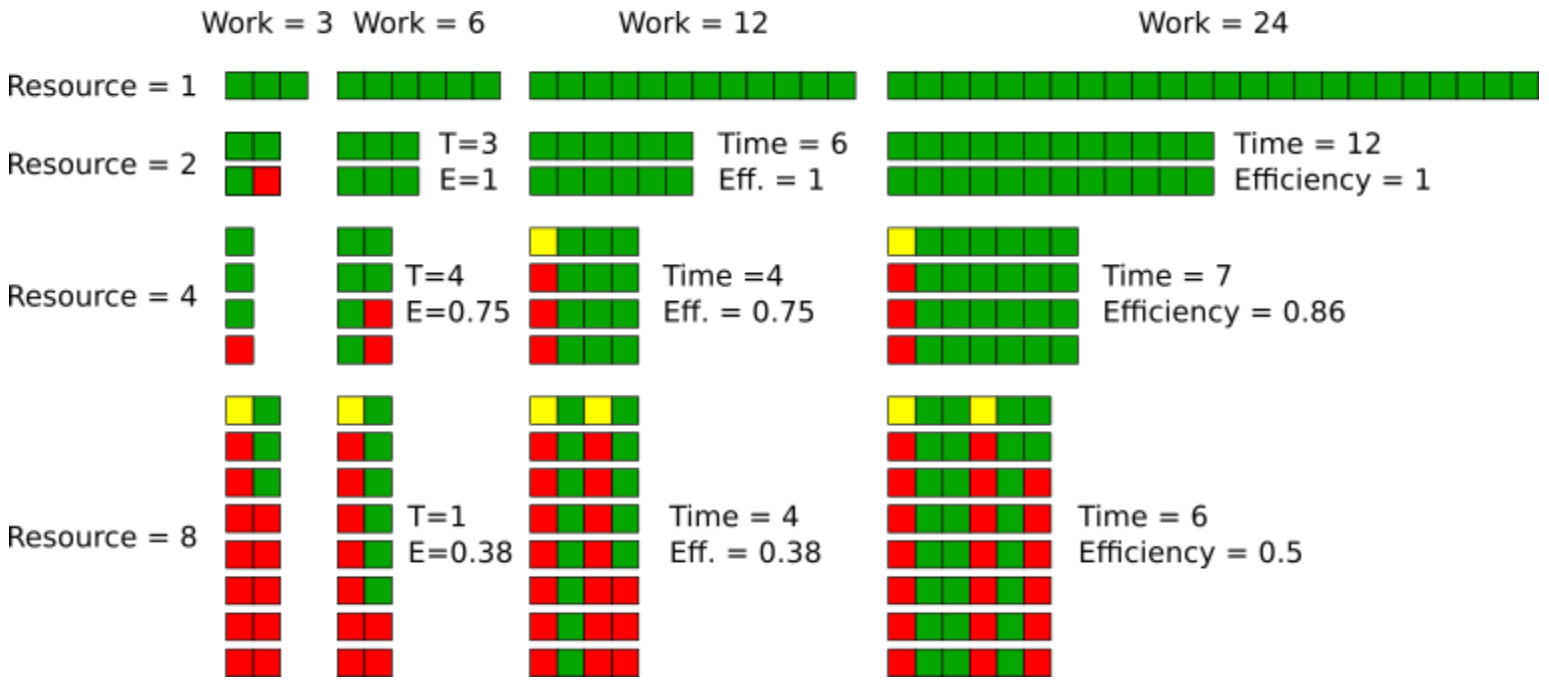
- Máquinas maiores, como supercomputadores, exigem problemas maiores para manter a eficiência.



- Sobrecarga de paralelização, dependências, serialização, etc. também prejudicam a eficiência.

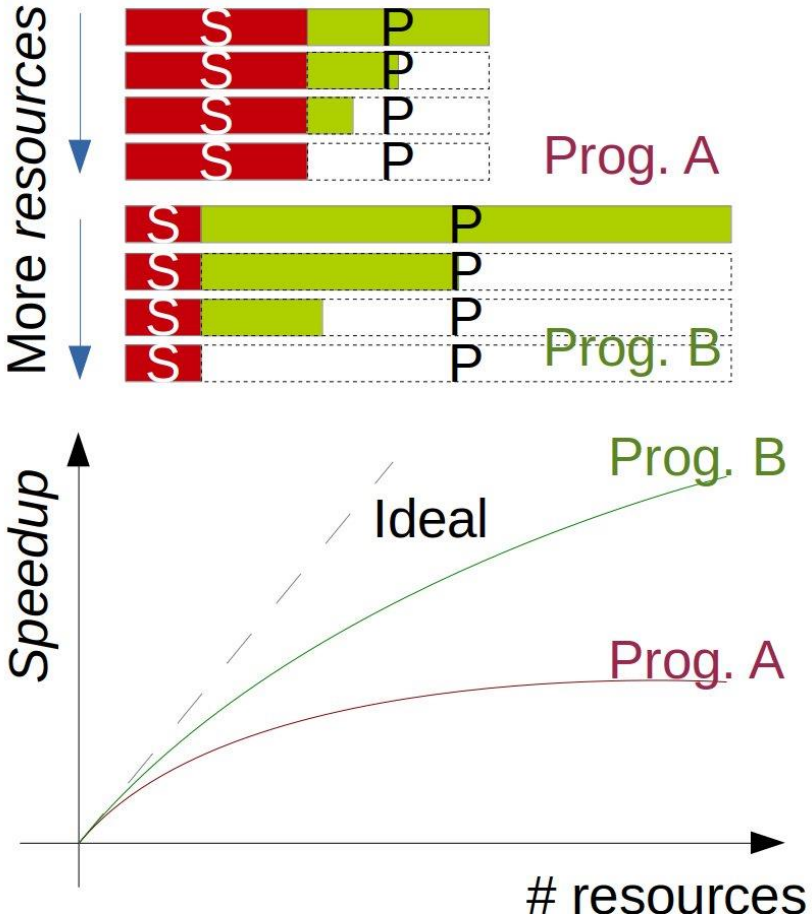
AVALIAÇÃO DE DESEMPENHO E ESCALABILIDADE PARALELA

- A eficiência pode variar de acordo com o escalonamento de recursos e de problemas.



AVALIAÇÃO DE DESEMPENHO E ESCALABILIDADE PARALELA

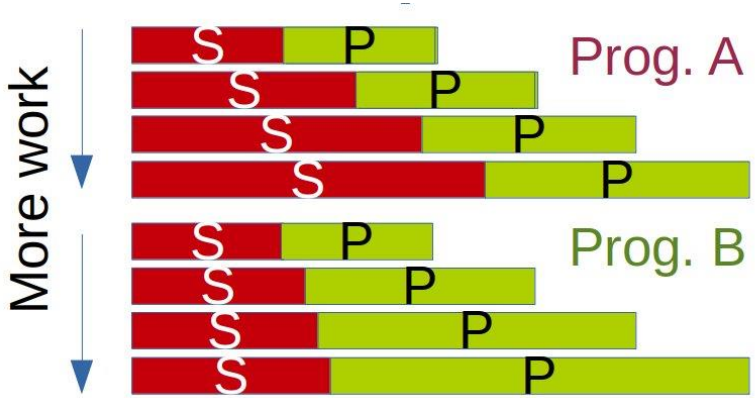
- Modelos de `Speedup`
  - Modelo de Amdahl

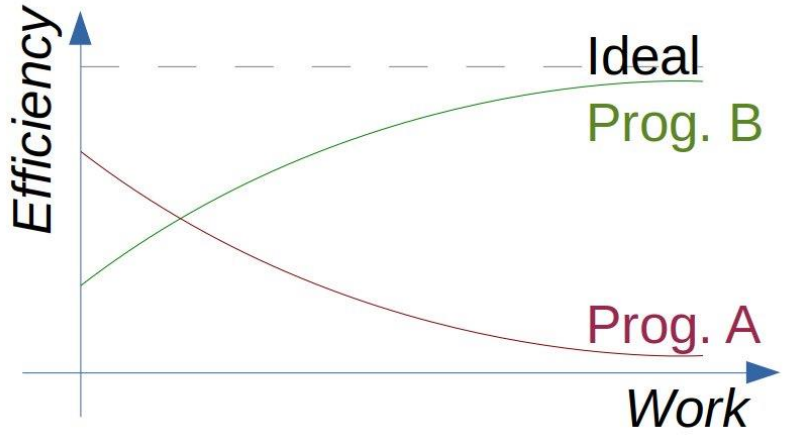


- `Speedup(p) = T(1)/T(p)`

AVALIAÇÃO DE DESEMPENHO E ESCALABILIDADE PARALELA

- Modelos de `Speedup`
  - Modelo de Gustafson

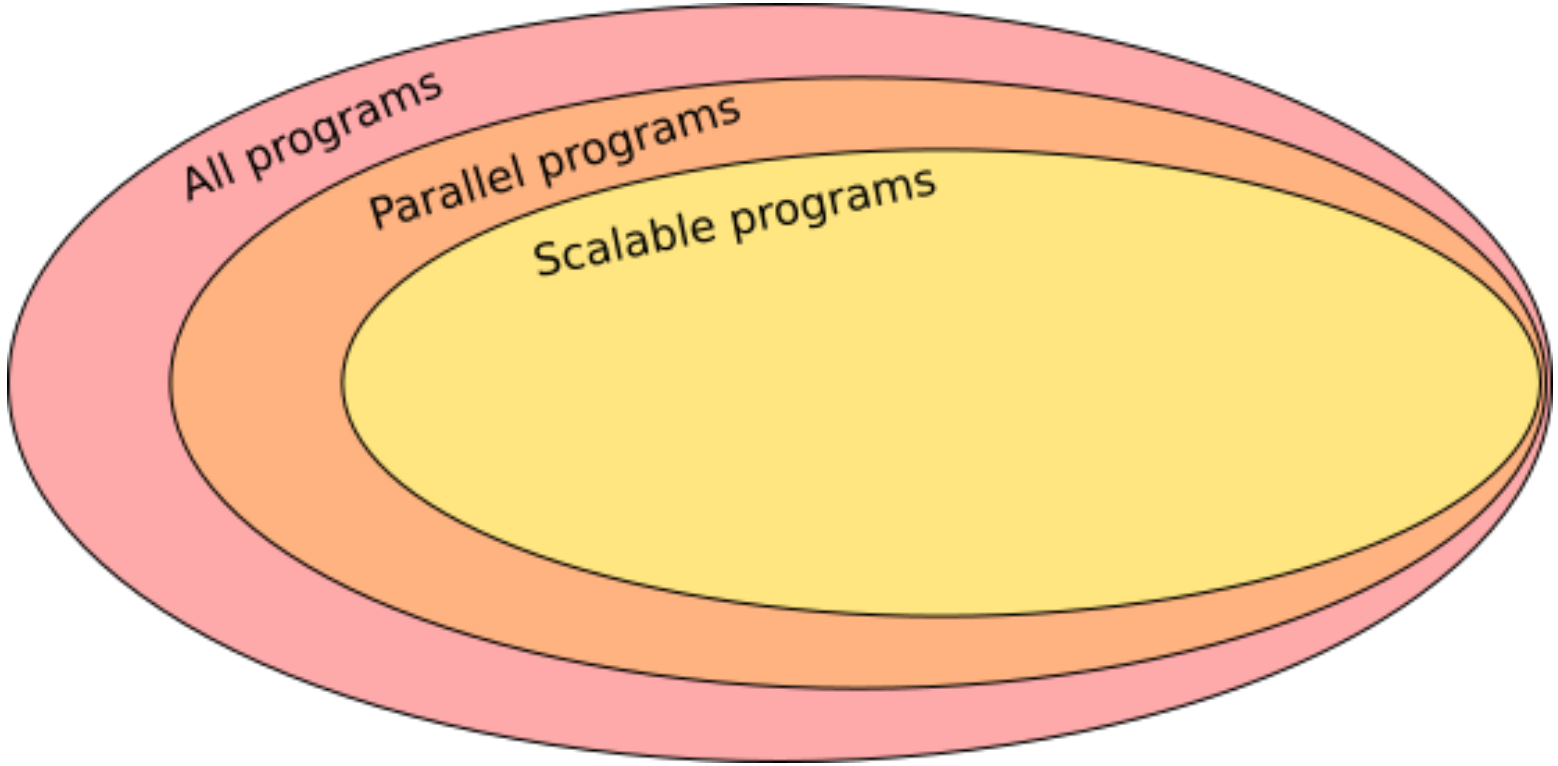




- `Eficiência(p) = S(p)/p`

AVALIAÇÃO DE DESEMPENHO E ESCALABILIDADE PARALELA

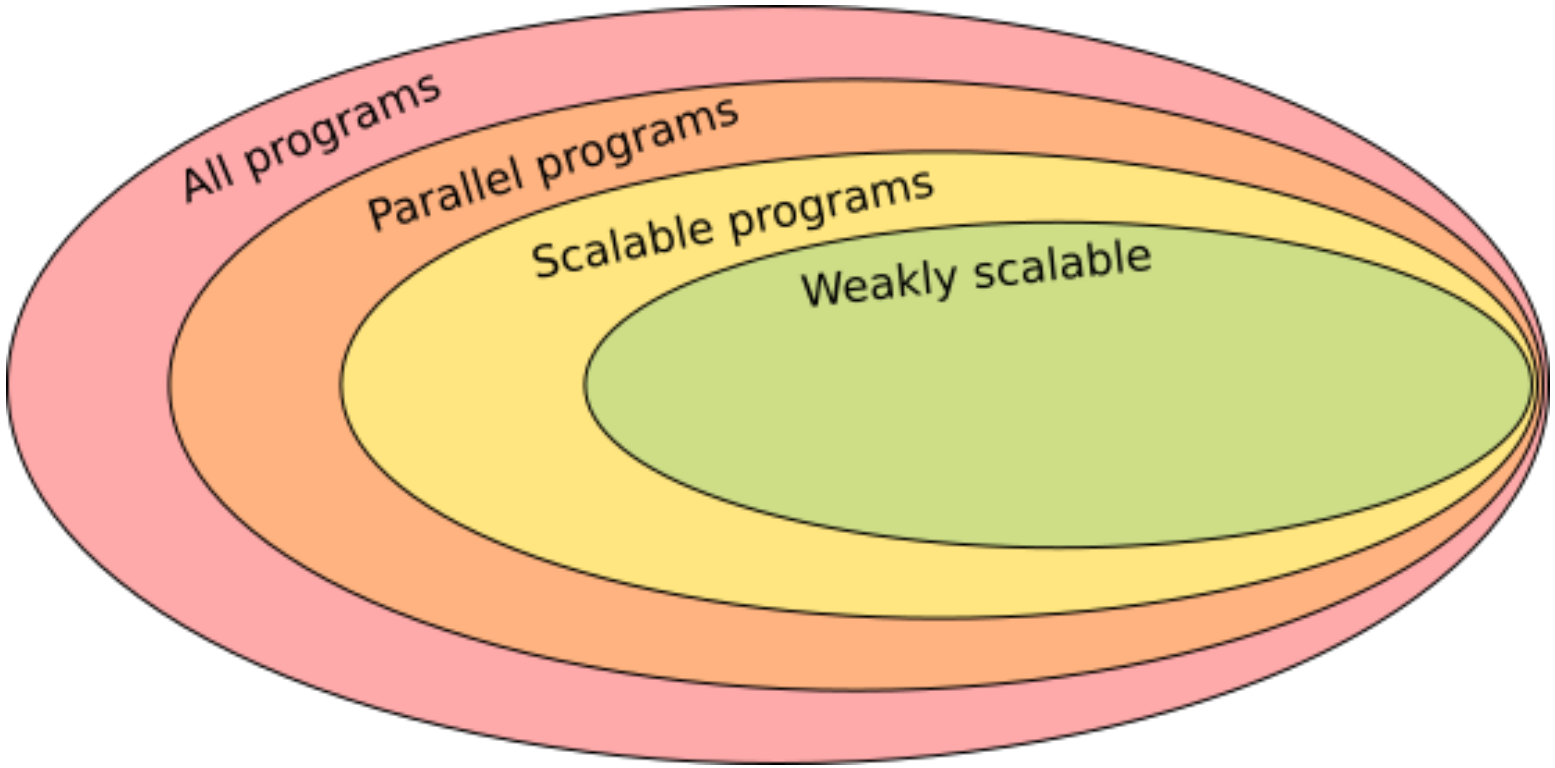
- Análise de Eficiência Paralela e Escalabilidade



- **Escalável**: mantém a eficiência para mais recursos com um problema maior de qualquer fator.

AVALIAÇÃO DE DESEMPENHO E ESCALABILIDADE PARALELA

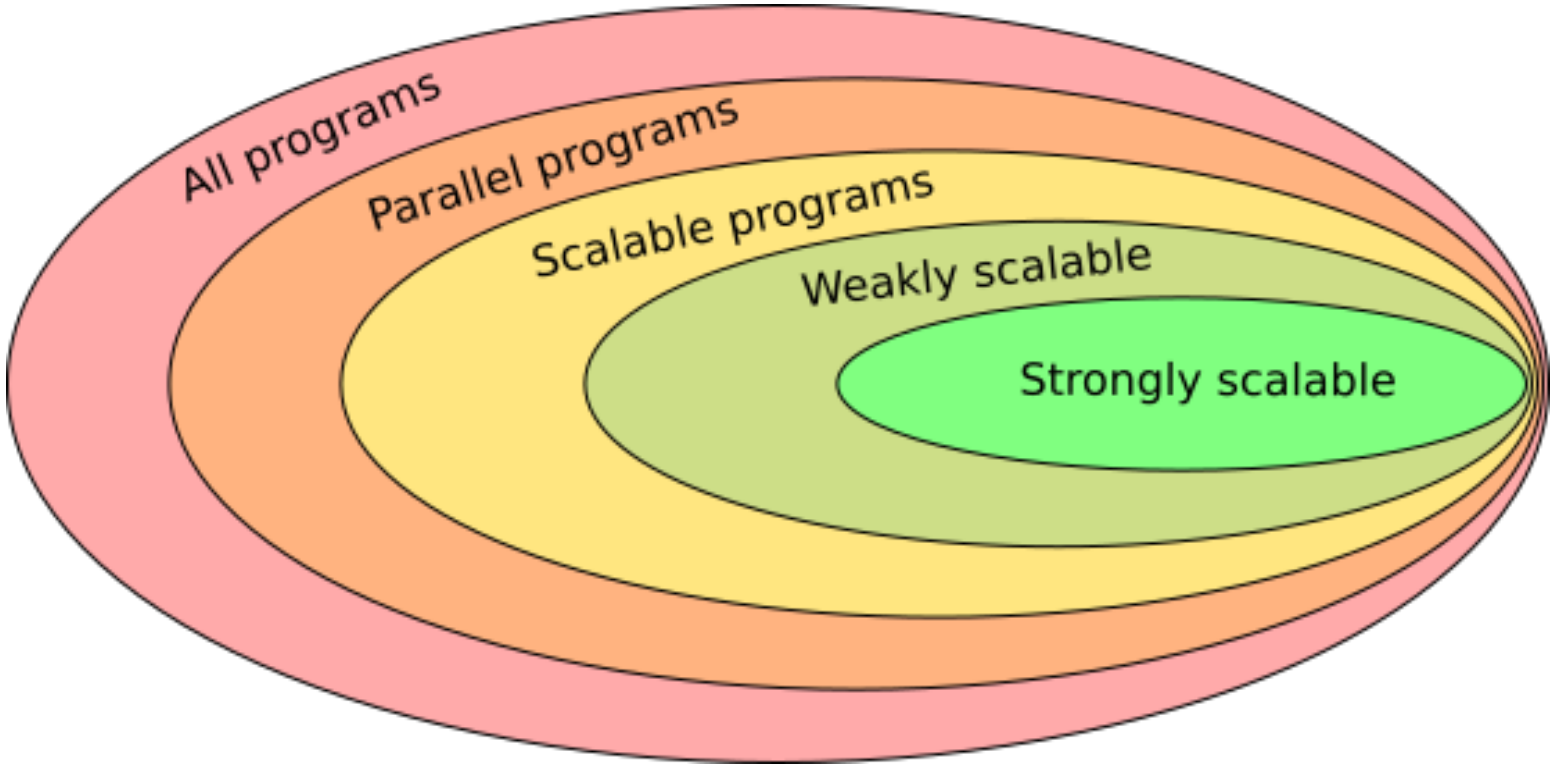
- Análise de Eficiência Paralela e Escalabilidade



- **Fracamente Escalável**: mantém a eficiência mesmo quando o tamanho do problema cresce menos do que os recursos.

AVALIAÇÃO DE DESEMPENHO E ESCALABILIDADE PARALELA

- Análise de Eficiência Paralela e Escalabilidade



- **Fortemente Escalável**: mantém a eficiência mesmo sem aumentar o tamanho do problema.

AVALIAÇÃO DE DESEMPENHO E ESCALABILIDADE PARALELA

- Avaliação da escalabilidade de um programa com tabelas

Timings

# cores \ Problem size	1x	2x	4x	8X	16X	32X
1	100,00	400,00	1600,00	6400,00	25600,00	102400,00
2	51,00	201,00	801,00	3201,00	12801,00	51201,00
4	27,00	102,00	402,00	1602,00	6402,00	25602,00
8	15,50	53,00	203,00	803,00	3203,00	12803,00
16	10,25	29,00	104,00	404,00	1604,00	6404,00
32	8,13	17,50	55,00	205,00	805,00	3205,00
64	7,56	12,25	31,00	106,00	406,00	1606,00
128	7,78	10,13	19,50	57,00	207,00	807,00

$$T_1 = n^2 \qquad T_P = \frac{n^2}{p} + log_2 p$$

AVALIAÇÃO DE DESEMPENHO E ESCALABILIDADE PARALELA

- Avaliação da escalabilidade de um programa com tabelas

Speedups

# cores \ Problem size	1x	2x	4x	8X	16X	32X
1	1,00	1,00	1,00	1,00	1,00	1,00
2	1,96	1,99	2,00	2,00	2,00	2,00
4	3,70	3,92	3,98	4,00	4,00	4,00
8	6,45	7,55	7,88	7,97	7,99	8,00
16	9,76	13,79	15,38	15,84	15,96	15,99
32	12,31	22,86	29,09	31,22	31,80	31,95
64	13,22	32,65	51,61	60,38	63,05	63,76
128	12,85	39,51	82,05	112,28	123,67	126,89

$$T_1 = n^2 \qquad T_P = \frac{n^2}{p} + log_2 p$$

$$S = \frac{T_1}{T_p}$$

AVALIAÇÃO DE DESEMPENHO E ESCALABILIDADE PARALELA

- Avaliação da escalabilidade de um programa com tabelas

Efficiencies

# cores \ Problem size	1x	2x	4x	8X	16X	32X
1	1,00	1,00	1,00	1,00	1,00	1,00
2	0,98	1,00	1,00	1,00	1,00	1,00
4	0,93	0,98	1,00	1,00	1,00	1,00
8	0,81	0,94	0,99	1,00	1,00	1,00
16	0,61	0,86	0,96	0,99	1,00	1,00
32	0,38	0,71	0,91	0,98	0,99	1,00
64	0,21	0,51	0,81	0,94	0,99	1,00
128	0,10	0,31	0,64	0,88	0,97	0,99

$$T_1 = n^2 \qquad T_P = \frac{n^2}{p} + log_2 p$$

$$S = \frac{T_1}{T_p} \qquad E = \frac{S}{p}$$

AVALIAÇÃO DE DESEMPENHO E ESCALABILIDADE PARALELA

- Avaliação da escalabilidade de um programa com tabelas

Efficiencies

# cores \ Problem size	1x	2x	4x	8X	16X	32X
1	1,00	1,00	1,00	1,00	1,00	1,00
2	0,98	1,00	1,00	1,00	1,00	1,00
4	0,93	0,98	1,00	1,00	1,00	1,00
8	0,81	0,94	0,99	1,00	1,00	1,00
16	0,61	0,86	0,96	0,99	1,00	1,00
32	0,38	0,71	0,91	0,98	0,99	1,00
64	0,21	0,51	0,81	0,94	0,99	1,00
128	0,10	0,31	0,64	0,88	0,97	0,99

$$T_1 = n^2 \qquad T_P = \frac{n^2}{p} + log_2 p$$

$$S = \frac{T_1}{T_p} \qquad E = \frac{S}{p}$$

Escalável? Fracamente? Fortemente?

A análise ainda é possível com um número pequeno de núcleos e taxas quadraticamente crescentes

AVALIAÇÃO DE DESEMPENHO E ESCALABILIDADE PARALELA

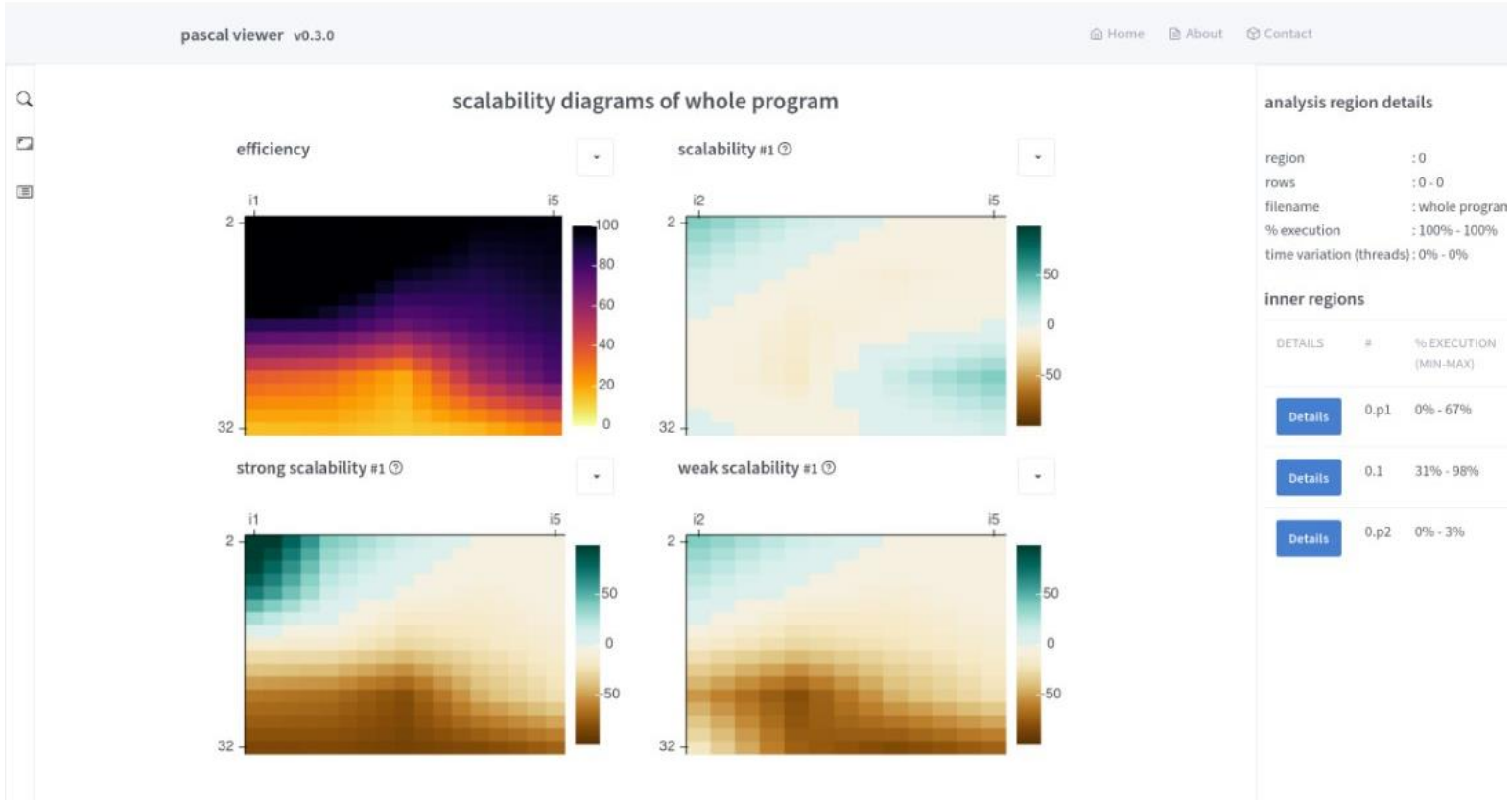
- Avaliação da escalabilidade de um programa com tabelas

# cores \ Problem size	1x	2x	3x	4x	5x	6x	7x	8x	9x	10x	11x	12x	13x	14x	15x	16x	17x	18x	19x	20x	21x	22x	23x	24x	25x	26x	27x	28x	29x
1	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	11	21	31	41	51	61	71	81	91	101	111	121	131	141	151	161	171	181	191
2	0,98	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00
3	0,95	0,99	0,99	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00
4	0,93	0,98	0,99	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00
5	0,90	0,97	0,99	0,99	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00
6	0,87	0,96	0,98	0,99	0,99	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00
7	0,84	0,95	0,98	0,99	0,99	0,99	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00
8	0,81	0,94	0,97	0,99	0,99	0,99	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00
9	0,78	0,93	0,97	0,98	0,99	0,99	0,99	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00
10	0,75	0,92	0,96	0,98	0,99	0,99	0,99	0,99	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00
11	0,72	0,91	0,96	0,98	0,99	0,99	0,99	0,99	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00
12	0,70	0,90	0,95	0,97	0,98	0,99	0,99	0,99	0,99	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00
13	0,68	0,89	0,95	0,97	0,98	0,99	0,99	0,99	0,99	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00
14	0,65	0,88	0,94	0,97	0,98	0,99	0,99	0,99	0,99	0,99	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00
15	0,63	0,87	0,94	0,96	0,98	0,98	0,99	0,99	0,99	0,99	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00
16	0,61	0,86	0,93	0,96	0,98	0,98	0,99	0,99	0,99	0,99	0,99	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00
17	0,59	0,85	0,93	0,96	0,97	0,98	0,99	0,99	0,99	0,99	0,99	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00
18	0,57	0,84	0,92	0,96	0,97	0,98	0,98	0,99	0,99	0,99	0,99	0,99	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00
19	0,55	0,83	0,92	0,95	0,97	0,98	0,98	0,99	0,99	0,99	0,99	0,99	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00
20	0,54	0,82	0,91	0,95	0,97	0,98	0,98	0,99	0,99	0,99	0,99	0,99	0,99	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00
21	0,52	0,81	0,91	0,95	0,96	0,98	0,98	0,99	0,99	0,99	0,99	0,99	0,99	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00
22	0,50	0,80	0,90	0,94	0,96	0,97	0,98	0,98	0,99	0,99	0,99	0,99	0,99	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00
23	0,49	0,79	0,90	0,94	0,96	0,97	0,98	0,98	0,99	0,99	0,99	0,99	0,99	0,99	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00
24	0,48	0,78	0,89	0,94	0,96	0,97	0,98	0,98	0,99	0,99	0,99	0,99	0,99	0,99	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00
25	0,46	0,78	0,89	0,93	0,96	0,97	0,98	0,98	0,99	0,99	0,99	0,99	0,99	0,99	0,99	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00
26	0,45	0,77	0,88	0,93	0,95	0,97	0,98	0,98	0,99	0,99	0,99	0,99	0,99	0,99	0,99	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00
27	0,44	0,76	0,88	0,93	0,95	0,97	0,97	0,98	0,98	0,99	0,99	0,99	0,99	0,99	0,99	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00
28	0,43	0,75	0,87	0,92	0,95	0,96	0,97	0,98	0,98	0,99	0,99	0,99	0,99	0,99	0,99	0,99	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00
29	0,42	0,74	0,86	0,92	0,95	0,96	0,97	0,98	0,98	0,99	0,99	0,99	0,99	0,99	0,99	0,99	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00
30	0,40	0,73	0,86	0,92	0,94	0,96	0,97	0,98	0,98	0,99	0,99	0,99	0,99	0,99	0,99	0,99	0,99	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00
31	0,39	0,72	0,85	0,91	0,94	0,96	0,97	0,98	0,98	0,98	0,99	0,99	0,99	0,99	0,99	0,99	0,99	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00
32	0,38	0,71	0,85	0,91	0,94	0,96	0,97	0,98	0,98	0,98	0,99	0,99	0,99	0,99	0,99	0,99	0,99	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00

¼ da tabela do slide anterior com taxas lineares

AVALIAÇÃO DE DESEMPENHO E ESCALABILIDADE PARALELA

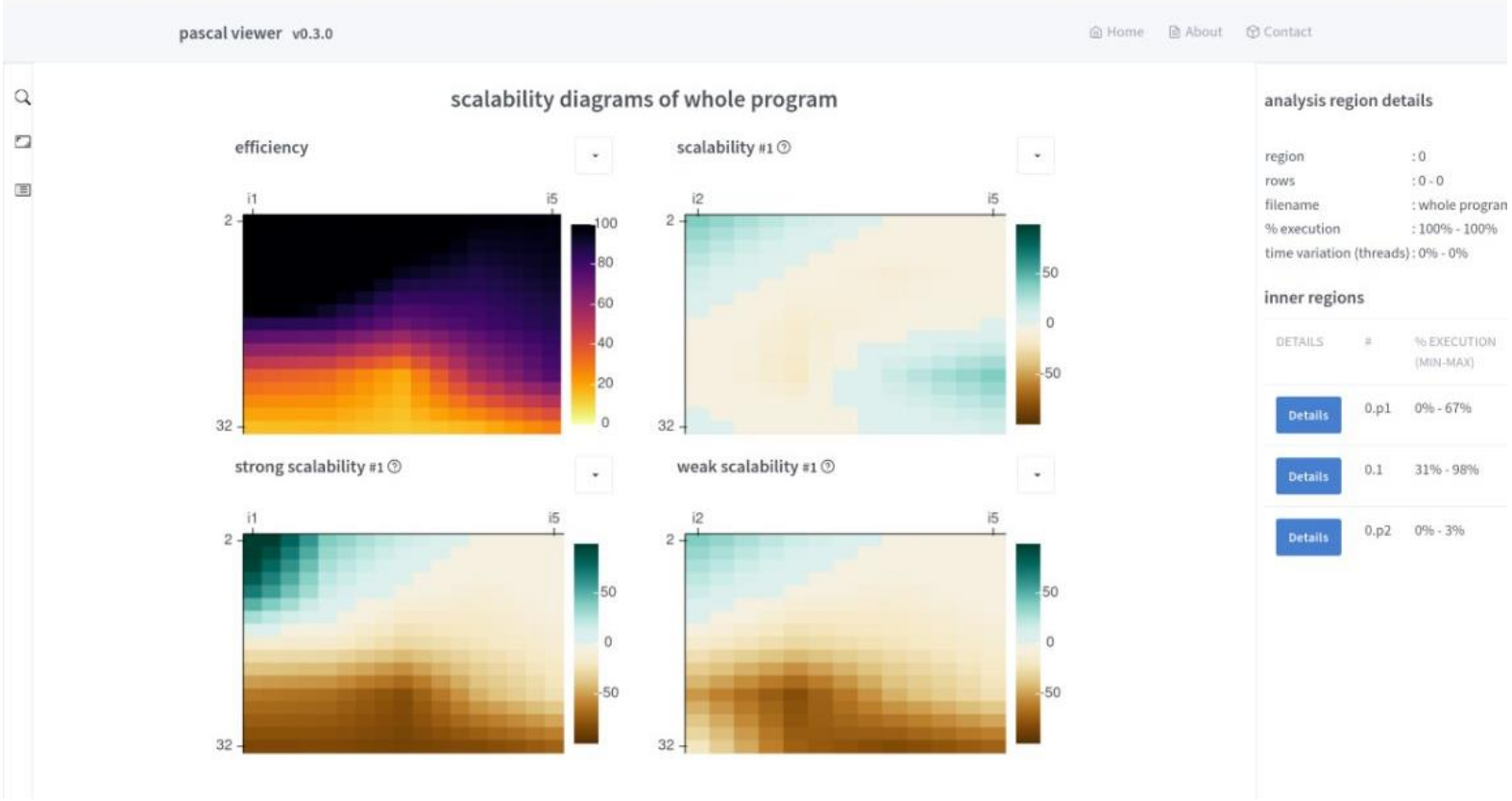
- Visualização da escalabilidade de um programa



AVALIAÇÃO DE DESEMPENHO E ESCALABILIDADE PARALELA

Tarefa 12:
Avalie a escalabilidade do seu código de Navier-Stokes utilizando algum nó de computação no NPAD. Procure identificar gargalos de escalabilidade e reporte o seu progresso em versões sucessivas da evolução do código otimizado. Comente sobre a escalabilidade, a escalabilidade fraca e a escalabilidade fortes das versões.

- Visualização da escalabilidade de um programa

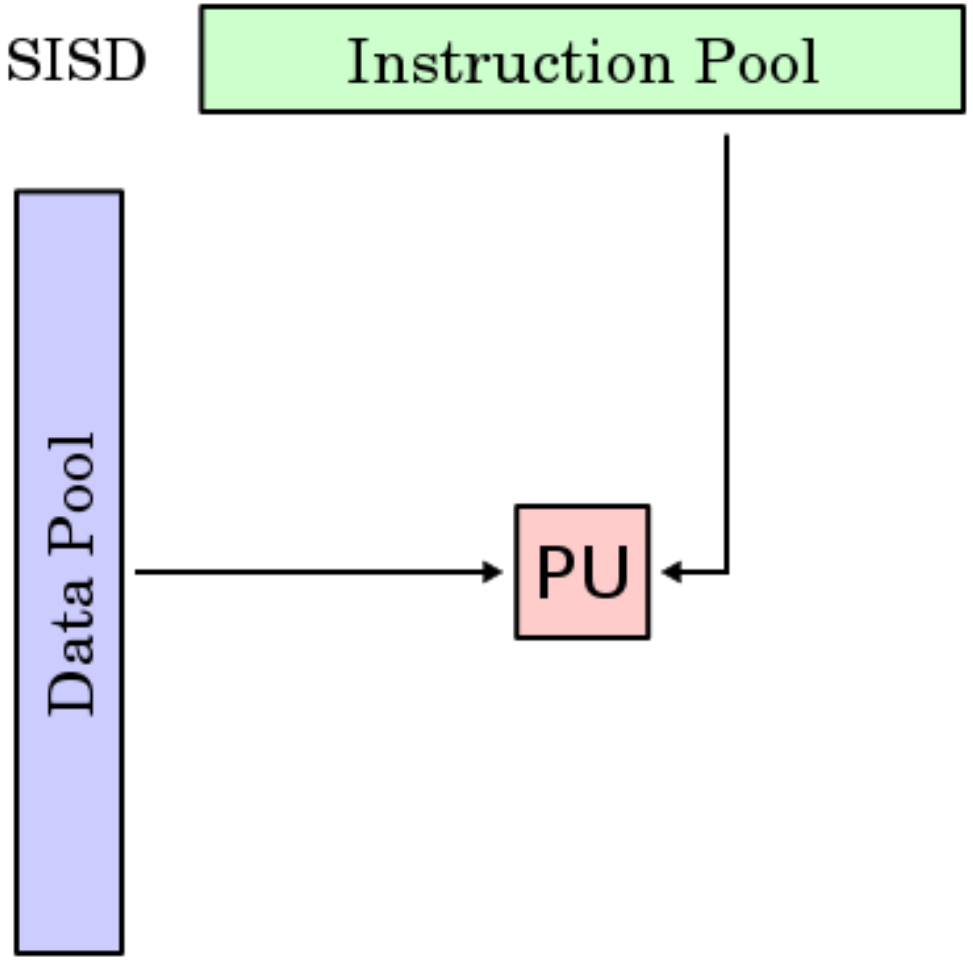


ARQUITETURAS PARALELAS

- Objetivos desse tópico:
  - Identificar as diversas classes de arquiteturas paralelas de hardware definidas pela Taxonomia de Flynn
  - Compreender o impacto e aprender a configurar as diversas formas de afinidades de threads

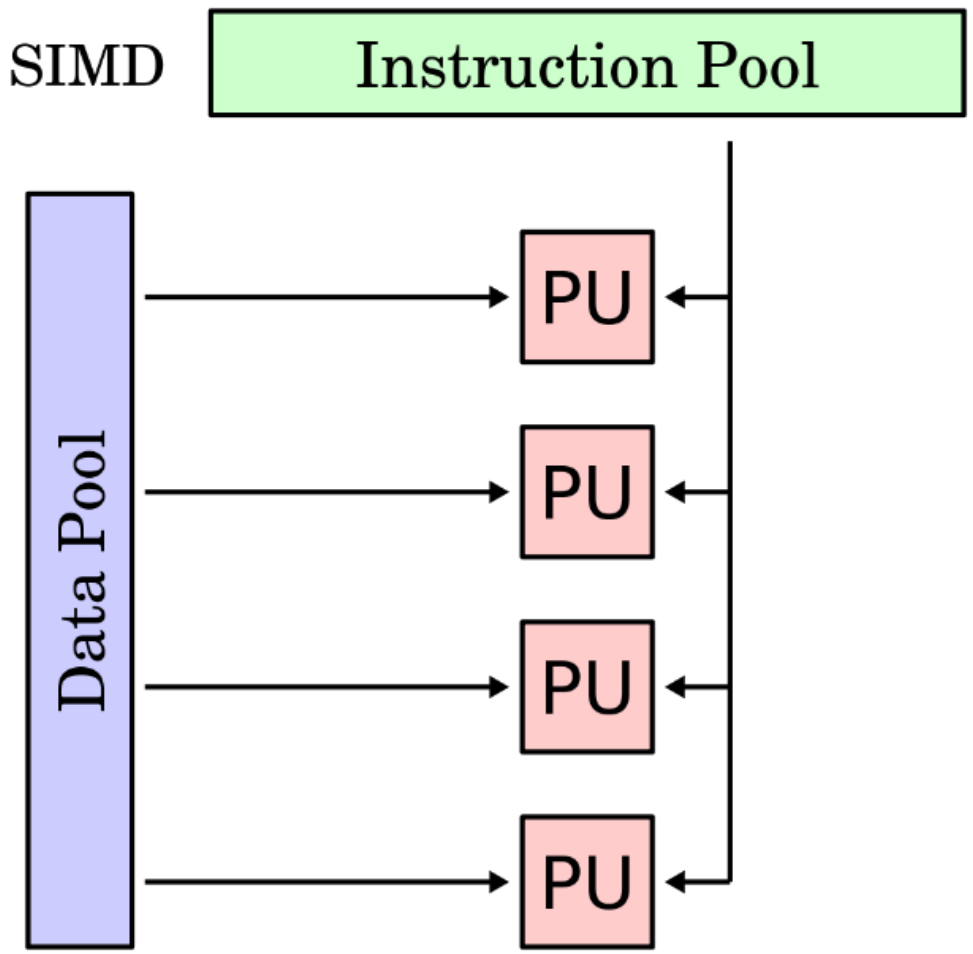
ARQUITETURAS PARALELAS

- Single Instruction Single Data



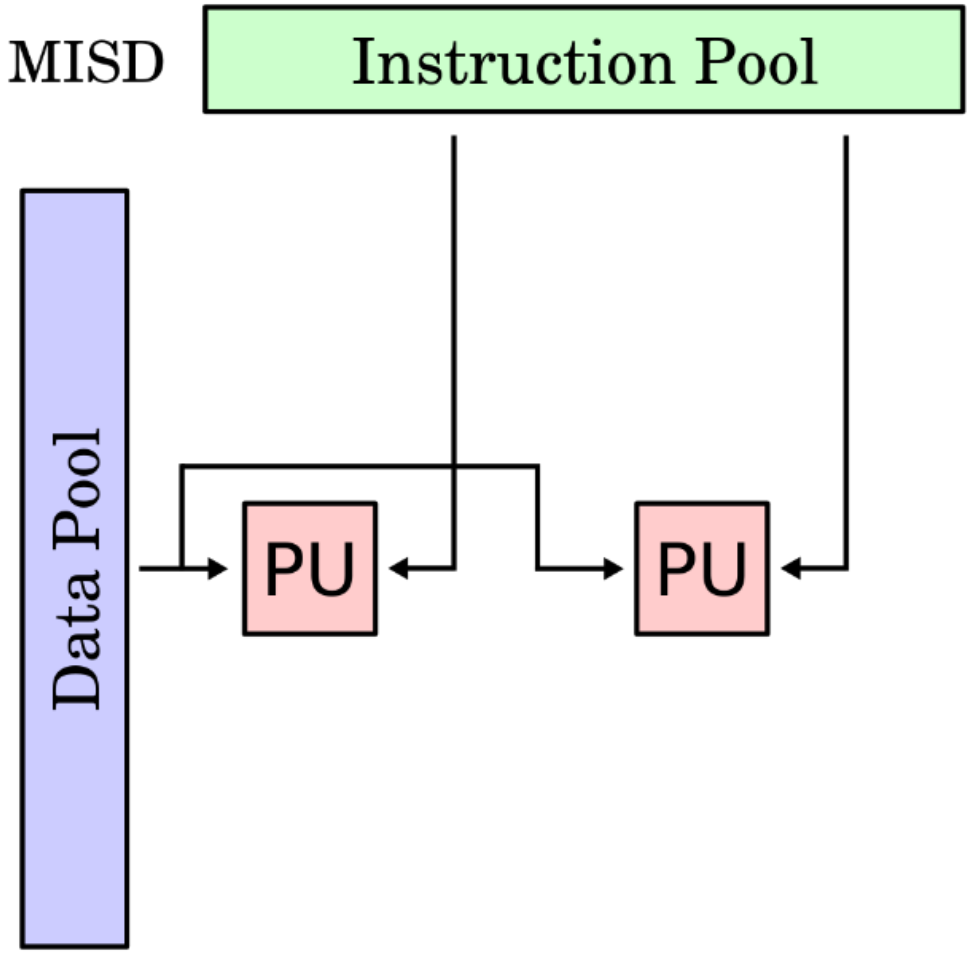
ARQUITETURAS PARALELAS

- Single Instruction Multiple Data



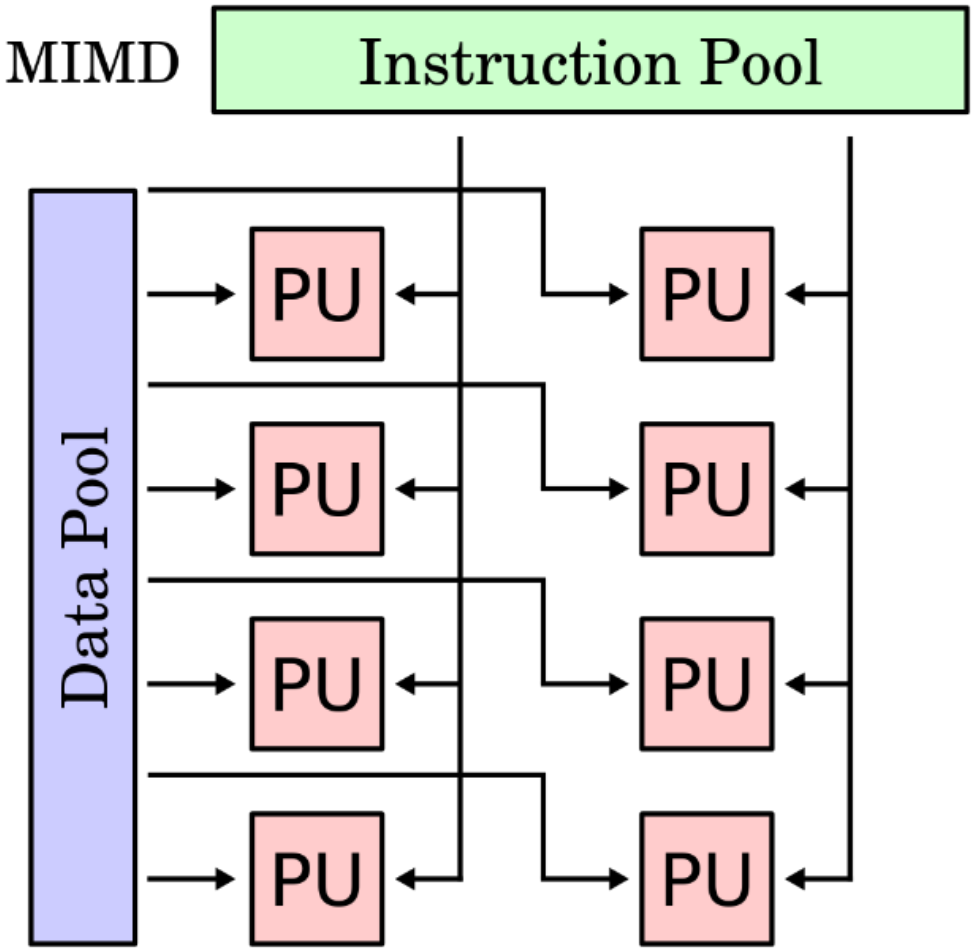
ARQUITETURAS PARALELAS

- Multiple Instruction Single Data



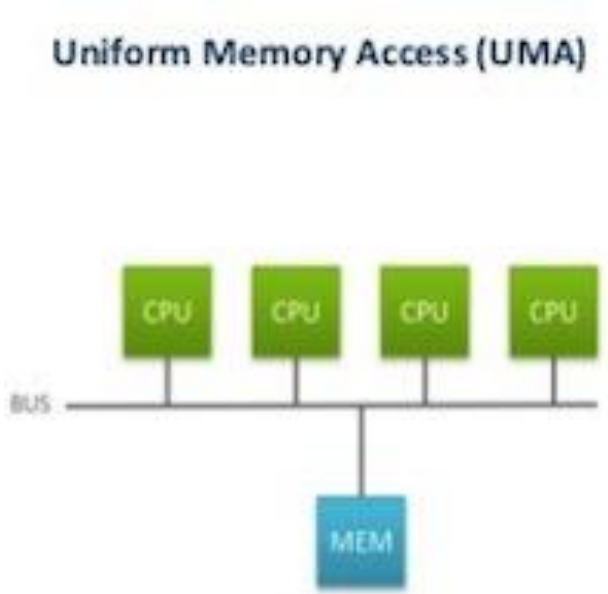
ARQUITETURAS PARALELAS

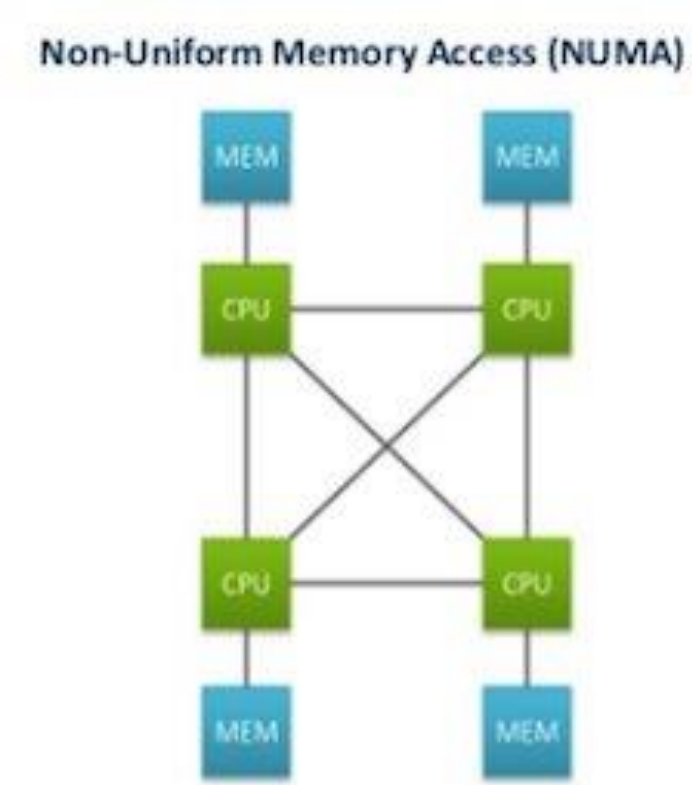
- Multiple Instruction Multiple Data



ARQUITETURAS PARALELAS

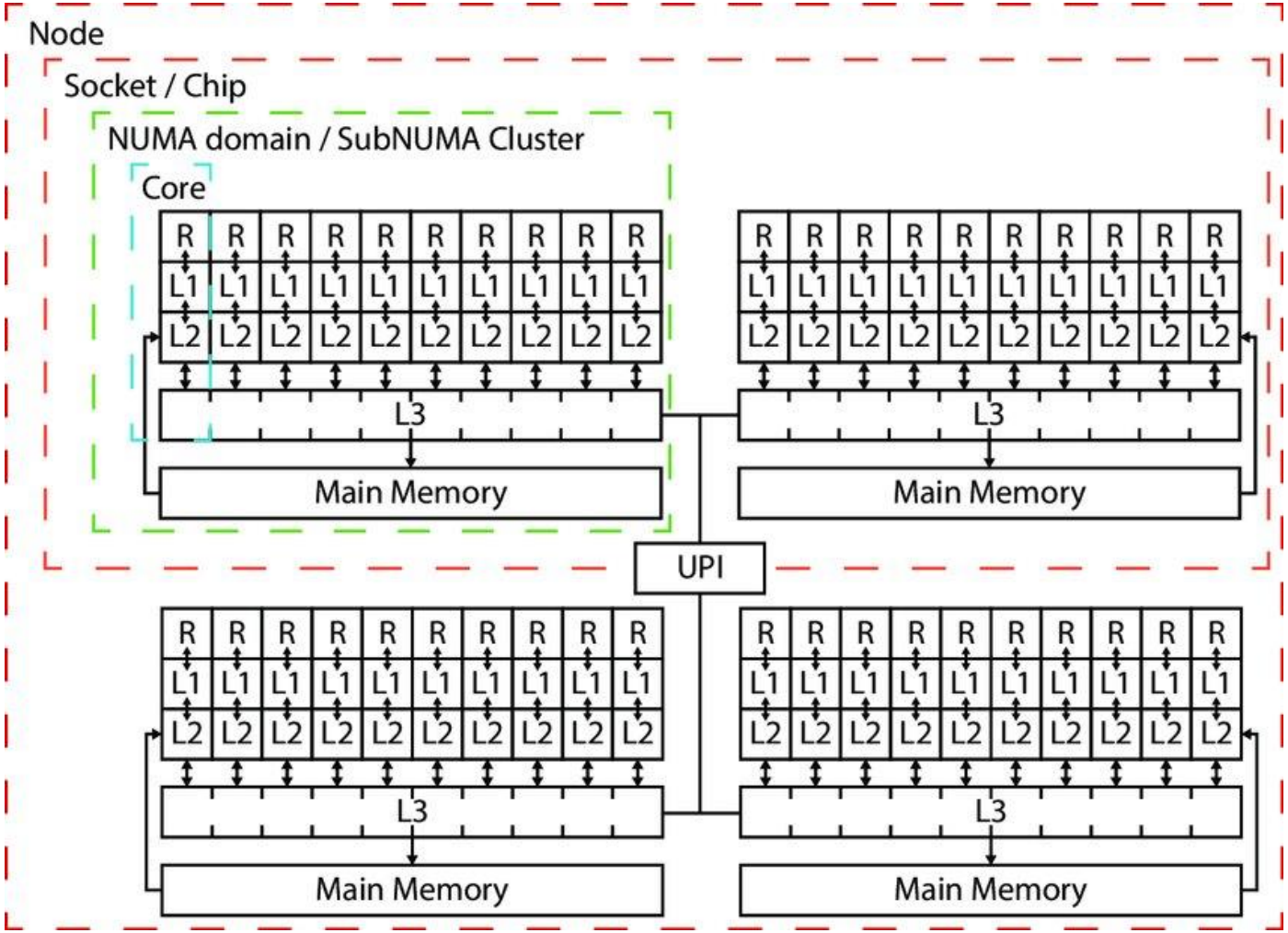
- Arquiteturas típicas MIMD de memória compartilhada:

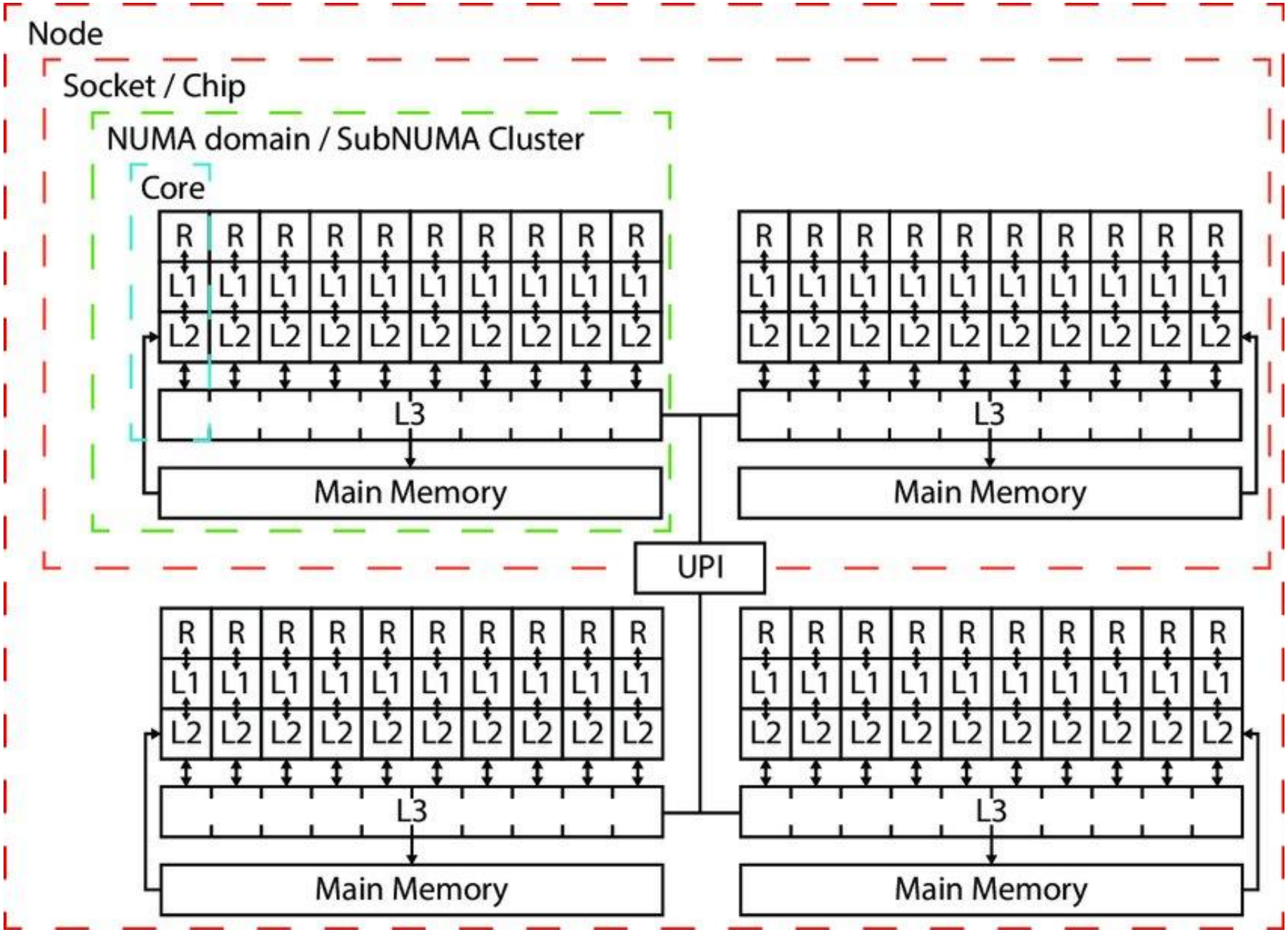




ARQUITETURAS PARALELAS

- Hierarquia de cache em sistemas NUMA:




Node
Socket / Chip
NUMA domain / SubNUMA Cluster
Core
R
L1
L2
L3
Main Memory
UPI

ARQUITETURAS PARALELAS

- Afinidade de threads com OpenMP
  - Exemplos com definição de políticas:
    - `export OMP_PROC_BIND=TRUE`; ou
    - `#pragma omp parallel proc_bind(spread)`
  - Valores comuns:
    - `false`: sem afinidade (pode migrar entre núcleos).
    - `true`: thread fica presa ao núcleo onde foi criada.
    - `close`: threads próximas da master thread, mas distribuídas.
    - `spread`: threads espalhadas pelos núcleos (boa para usar múltiplos sockets).
    - `master`: todas seguem a thread 0 (pode forçar várias threads no mesmo núcleo).
  - Para mais controle, use `OMP_PLACES`, por exemplo:
    - `OMP_PLACES=cores`
    - `OMP_PLACES="{0:8:1},{8:8:1}"`

ARQUITETURAS PARALELAS

Tarefa 13:
Avalie como a escalabilidade do seu código de Navier-Stokes muda ao utilizar os diversos tipos de afinidades de threads suportados pelo sistema operacional e pelo OpenMP no mesmo nó de computação do NPAD que utilizou para a tarefa 12.

- Afinidade de threads com OpenMP
  - Exemplos com definição de políticas:
    - `export OMP_PROC_BIND=TRUE`; ou
    - `#pragma omp parallel proc_bind(spread)`
  - Valores comuns:
    - `false`: sem afinidade (pode migrar entre núcleos).
    - `true`: thread fica presa ao núcleo onde foi criada.
    - `close`: threads próximas da master thread, mas distribuídas.
    - `spread`: threads espalhadas pelos núcleos (boa para usar múltiplos sockets).
    - `master`: todas seguem a thread 0 (pode forçar várias threads no mesmo núcleo).
  - Para mais controle, use `OMP_PLACES`, por exemplo:
    - `OMP_PLACES=cores`
    - `OMP_PLACES="{0:8:1},{8:8:1}"`

PROGRAMAÇÃO EM MEMÓRIA DISTRIBUÍDA

- Objetivos desse tópico:
  - Compreender os conceitos básicos de comunicação ponto-a-ponto em MPI e aprender a utilizar as funções principais
  - Compreender os conceitos básicos de comunicação coletiva em MPI e aprender a utilizar as funções principais de comunicação e organização de dados

PROGRAMAÇÃO EM MEMÓRIA DISTRIBUÍDA

- Message Passing Interface:
  - Processos independentes, sem memória compartilhada
- Compilação de programas MPI
  - Implementações diversas
  - Wrappers: `mpicc`, `mpic++`, `mpifort`
- Execução de programas MPI
  - `mpirun`, `mpiexec`, `srun` (SLURM)
    - Sistema de arquivos compartilhados
    - Autenticação automática
  - Ambientes
    - Redes de computadores
      - Machine file
    - Supercomputadores/clusters
      - Gerenciador de recursos (SLURM, PBS etc)

PROGRAMAÇÃO EM MEMÓRIA DISTRIBUÍDA

- Inicialização e finalização:
  - `MPI_Init | MPI_Finalize`
- Identificação de processos:
  - `MPI_Comm_rank | MPI_Comm_size`
- Comunicação ponto-a-ponto:
  - `MPI_Send | MPI_Recv`
  - `MPI_Bsend | MPI_Ssend`

PROGRAMAÇÃO EM MEMÓRIA DISTRIBUÍDA

Tarefa 14:
Implemente um programa MPI com exatamente dois processos. O processo 0 deve enviar uma mensagem ao processo 1, que imediatamente responde com a mesma mensagem. Meça o tempo total de execução de múltiplas trocas consecutivas dessa mensagem, utilizando `MPI_Wtime`. Registre os tempos para diferentes tamanhos, desde mensagens pequenas (como 8 bytes) até mensagens maiores (como 1MB ou mais). Analise graficamente o tempo em função do tamanho da mensagem e identifique os regimes onde a latência domina e onde a largura de banda se torna o fator principal.

- Inicialização e finalização:
  - `MPI_Init | MPI_Finalize`
- Identificação de processos:
  - `MPI_Comm_rank | MPI_Comm_size`
- Comunicação ponto-a-ponto:
  - `MPI_Send | MPI_Recv`
  - `MPI_Bsend | MPI_Ssend`

PROGRAMAÇÃO EM MEMÓRIA DISTRIBUÍDA

- Comunicação ponto-a-ponto não bloqueante
  - Escondendo latência com sobreposição de computação e comunicação
  - Criação de `MPI_Request: MPI_Isend | MPI_Irecv`
  - Consumindo `MPI_Request: MPI_Wait | MPI_Test`

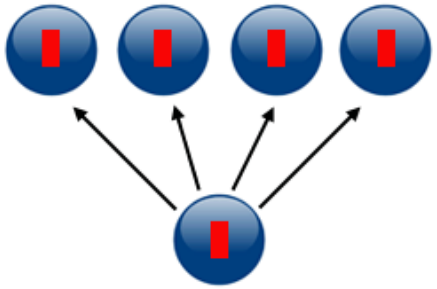
PROGRAMAÇÃO EM MEMÓRIA DISTRIBUÍDA

Tarefa 15:
Implemente uma simulação da difusão de calor em uma barra 1D, dividida entre dois ou mais processos MPI. Cada processo deve simular um trecho da barra com células extras para troca de bordas com vizinhos. Implemente três versões: uma com `MPI_Send/MPI_Recv`, outra com `MPI_Isend/MPI_Irecv` e `MPI_Wait`, e uma terceira usando `MPI_Test` para atualizar os pontos internos enquanto aguarda a comunicação. Compare os tempos de execução e discuta os ganhos com sobreposição de comunicação e computação.

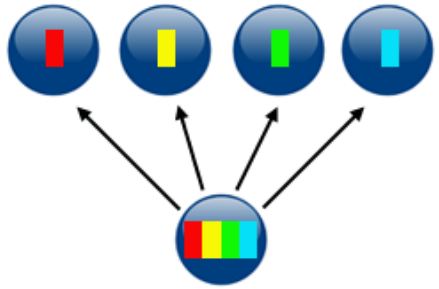
- Comunicação ponto-a-ponto não bloqueante
  - Escondendo latência com sobreposição de computação e comunicação
  - Criação de `MPI_Request: MPI_Isend | MPI_Irecv`
  - Consumindo `MPI_Request: MPI_Wait | MPI_Test`

PROGRAMAÇÃO EM MEMÓRIA DISTRIBUÍDA

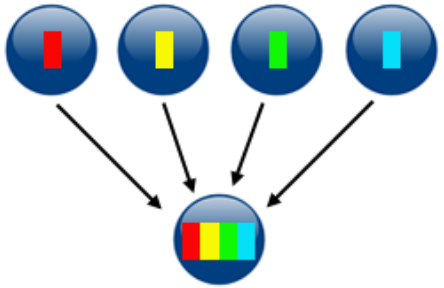
- MPI_Barrier
- MPI_Bcast, MPI_Scatter, MPI_Gather, MPI_Reduce



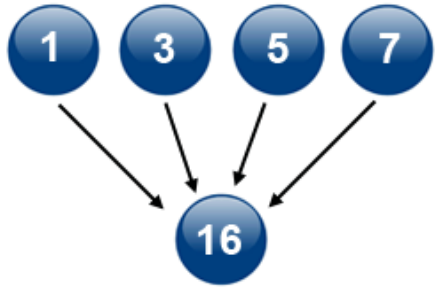
broadcast



scatter



gather



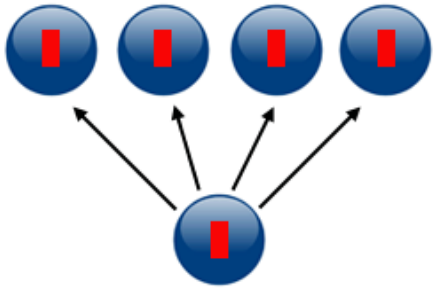
reduction

- MPI_Allgather, MPI_Allreduce

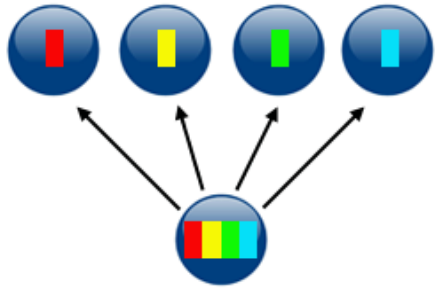
PROGRAMAÇÃO EM MEMÓRIA DISTRIBUÍDA

Tarefa 16:
Implemente um programa MPI que calcule o produto y=A•x, onde A é uma matriz MxN e x é um vetor de tamanho N. Divida a matriz A por linhas entre os processos com `MPI_Scatter`, e distribua o vetor x inteiro com `MPI_Bcast`. Cada processo deve calcular os elementos de y correspondentes às suas linhas e enviá-los de volta ao processo 0 com `MPI_Gather`. Compare os tempos com diferentes tamanhos de matriz e número de processos.

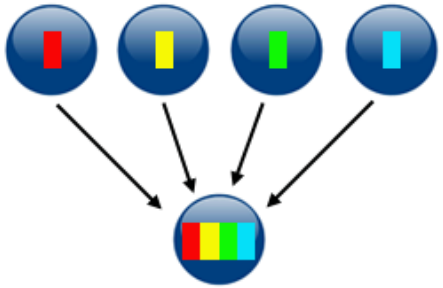
- MPI_Barrier
- MPI_Bcast, MPI_Scatter, MPI_Gather, MPI_Reduce



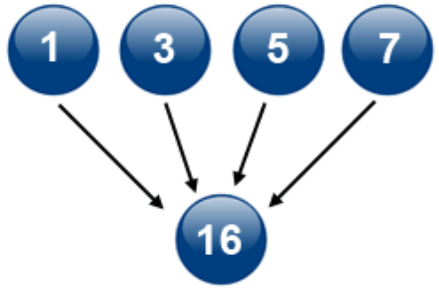
broadcast



scatter



gather



reduction

- MPI_Allgather, MPI_Allreduce

PROGRAMAÇÃO EM MEMÓRIA DISTRIBUÍDA

- MPI permite representar estruturas de dados não contíguas usando tipos derivados.
- Ajudam a evitar cópias manuais ou laços para enviar cada elemento.
- `MPI_Type_vector` representa padrões de blocos regulares, como colunas em matrizes armazenadas por linhas.
- `MPI_Type_create_resized` é usado para ajustar o espaçamento (`extent`) entre blocos derivados.
- Isso permite usar funções coletivas como `MPI_Scatter` para enviar colunas, como se fossem blocos contíguos.

PROGRAMAÇÃO EM MEMÓRIA DISTRIBUÍDA

Tarefa 17:
Reimplemente a tarefa 16, agora distribuindo as colunas entre os processos. Utilize `MPI_Type_vector` e `MPI_Type_create_resized` para definir um tipo derivado que represente colunas da matriz. Use `MPI_Scatter` com esse tipo para distribuir blocos de colunas, e `MPI_Scatter` para enviar os segmentos correspondentes de x. Cada processo deve calcular uma contribuição parcial para todos os elementos de y e usa `MPI_Reduce` com `MPI_SUM` para somar os vetores parciais no processo 0. Discuta as diferenças de acesso à memória e desempenho em relação à distribuição por linhas.

- MPI permite representar estruturas de dados não contíguas usando tipos derivados.
- Ajudam a evitar cópias manuais ou laços para enviar cada elemento.
- `MPI_Type_vector` representa padrões de blocos regulares, como colunas em matrizes armazenadas por linhas.
- `MPI_Type_create_resized` é usado para ajustar o espaçamento (`extent`) entre blocos derivados.
- Isso permite usar funções coletivas como `MPI_Scatter` para enviar colunas, como se fossem blocos contíguos.

PROGRAMAÇÃO DE DISPOSITIVOS MASSIVAMENTE PARALELOS

- CUDA:
  - Principal linguagem de programação para GPUs
  - Proprietária da Nvidia
- SYCL
  - Padrão aberto para programação heterogênea com C++
- OpenAcc
  - Padrão aberto, parte do arsenal da Nvidia
- OpenMP
  - `#pragma omp target`
    - Cláusulas: `map`, `device`, `private`, `firstprivate`, `nowait`
  - `#pragma omp loop`
    - Cláusulas: `reduction`, `collapse`, etc.
- Tutorial sobre programação de GPUs com OpenMP da Universidade de Bristol
  - https://github.com/UoB-HPC/openmp-tutorial
  - https://github.com/NPAD-UFRN/openmp-tutorial (adaptação para o NPAD)

PROGRAMAÇÃO DE DISPOSITIVOS MASSIVAMENTE PARALELOS

Tarefa 18:
Faça os exercícios de adição de vetores, `vadd.c`, dos slides 27 e 48 do tutorial de programação de GPUs com OpenMP em um dos nós com GPU do NPAD.
Compare os tempos de execução somente na CPU e com o uso da GPU.
Reporte seu progresso apresentando os problemas encontrados e as soluções apresentadas.

- CUDA:
  - Principal linguagem de programação para GPUs
  - Proprietária da Nvidia
- SYCL
  - Padrão aberto para programação heterogênea com C++
- OpenAcc
  - Padrão aberto, parte do arsenal da Nvidia
- OpenMP
  - `#pragma omp target`
    - Cláusulas: `map`, `device`, `private`, `firstprivate`, `nowait`
  - `#pragma omp loop`
    - Cláusulas: `reduction`, `collapse`, etc.
- Tutorial sobre programação de GPUs com OpenMP da Universidade de Bristol
  - https://github.com/UoB-HPC/openmp-tutorial
  - https://github.com/NPAD-UFRN/openmp-tutorial (adaptação para o NPAD)

PROGRAMAÇÃO DE DISPOSITIVOS MASSIVAMENTE PARALELOS

Tarefa 19:
Faça o exercício de transferência de calor (`heat.c`, slide 64) do tutorial de programação de GPUs com OpenMP.
Explore as diretivas de paralelização e movimentação de dados para copiar dados entre host e dispositivo.
Experimente com diferentes tamanhos de problema perfilando o programa com o nsys.

- CUDA:
  - Principal linguagem de programação para GPUs
  - Proprietária da Nvidia
- SYCL
  - Padrão aberto para programação heterogênea com C++
- OpenAcc
  - Padrão aberto, parte do arsenal da Nvidia
- OpenMP
  - `#pragma omp target`
    - Cláusulas: `map`, `device`, `private`, `firstprivate`, `nowait`
  - `#pragma omp loop`
    - Cláusulas: `reduction`, `collapse`, etc.
- Tutorial sobre programação de GPUs com OpenMP da Universidade de Bristol
  - https://github.com/UoB-HPC/openmp-tutorial
  - https://github.com/NPAD-UFRN/openmp-tutorial (adaptação para o NPAD)

PROGRAMAÇÃO DE DISPOSITIVOS MASSIVAMENTE PARALELOS

- OpenMP
  - `#pragma omp target`
    - Cláusulas: `map`, `device`, `private`, `firstprivate`, `nowait`
  - `#pragma omp loop`
    - Cláusulas: `reduction`, `collapse`, etc.
  - `#pragma omp target data`
  - `#pragma omp target enter data`
  - `#pragma omp target exit data`
  - `#pragma omp target update`
- Tutorial sobre programação de GPUs com OpenMP da Universidade de Bristol
  - https://github.com/UoB-HPC/openmp-tutorial
  - https://github.com/NPAD-UFRN/openmp-tutorial (adaptação para o NPAD)

PROGRAMAÇÃO DE DISPOSITIVOS MASSIVAMENTE PARALELOS

Tarefa 20:
Faça o exercício de transferência de calor (`heat.c`, slide 102) do tutorial de programação de GPUs com OpenMP.
Explore as diretivas de movimentação de dados para evitar que esse aspecto domine a execução, evitando a GPU fique muito tempo ociosa e otimizando o desempenho.

- OpenMP
  - `#pragma omp target`
    - Cláusulas: `map`, `device`, `private`, `firstprivate`, `nowait`
  - `#pragma omp loop`
    - Cláusulas: `reduction`, `collapse`, etc.
  - `#pragma omp target data`
  - `#pragma omp target enter data`
  - `#pragma omp target exit data`
  - `#pragma omp target update`
- Tutorial sobre programação de GPUs com OpenMP da Universidade de Bristol
  - https://github.com/UoB-HPC/openmp-tutorial
  - https://github.com/NPAD-UFRN/openmp-tutorial (adaptação para o NPAD)