

# Introdução

## ■ Desempenho:

- Há vantagem na utilização de mais que um processador?
- Quantos processadores seriam necessários?
- Como medir a vantagem na utilização da computação paralela?
- Pode-se fazer uso de duas medidas:
  - *Speedup*: Qual o ganho na utilização da computação paralela?
  - Eficiência: Quanto da potência computacional envolvida foi utilizada?

# Introdução

## ■ Desempenho:

- *Speedup* ( $S_p$ ): Relação entre o tempo para executar um algoritmo em um único processador ( $T_1$ ) e o tempo para executá-lo em  $p$  processadores ( $T_p$ );

$$S_p = T_1 / T_p$$

## – Exemplo:

- Tempo para execução da multiplicação de matrizes em um processador:  $T_1 = 10 \text{ ut}$
- Tempo para execução da multiplicação de matrizes em 3 processadores:  $T_p = 4 \text{ ut} \quad (p=3)$
- *Speedup* alcançado  $\rightarrow S_p = 10/4 \rightarrow \mathbf{S_p = 2,5}$

# Introdução

## ■ Desempenho:

- Eficiência ( $E_f$ ): Relaciona *Speedup* e o número de processadores;

$$E_f = S_p / p$$

- Tomando como base o exemplo anterior ( $S_p = 2,5$  e  $p = 3$ ):
  - Eficiência  $\rightarrow E_f = 2,5 / 3 \rightarrow E_f = 0,83$

# Speedup X Eficiência

- No caso ideal:
  - $Speedup = p$  e Eficiência = 1
- No caso real:
  - $Speedup < p$  e Eficiência < 1

## Lei de Grosch e Efeito Amdahl

- Em casos excepcionais:
  - $Speedup > p$  e Eficiência > 1

**Como?**

# Speedup X Eficiência

Em casos excepcionais:

- $\text{Speedup} > p$  e  $\text{Eficiência} > 1$

**Como?**

- Quantidade de Memória
- Quantidade de Cache

Lembrar exemplo da matriz

Toda a matriz não cabe na memória (ou no cache) de um processador  
Dividindo a matriz as partes cabem na memória (ou no cache) de cada processador -> evita acessos a disco (ou a memória)

# Speedup X Eficiência

- No caso ideal:
  - $Speedup = p$  e Eficiência = 1
- No caso real:
  - $Speedup < p$  e Eficiência < 1

## Lei de Grosch e Efeito Amdahl

- Em casos excepcionais:
  - $Speedup > p$  e Eficiência > 1

**Como?**

# Desempenho

- Lei de Grosch: A utilização de dois processadores com capacidade " $c$ " levará a um desempenho pior que a utilização de um processador de capacidade " $2*c$ ";
- Efeito de Amdahl: As sobrecargas (sincronismo, comunicação e ativação de processos) tendem a diminuir com o aumento da complexidade das tarefas (diminuição da parte seqüencial);

# Lei de Grosch

## ■ Desempenho:

- Throughput: Número de Resultados Produzidos por Unidade de Tempo. Formas de Aumentar:
  - Aumentar a velocidade do Processador
  - Aumentar o número de Processadores

**Lei de Grosch X Custo**



# Efeito de Amdahl

N processadores trabalhando em paralelo

Ideal  $\rightarrow T(N) = 1/N * T(1)$

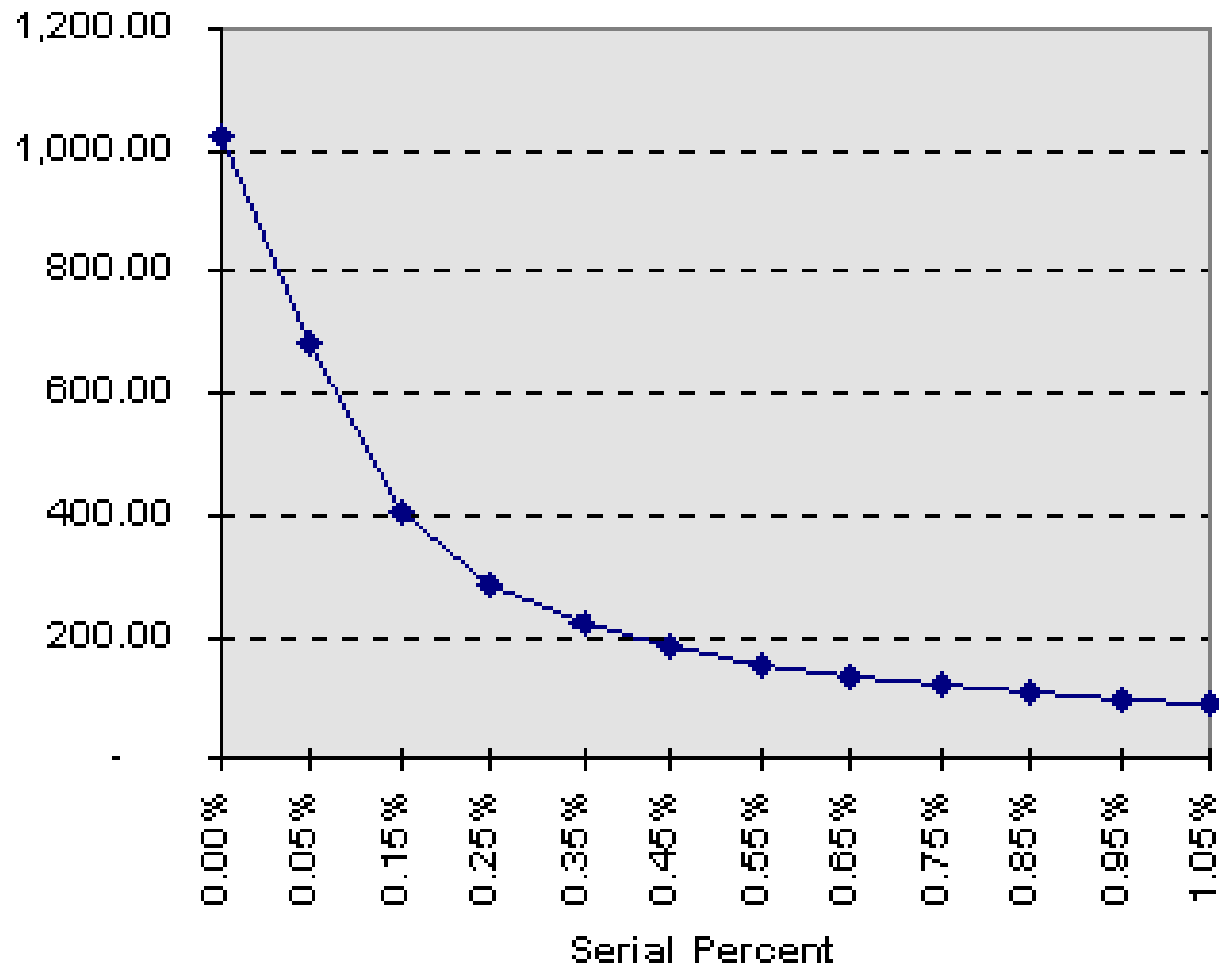
Real:

- Temos uma parte B do programa que é estritamente seqüencial
- Tempo para a parte seqüencial  $\rightarrow B * T(1)$
- Tempo para a parte paralela  $\rightarrow (1-B) * T(1)/N$
- Tempo total em paralelo  $\rightarrow T(N) = B * T(1) + (1-B) * T(1)/N$
- $Sp = T(1) / T(N)$

$$Sp = \frac{N}{(B*N) + (1-B)}$$

# Efeito de Amdahl

Speedup by Amdahl's Law ( $P=1024$ )



# Efeito de Amdahl X Lei de Groschi

---

Se B é pequeno – programação paralela é eficiente

Se B é grande – Melhor aumentar capacidade do processador

# Problemas com a métrica Speedup

## Comparação de resultados

- Escolha entre dois algoritmos para resolver um problema
- 10 Processadores

Algoritmo 1  $\rightarrow$   $Sp = 10$

Algoritmo 2  $\rightarrow$   $Sp = 6$

# Problemas com a métrica Speedup

## Comparação de resultados

- Escolha entre dois algoritmos para resolver um problema
- 10 Processadores

Algoritmo 1  $\rightarrow$   $Sp = 10$

Algoritmo 2  $\rightarrow$   $Sp = 9$

Otimo!

Speedup ideal!

# Problemas com a métrica Speedup

- 10 Processadores

Algoritmo 1

$$T(1) = 100 \quad T(10) = 10 \quad Sp = 10$$

Algoritmo 2

$$T(1) = 81 \quad T(10) = 9 \quad Sp = 9$$

# Problemas com a métrica Speedup

- 10 Processadores

Algoritmo 1

$$T(1) = 100 \quad T(10) = 10 \quad Sp = 10$$

Algoritmo 2

$$T(1) = 81 \quad T(10) = 9 \quad Sp = 9$$

# **Etapas para o Desenvolvimento e Análise de um Programa Paralelo**

- **I - Desenvolvimento de um Algoritmo Paralelo**
  - Abordagem do Algoritmo
  - Identificação do Algoritmo e Divisão dos Processos
  - Organização do Trabalho
  
- **II - Desenvolvimento do Programa Paralelo**
  - Formas de Expressar Paralelismo
  - Comunicação e Sincronismo
  - Linguagens para Programação Paralela



# Etapas para o Desenvolvimento e Análise de um Programa Paralelo

## ■ III - Mapeamento de Processos:

- Escalonamento
- Balanceamento de Carga
- Migração de Processos

## ■ IV - Teste e Depuração

- Efeito na Inserção de Testes
- Dificuldades com E/S Paralelo

## ■ V - Avaliação de Desempenho

- *Speedup*
- Eficiência

# **Etapas para o Desenvolvimento e Análise de um Programa Paralelo**

- I - Desenvolvimento de um Algoritmo Paralelo
- II - Desenvolvimento do Programa Paralelo
- III - Mapeamento de Processos:
- IV - Teste e Depuração
- V - Avaliação de Desempenho



# **Desenvolvimento do Algoritmo Paralelo**

Abordagem do Algoritmo  
Identificação do Algoritmo e Divisão dos Processos  
Organização do Trabalho

# Desenvolvimento do Algoritmo Paralelo

## ■ Programa Seqüencial:

- Existência de programa seqüencial com documentação e especificação pobres;
- Utilização de ferramentas;
- Baixa flexibilidade - *Speedup* limitado.

## ■ Algoritmo seqüencial

- Adaptar algoritmo seqüencial e refazer programa;
- Nem sempre melhor algoritmo seqüencial é o melhor algoritmo paralelo;
- Maior flexibilidade, mas ainda limitada;
- Melhores *speedups*, mas ainda pode não ser ideal.

# Desenvolvimento do Algoritmo Paralelo

- Problema a ser resolvido:
  - Análise do problema e proposta de algoritmo;
  - Programador deve conhecer bem o problema;
  - Alta flexibilidade, pode-se obter bom *speedup*;
  - Base em algoritmos paralelos que resolvem outros problemas.

# Desenvolvimento do Algoritmo Paralelo

- Problema em Programa Seqüencial -> Detectar e Eliminar as dependências
- Tipos de Dependências:
  - Dependência entre instruções -> ordem da execução influencia no resultado do programa
    - Exemplo – comandos de desvio, de leitura, etc.
  - Dependência de dados -> utilização de resultados anteriores
    - $A=B+C$ ;  $B=B+1$
  - Dependência em loops

# Desenvolvimento do Algoritmo Paralelo

Dependência em Loops:

- Independência entre as instâncias

Para  $J = 1$  até 10

$C[J] = A[J] + B[J]$

$C[J] = C[J] * 2.0$

- Dependência entre as instâncias

Para  $J = 1$  até 10

$A[J] = A[J-1] * 2.0$

# **Desenvolvimento do Algoritmo Paralelo**

- Dependência em programas paralelos:
  - Dependência de Comunicação
  - Dependência entre tarefas



# Abordagem do Algoritmo

## ■ Exemplo

Analizando pelo programa:

```
for (k=1;k<n-1;k++)  
    for (j=1;j<n-k;j++)  
        for (i=1;i<n-1-k;i++)  
            if (A[i]>A[i+1])  
            {  
                aux = A[i+1]  
                A[i+1] = A[i]  
                A[i] = aux  
            }
```

# Abordagem do Algoritmo

## ■ Exemplo: Método de Ordenação "Bolha"

Analizando pelo programa:

```
for (k=1;k<n-1;k++)  
    for (j=1;j<n-k;j++)  
        for (i=1;i<n-1-k;i++)  
            if (A[i]>A[i+1])  
            {  
                aux = A[i+1]  
                A[i+1] = A[i]  
                A[i] = aux  
            }
```

⇒ **Difícil paralelizar!**

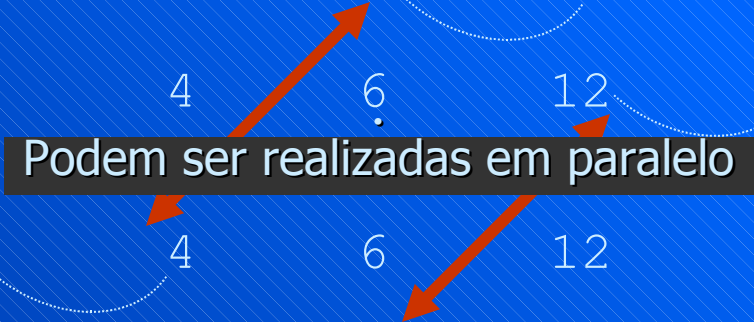
# Abordagem do Algoritmo

## ■ Exemplo: Método de Ordenação "Bolha"

### Analizando pelo algoritmo

passos

1.1	12	10	4	6	15	2	8
1.2	10	12	4	6	15	2	8
1.3	10	4	12	6	15	2	8
1.4	10	4	6	12	15	2	8
2.1	10	4	6	12	2	8	15
2.2	4	10	6	12	2	8	15
2.3	4	6	10	12	2	8	15
			⋮				



Podem ser realizadas em paralelo

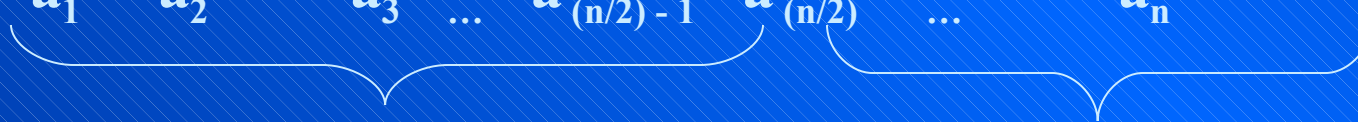
# Abordagem do Algoritmo

## ■ Pode-se concluir que:

- O passo 1.3 pode ser executado em paralelo com 2.1;
- O passo 1.4 pode ser executado em paralelo com 2.2;
- O passo 1.5 pode ser executado em paralelo com 2.3 e 3.1;

## ■ Pelo Problema:

–  $a_1 \quad a_2 \quad a_3 \quad \dots \quad a_{(n/2)-1} \quad a_{(n/2)} \quad \dots \quad a_n$



– Ao final,  $\overset{P1}{\text{deve-se realizar um merge.}} \overset{P2}{\text{merge.}}$

# **Desenvolvimento do Algoritmo Paralelo**

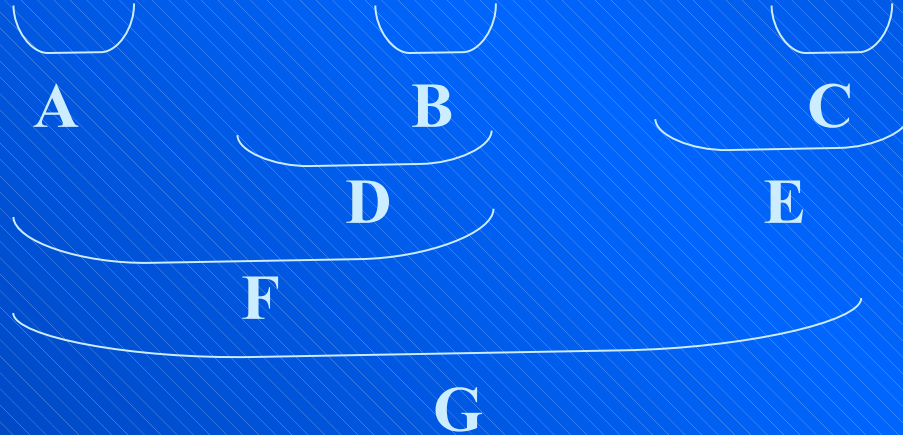
Abordagem do Algoritmo  
Identificação do Algoritmo e Divisão dos Processos  
Organização do Trabalho

# Identificação do paralelismo e divisão dos processos

- Aspectos importantes que devem ser considerados:
  - Arquitetura das máquinas disponíveis;
  - Tipo de comunicação;
  - Granulação;
  - Sincronismo.
- Exemplo: Expressão Aritmética.

# Identificação do paralelismo e divisão dos processos

■  $(a * b + c * d^2) * (g + f * h)$



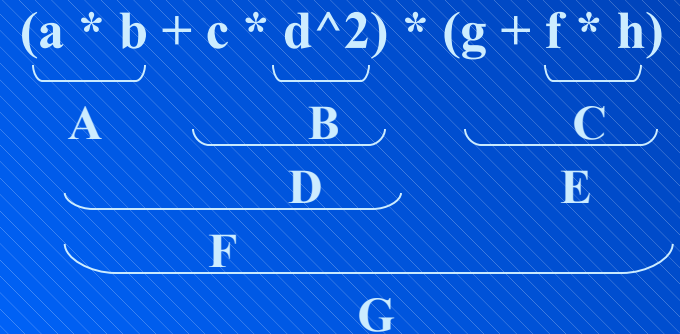
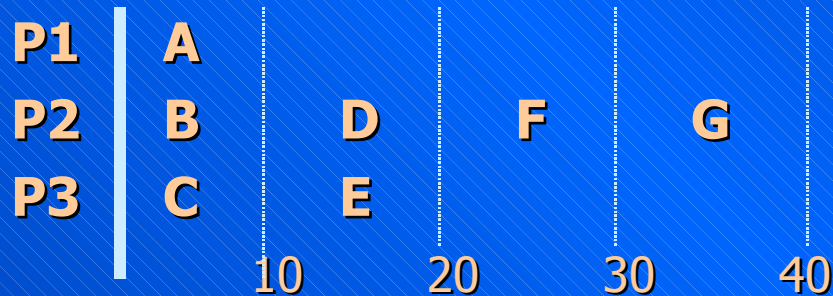
■ Considerando Toperação = 10 ut, tem-se que:

$$T_{seq} = 70 \text{ ut}$$

$$T_{par} = ?$$

# Identificação do paralelismo e divisão dos processos

- Utilizando três processadores:



- Ao final da execução, verifica-se que:

$T_{par} = 40 \text{ ut}$

(sem considerar comunicação!)

- Calculando o *speedup* e a eficiência...

$Sp = 70 / 40 = 1.75$

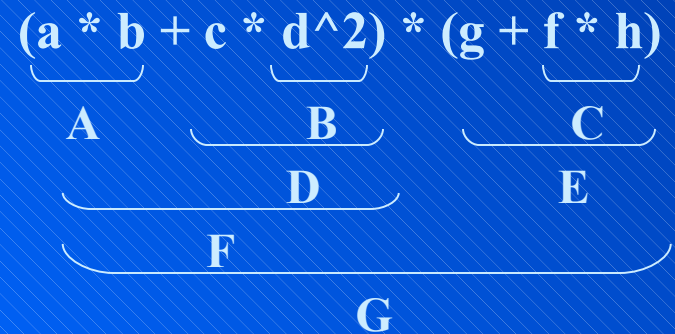
$Ef = 1.75 / 3 = 58\%$



# Identificação do paralelismo e divisão dos processos

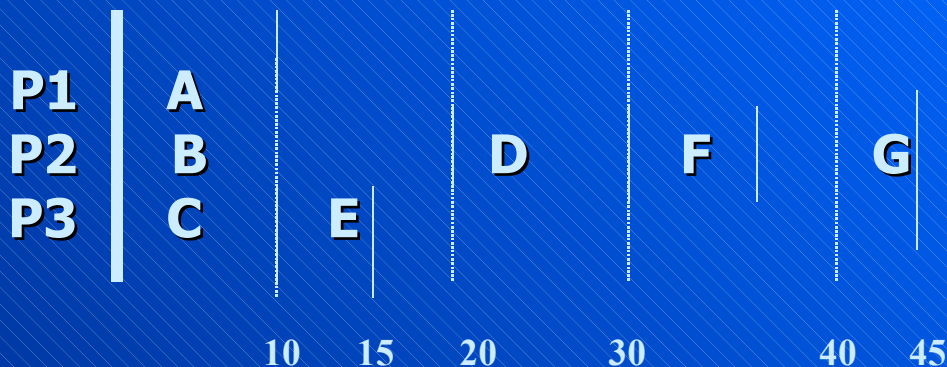
■ Agora, considerando:

- $T^{\wedge} = 20$  ut
- $T^* = 10$  ut
- $T^+ = 5$  ut



■  $T_{seq} = 70$  ut

■  $T_{par} = 45$  ut

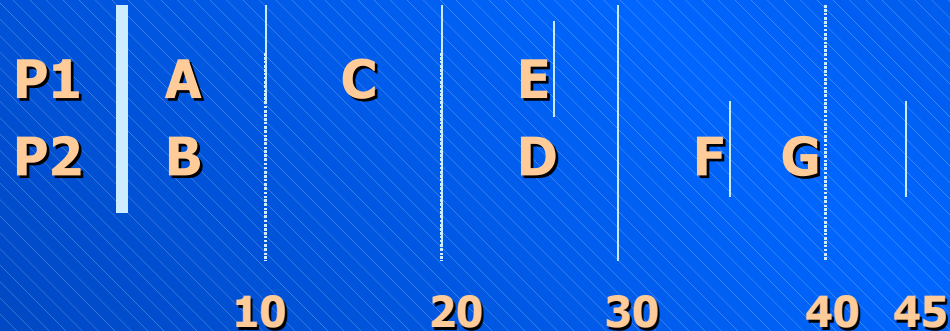


$$Sp = 70 / 45 = 1,55$$

$$Ef = 1,55 / 3 = 52\%$$

# Identificação do paralelismo e divisão dos processos

- Analisando o diagrama, nota-se que A pode ser executado por P3 ou C e E por P1



O tempo de execução  
ainda continua sendo 45 ut

- Assim:

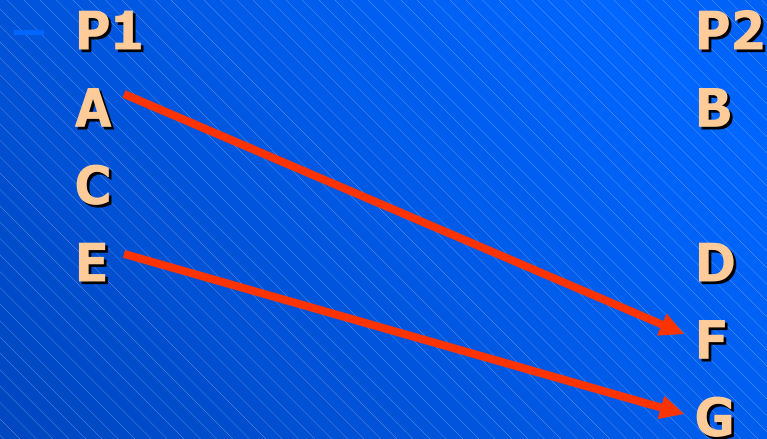
$$S_p = 70/45 = 1,55$$

$$E_f = 1,55 / 2 = 77\%$$



# Identificação do paralelismo e divisão dos processos

- Considerando comunicação...



$$\underbrace{(a * b + c * d^2)}_A * \underbrace{(g + f * h)}_E$$

Diagram illustrating the expression  $(a * b + c * d^2) * (g + f * h)$  with sub-expressions labeled A through G:

- A:  $a * b$
- B:  $c * d^2$
- C:  $g + f * h$
- D:  $a * b + c * d^2$
- E:  $g + f * h$
- F:  $(a * b + c * d^2) * (g + f * h)$
- G:  $(a * b + c * d^2) * (g + f * h)$

- Assim:  $T_{par} = 45 + 2 * T_{comunicação}$

**Comunicação = sobrecarga!**

# Abordagem do Algoritmo

## ■ Exemplo 2 - Soma dos Elementos de um Vetor

$$- a_1 + a_2 + a_3 + \dots a_{(n/2)-1} + a_{(n/2)} + \dots + a_{n-1} + a_n$$