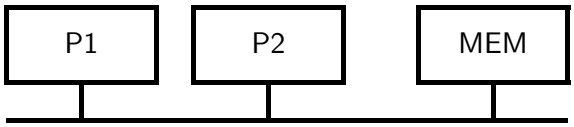
Processamento Paralelo

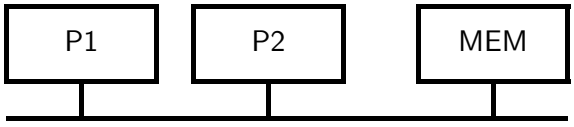
- Dois exemplos
- Ganho, *speed-up* ou aceleração
- Por que é difícil?
  - granularidade
  - escalabilidade
  - balanceamento de carga
  - sincronização
  - computação×comunicação

Modelo de Máquina



- Todas as variáveis compartilhadas através do barramento;
- tempo de acesso à memória igual para P1 e P2;
- variáveis escalares acessadas via *global pointer* (gp)
  "variáveis pequenas" armazenadas na *global area*;
- código não é compartilhado.

Exemplo: redução de um vetor

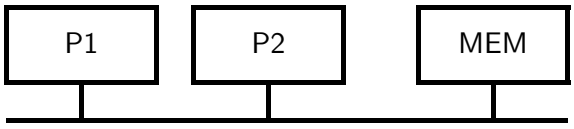


// P1 computa $i \in [0, N/2)$, P2 computa $i \in [N/2, N)$

```
sum = 0;
for (i=0; i < N; i+=1)
   sum = sum + A[i];
```

Problemas?

Exemplo: redução de um vetor (cont)



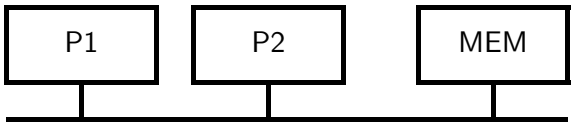

```
// P1 computa i ∈ [0, N/2) , P2 computa i ∈ [N/2, N)
sum = 0;
for (i=0; i < N; i+=1)
   sum = sum + A[i];     // lw r5, adr_SUM(gp)
                         // add r5, r5, rA     # rA=A[i]
                         // sw r5, adr_SUM(gp)
```

Problemas?

— quais variáveis *podem* ser privatizadas?

— quais variáveis devem ser compartilhadas?

Exemplo: redução de um vetor (cont)




```
// P1 computa i ∈ [0, N/2) , P2 computa i ∈ [N/2, N)
sum = 0;                  // sum é variável global
for (i=0; i < N; i+=1)    // i é variável local ⇝ reg?
   sum = sum + A[i];      // A[] é variável global
y = f(sum);               // uso da variável global
```

Problemas?

— como sabe se P1 e P2 terminaram? sincronização

— quem define a parcela de trabalho de P1 e P2? balanceamento

— o que acontece se adicionamos P3 e P4? escalabilidade

Linguagens para programação paralela

Linguagens usadas em programação paralela (ou concorrente) possuem construções que escondem detalhes do programador

Tipicamente, "compilador paralelo" pré-processa código fonte, detecta paralelismo simples e adiciona código ao fonte; "compilador sequencial" compila código e produz executável

Compilador paralelo adiciona *diretivas* que sinalizam a necessidade de tratamento especial pelo compilador sequencial

Exemplos:

`_PARALLEL_` comando deve ser expandido para execução paralela

`_SPAWN_` função deve ser replicada, uma cópia por processo

`_THREAD_` função que será executada por um processo

Exemplo: redução de um vetor em mais detalhes (i)

```
main() {                          // P procs
   int V[MAX];
   res = _PARALLEL_( reduz(), V, MAX, P );  // diretiva
   y = f(res);                    // uso da variável global
}

int reduz(int *V, int n, int m) {
   for (r=0,i=n; i < m; i+=1)
      r += V[i];
   return r;
}
```


Exemplo: redução de um vetor em mais detalhes (ii)

```
main() {                          // P procs
   int V[MAX], R[P];
   // expansão da diretiva _PARALLEL_
   s = MAX / P;                   // atribui trabalho
   for (i=0; i<P; i++)            // cria processos paralelos
      R[i] = _SPAWN_( reduz(V, i*s, (i+1)*s) );
   for (sum=0,i=0; i<P; i++)      // coleta resultados
      sum += R[i];
   y = f(sum);                    // uso da variável global
}
int _THREAD_ reduz(int *V, int n, int m) {
   for (r=0,i=n; i < m; i+=1) { r += V[i]; }
   return r;
}
```

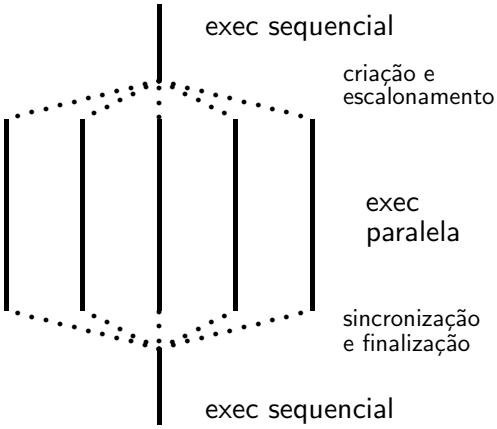

Exemplo: redução de um vetor em mais detalhes (iii)

```
main() {
   int V[MAX], R[P];
   s = MAX / P;                     // atribui trabalho
   for (i=0; i<P; i++)              // cria processos paralelos
      R[i] = _SPAWN_( reduz(V, i*s, (i+1)*s) );
   for (sum=0,i=0; i<P; i++)        // coleta resultados parciais
      sum += R[i];
   y = f(sum); }
```

Custos da execução paralela:

— computar distribuição de carga e atribuir trabalho às *threads*
— criar *threads* e disparar execução paralela scatter
— coletar os resultados parciais das *threads* gather
estas tarefas são executadas sequencialmente...

Exemplo: redução de um vetor em mais detalhes (iv)



tarefas do sistema de suporte à execução paralela:

criar *threads*
escalonar trabalho

execução em paralelo

sincronizar *threads*
coletar resultados

Processamento Paralelo — conceitos

multiprocessador sistema com mais de um processador

process level parallelism programas executam independentemente num multiprocessador (servidores Unix)

processo $\triangleq$ programa em execução: estado + espaço de endereçamento

thread level parallelism mais de uma linha de execução dentro de um processo (TLP)

thread $\triangleq$ processo leve, com registradores, PC, pilha privativos, *threads* compartilham espaço de endereçamento do processo

data level parallelism mesma operação aplicada sobre um conjunto de dados (unidades vetoriais, DLP)

instruction level paralleilsm paralelismo entre instruções (ILP)

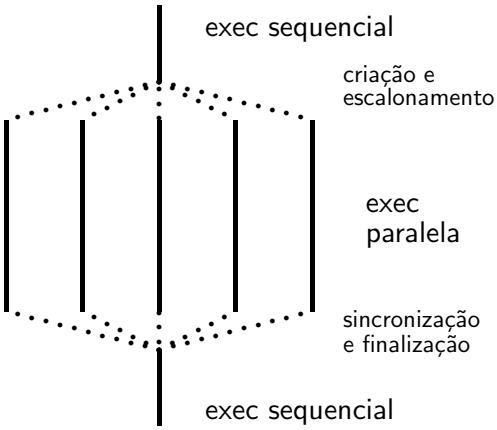
granularidade fina: ILP, DLP; grossa: processo, thread

Escrever programas paralelos é DIFÍCIL!!

1. Escalonamento: que processador executa qual tarefa e quando
   alocar trabalho, iniciar e terminar execução paralela são operações custosas que serializam execução
2. balanceamento de carga: todos processadores devem executar o mesmo tanto de trabalho
   proc. com mais carga atrasa a todos
3. sincronização: mal necessário com *threads*, para garantir corretude sob modelo sequencial de execução
   serializa execução
4. comunicação: cooperação necessita comunicação
   trabalho adicional para "empacotar", enviar, receber, "desempacotar" mensagens
   enquanto comunica, não computa

Execução paralela num multiprocessador deve prover melhor desempenho e eficiência que execução serial num uniprocessador.

Execução paralela



Lei de Amdahl,
versão paralela

$$G = \frac{T_{serial}}{T_{paralelo}}$$

$$= \frac{1}{(1 - F_p) + (F_p \ / \ G_p)}$$

F_p é a *fração paralela*

O que é a fração serial?

- Inicialização
  - ▷ de estruturas de dados
  - ▷ criação de processos nos processadores
  - ▷ distribuição de trabalho entre os processadores
- Sincronização
  - ▷ contenção no acesso às regiões críticas — semáforos
  - ▷ mudanças de fases do programa — laço mais externo
- Término
  - ▷ coleta dos resultados parciais
  - ▷ E/S em disco

Lei de Amdahl (i)

Tempo de execução após melhoria =
(tempo de execução afetado / quanto melhorou)
+ tempo de execução não-afetado

este termo é muito importante

Exemplo:
programa executa em 100s, multiplicações consomem 80%
do tempo total. Quanto devo melhorar o circuito multiplicador
se quero tempo total em 20s ?

$20 = 80/n + 20$ erm...

Idem, se quero tempo total em 40s ?

$40 = 80/n + 20$

Lei de Amdahl (ii)

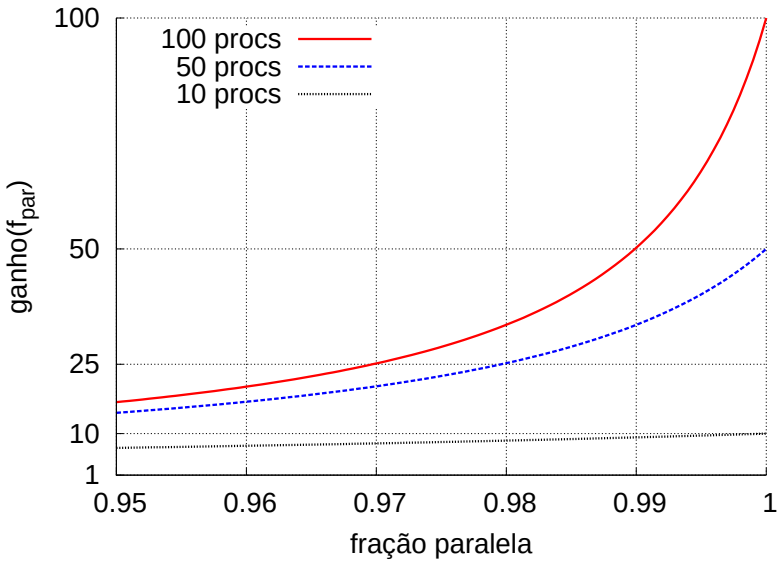
$$\begin{aligned} \text{Ganho}_{\text{total}} &= \frac{\text{Tempo}_{\text{orig}}}{\text{Tempo}_{\text{melhor}}} \\ &= \frac{1}{(1 - \text{Frac}_{\text{melhor}}) + (\text{Frac}_{\text{melhor}} \ / \ \text{Ganho}_{\text{melhor}})} \end{aligned}$$

Desejo ganho de 90x ao executar com 100 processadores. Qual a fração da computação que pode ser sequencial?

$$\begin{aligned} 90 &= \frac{1}{(1-f) + f/100} \\ f &= 0,999 \\ (1-f) &= 0,1\% \end{aligned}$$

fração serial = 0,1% para ganho 90/100

Lei de Amdahl (iii)



Escalabilidade (i)

Programa soma 10 escalares (serial) e então duas matrizes 10x10 com P_{10} e com P_{100}. Desempenho proporcional ao tempo t da adição.

$$\begin{aligned} T(P_1) &= 10t + 100t = 110t \\ T(P_{10}) &= 10t + 100t/10 = 20t \\ T(P_{100}) &= 10t + 100t/100 = 11t \\ \\ G(P_1/P_{10}) &= 110t/20t = 5,5 \qquad \text{5,5 com P=10} \\ G(P_1/P_{100}) &= 110t/11t = 10 \qquad \text{10 com P=100} \\ \\ E(P_{10}) &= G(P_{10})/10 \quad = 5,5/10 = 55\% \\ E(P_{100}) &= G(P_{100})/100 = 10/100 = 10\% \end{aligned}$$

T tempo, G ganho, E eficiência $= G(P_n)/n$

Escalabilidade (ii)

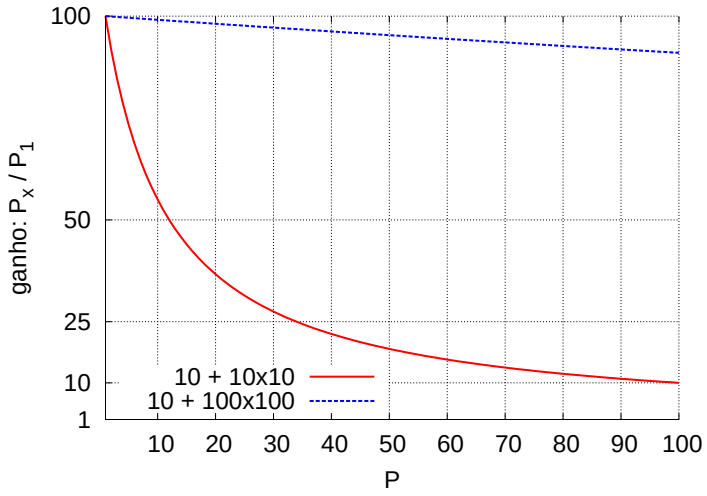
Programa soma 10 escalares (serial) e então duas matrizes 100x100 com P_{10} e com P_{100}. Desempenho prop. ao tempo t da adição.

$$
\begin{aligned}
T'(P_1) &= 10t + 10.000t = 10.010t \\
T'(P_{10}) &= 10t + 10.000t/10 = 1.010t \\
T'(P_{100}) &= 10t + 10.000t/100 = 110t \\
G'(P_1/P_{10}) &= 10.010t/1.010t = 9,9 \qquad \text{9,9 com P=10} \\
G'(P_1/P_{100}) &= 10.010t/110t = 91 \qquad \text{91 com P=100} \\
E'(P_{10}) &= G(P_{10})/10 \quad = 9,9/10 = 99\% \\
E'(P_{100}) &= G(P_{100})/100 = 91/100 = 91\%
\end{aligned}
$$

T tempo, G ganho, E eficiência $= G(P_n)/n$

Escalabilidade (iii)

Programa soma 10 escalares (serial) e duas matrizes NxN com $P_{1..100}$



Escalabilidade (iv)

- **Strong Scaling:** mede-se ganho sem aumentar conjunto de dados
  mesmo problema em menor tempo
  Para conjunto de trabalho | M |, mem/proc = M/P

- **Weak Scaling:** mede-se ganho aumentando problema
  proporcionalmente a P
  problema maior no mesmo tempo
  Para conjunto de trabalho | M |, mem/proc = M

Balanceamento de Carga (i)

Programa soma 10 escalares (serial) e então duas matrizes 100x100 com P_{10} e com P_{100}. Desempenho prop. ao tempo t da adição.
P^0 executa 2% do trabalho, $P^{1..99}$ dividem resto. Como afeta ganho?

$$\begin{aligned} T''(P_{100}) &= 10t + \max(9800t/99, 200t/1) = 210t \\ G''(P_1/P_{100}) &= 10.010t/210t = 48 \ll 91 \quad \approx 1/2 \end{aligned}$$

P^0 executa 5% do trabalho, $P^{1..99}$ dividem resto. Como afeta ganho?

$$\begin{aligned} T'''(P_{100}) &= 10t + \max(9500t/99, 500t/1) = 510t \\ G'''(P_1/P_{100}) &= 10.010t/510t = 20 \ll 91 \quad \approx 1/5 \end{aligned}$$

Balanceamento de Carga (ii)

Mesmo problema, P_0 executa mais trabalho, $P_{1..99}$ dividem resto