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
- III - Mapeamento de Processos:
- IV - Teste e Depuração
- V - Avaliação de Desempenho

Desenvolvimento do Algoritmo Paralelo 

Desenvolvimento do Programa Paralelo 

Desenvolvimento de Programas Paralelos

- Programa Seqüencial: conjunto de comandos executados seqüencialmente (PROCESSO);
  - Atribuições, Decisão, *Loops*, Subprogramas, etc.
- Programa Paralelo: Necessidade das construções para programação seqüencial e ainda:
  - Ativação de processos em paralelo;
  - Sincronização e Comunicação.
- Necessidade de linguagens e ferramentas para o desenvolvimento de programas paralelos!

Formas de Expressar Paralelismo

- Mecanismos para ativação de processos paralelos:
  - FORK/JOIN: Ativa dois processos em paralelo;
  - COBEGIN/COEND: Ativa vários processos em paralelo;
  - DOALL: Ativação das iterações de um *loop* em paralelo.

- FORK/JOIN
  - Opção primitiva e desestruturada.
  - Sintaxe:
    - **FORK t** $\rightarrow$ Ativa concorrentemente um processo no endereço t;
    - **JOIN n, a, b** $\rightarrow$ **n=n-1**

      Se n == 0 (último processo)

      Então vai para o endereço a

      Senão vai para o endereço b

Formas de expressar paralelismo

- COBEGIN/COEND
  - Especifica um conjunto de comandos que devem ser executados em paralelo;

```
COBEGIN
  A
  B        (Executa os processos A, B e C concorrentemente)
  C
COEND
```

Formas de expressar Paralelismo

- COBEGIN/COEND
  - Mais Estruturado;
  - Mais alto nível - fácil reconhecer os processos que estão sendo executados em paralelo;
  - Pouco flexível.
- FORK/JOIN
  - Toda sincronização é responsabilidade do desenvolvedor;

    Código obtido é obscuro e desestruturado;

    No entanto, é mais flexível que COBEGIN/COEND.

Formas de expressar paralelismo

- DOALL
  - Cada instância do loop será executada como um processo que roda concorrentemetne com as outras instâncias
  - Muito utilizado para operações com matrizes e vetores
  - Fácil de paralelizar
  - Dependência entre as diversas instâncias do loop

Formas de expressar Paralelismo

- Exemplo – Previsão de Tempo

```
foreach longitude, latitude, altitude
    ustar[i,j,k] = n * pi[i,j] * u[i,j,k]
    vstar[i,j,k] = m[j] * pi[i,j] * v[i,j,k]
    sdot[i,j,k] = pi[i,j] * sigmadot[i,j]
end
foreach longitude, latitude, altitude
    D = 4 * ((ustar[i,j,k] + ustar[i-1,j,k]) * (q[i,j,k] + q[i-1,j,k]) +
            terms in {i,j,k}{+,-}{1,2}
    piq[i,j,k] = piq[i,j,k] + D * delat
    similar terms for piu, piv, piT, and pi
end
foreach longitude, latitude, altitude
    q[i,j,k] = piq[i,j,k]/pi[i,j,k]
    u[i,j,k] = piu[i,j,k]/pi[i,j,k]
    v[i,j,k] = piv[i,j,k]/pi[i,j,k]
    T[i,j,k] = piT[i,j,k]/pi[i,j,k]
```

Formas de expressar Paralelismo

Exemplo 1

For i = 0 **To** n

$a_i = 0$

$a_0 = 0$ // $a_1 = 0$ // $a_2 = 0$ // $a_3 = 0$ //...../// $a_{n-1} = 0$ // $a_n = 0$

Doall i =0 To n

$a_i = 0$

Formas de expressar Paralelismo

Exemplo 2

Total = 0

For i = 1 **To** n

Total = Total + a_i

I. Considerando 2 processos

Total = (a_1 + a_2 + ... + $a_{n/2}$) + ($a_{n/2+1}$ + ...+ a_n)

Formas de expressar Paralelismo

Exemplo 2

```
Cobegin
    Begin
        Total1 = 0
        For i = 1 To n/2
            Total1 = Total1 + a_i
    End
    Begin
        Total2 = 0
        For i = n/2+1 To n
            Total2 = Total2 + a_i
    End
Coend
Total = Total1 + Total2
```

Formas de expressar Paralelismo

II Considerando n/2 processadores

Total = $(a_1 + a_2) + (a_3 + a_4) + (a_5 + a_6) + (a_7 + a_8) + \ldots + (a_{n-1} + a_n)$

Total = $b_1 + b_2 + b_3 + b_4 + \ldots + b_{n/2}$ (1)

Total = $c_1 + c_2$ .....

Doall i = 1 **To** n-1 **Step** 2
$j = \lceil i/2 \rceil + 1$
$b_j = a_i + a_{i+1}$

Doall i = 1 **To** n/2-1 **Step** 2
$j = \lceil i/2 \rceil + 1$
$c_j = b_i + b_{i+1}$

Formas de expressar Paralelismo

II Considerando n/2 processadores

Se n = potência de 2

k = n
While K >= 2
Doall i = 1 **To** k-1 **Step** 2
j = $\lceil i/2 \rceil + 1$
$a_j = a_i + a_{i+1}$
k = k / 2

Formas de expressar Paralelismo

Exemplo 3

$a_0 = 0$

$a_1 = 0$

For i = 2 To n

$a_i = a_{i-2} + x$

Cobegin

$a_0 = 0$

$a_1 = 0$

Coend

$a_2 = a_0 + x$

For i = 3 To n Step 2

Cobegin

$a_i = a_{i-2} + x$

$a_{i+1} = a_{i-1} + x$

Formas de expressar Paralelismo

Exemplo 4

For i = 0 **To** n

$x_i = (b_i * c_i) + 4$

$y_i = 2 * x_i$

Opção 1

Doall i = 0 To n

$x_i = (b_i * c_i) + 4$

Doall i = 0 To n

$y_i = 2 * x_i$

Formas de expressar Paralelismo

Exemplo 4

For i = 0 **To** n

$x_i = (b_i * c_i) + 4$

$y_i = 2 * x_i$

Opção 2

$x_0 = (b_0 * c_0) + 4$

For i = 1 **To** n

Cobegin

$x_i = (b_i * c_i) + 4$

$y_{i-1} = 2 * x_{i-1}$

Coend

Comunicação e Sincronismo

- Sincronização: Imposição de uma ordem na execução de processos;
- Comunicação: Permite que a execução de um processo influencie na execução do outro;
- Exemplo: Execução da operação a = a+1 paralelamente em dois processadores.

Comunicação e Sincronismo

- Exemplo:

```
proc1                          proc2
lê a
soma 1
armazena a

                               lê a
                               soma 1
                               armazena a
```

se inicialmente a = 0; Então, após a execução a =2

Comunicação e Sincronismo

- Exemplo:

```
proc1                    proc2
lê a

                         lê a

soma 1
armazena a

                         soma 1
                         armazena a
```

No final tem-se a =1 $\rightarrow$ Faltou Sincronização

Comunicação e Sincronismo

Custo da Sincronização:

- SIMD/ Paralelismo de baixo nível - hardware
- MIMD - software

Comunicação e Sincronismo

Sincronização:

- Hardware:
  - Paralelismo Fino
  - Clock Global
  - Exemplos: Pipelines, processadores vetoriais
- Software:
  - Sistemas Multicomputadores
  - Memória Compartilhada
    - extensão natural das Máquinas de Von Neumann Seqüenciais
  - Memória Distribuída
    - Troca de mensagens
    - Utilização do SEND/RECEIVE
    - CSP - Hoare

Comunicação e Sincronismo

- Sincronização em Memória Distribuída → Troca de Mensagens
  - `SEND <lista de expressões> TO <destino>`
  - `RECEIVE <lista de variáveis> FROM <fonte>`

- Fatores que devem ser verificados:
  - Tamanho da mensagem: fixa ou variável;
  - Comunicação: direta ou por buffer/bidirecional ou unidirecional;
  - Sincronização: bloqueante ou não-bloqueante;
  - Nomeação: direta ou global.

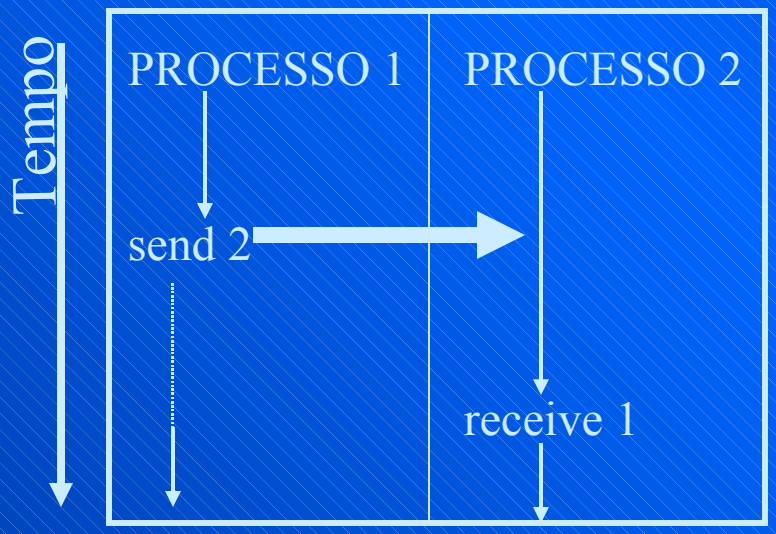
Comunicação e Sincronismo

- Sincronização em Memória Distribuída → Troca de Mensagens

- A troca de mensagens entre programas pode utilizar os mecanismos:
  - Ponto a Ponto;
  - *Rendezvous;*
  - RPC (*Remote Procedure Call*).

Comunicação e Sincronismo

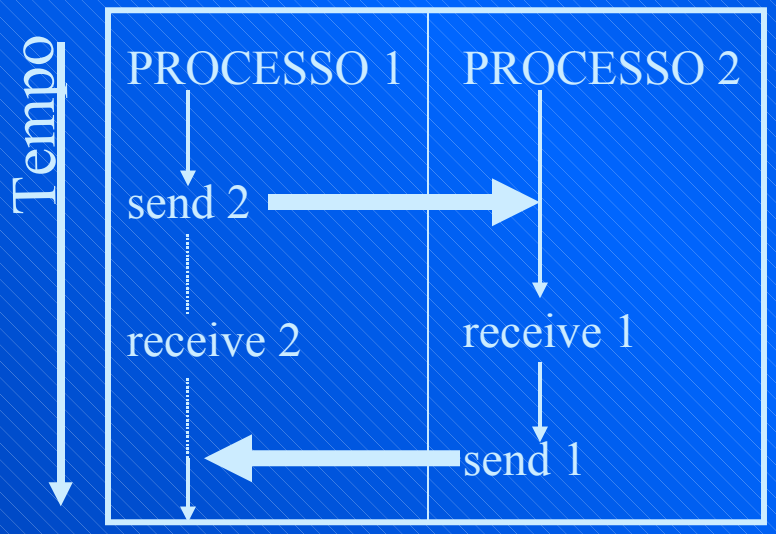
- Sincronização em Memória Distribuída → Troca de Mensagens
- Mecanismo Ponto a Ponto:
  - Dois processos paralelos executam as primitivas *SEND* e *RECEIVE*;
  - Comunicação síncrona e unidirecional;
  - Primitivas *Send/Receive* bloqueantes.



Comunicação e Sincronismo

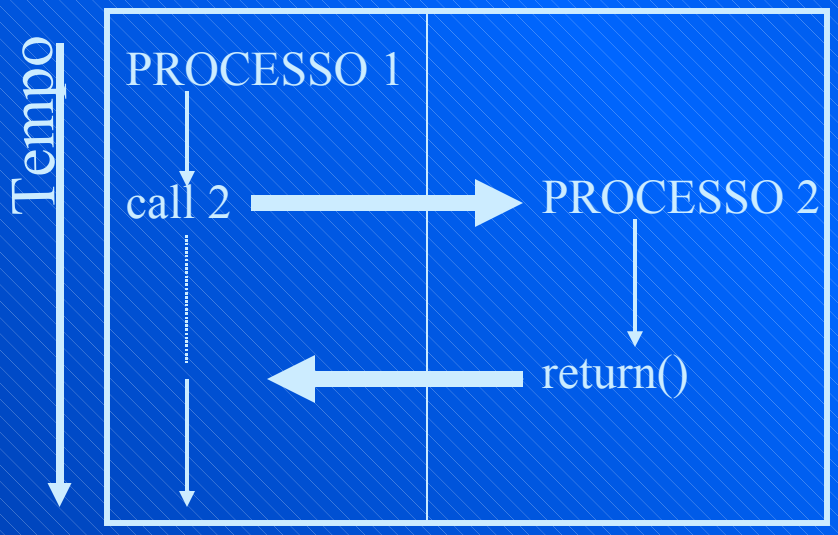
- Sincronização em Memória Distribuída → Troca de Mensagens

- Mecanismo *Rendezvous*
  - Comunicação síncrona e bidirecional;
  - Cada processo paralelo executa *SEND/RECEIVE*.



Comunicação e Sincronismo

- Sincronização em Memória Distribuída → Troca de Mensagens
- Mecanismo RPC
  - Comunicação síncrona e bidirecional;
  - Destino não executa um *RECEIVE*, mas é ativado pelo *SEND* requisitando um serviço;
  - Semelhante a chamadas de procedimentos locais.



Comunicação e Sincronismo

Exemplo a = a + 1 – Ponto a Ponto
Variável "a" na memória do proc1

```
proc1                          proc2
lê a
soma 1
armazena a
Send a, proc2

                               Receive a, proc1
                               soma 1
Receive a, proc2               Send a, proc1
```

Comunicação e Sincronismo

Proc3 – responsável pela variável "a"

```
proc1                    proc2                    proc3
Receive a,proc3                                   Loop
soma 1                                             Send a, any
Send a, proc3                                      Receive a,any
                         Receive a, proc3
                         soma 1
                         Send a, proc3
```

Comunicação e Sincronismo

- Sincronização em Memória Compartilhada → Cuidar do controle de acesso aos dados

- Problemas com:
  - Escrita em *buffer* cheio;
  - Leitura em *buffer* vazio;
  - Utilização de variáveis não calculadas.

Comunicação e Sincronismo

- Sincronização em Memória Compartilhada → Cuidar do controle de acesso aos dados

- Estratégias para controle de acesso à memória compartilhada:
  - **Semáforos**
  - **Monitores**
  - **Test-and-set**

Comunicação e Sincronismo

Semáforos: Variável não negativa (S) sobre a qual são definidas duas operações que devem ser executadas de forma indivisível;

```
Down(S) e Up(S)(ou P e V)
```

```
UP (S)    S = S + 1
DOWN (S) Se S = 0
              Então Bloqueia Processo
              Senão S = S -1
```

Comunicação e Sincronismo

- **Monitores:**
  - Mecanismo de mais alto nível
  - Encapsula em um único módulo a definição do recurso e das operações que o manipulam;

Comunicação e Sincronismo

- **Test-and-set**
  - busy wait
  - Necessita auxílio do kernel
  - Baixo nível
  - Pouco utilizada em programação paralela