Exploração de Paralelismo em Sistemas Vetoriais

Tópicos:

- Vetorização × Paralelização
- Reconhecimento de Variáveis de Indução
- Substituição Global
- Análise Semântica
- Eliminação de Anti-Dependências ou Dep.Saída
- Expansão de Escalares

Referência: Padua,D.A. & Wolfe,M.J. “Advanced Compiler Optimizations for Supercomputers”, *Communications of the ACM*, 29(12), Dec. 1986, pp.1184-1200.

Suporte à Paralelização

- Paralelismo total: *doall*

  Ex:
```
doall i=1,N
        ...  (iterações independentes)
enddoall
```
- Paralelismo parcial (com sincronização): *doacross*

  Ex:
```
doacross i=1,N
        ...
        signal(i)
        ...
        wait(i-1)
        ...
enddoacross
```

Vetorização x Paralelização

- **Problema**: Decidir pela vetorização ou pela paralelização, quando ambas são possíveis
- **Loops dentro de loops**: Pode ser possível paralelizar alguns e/ou vetorizar outros...

Critério: Eficiência na execução (pode depender da arquitetura específica!)

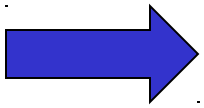
Exemplo:

```
          do j = 1,N
             do i=1,N
S1:            A(i,j+1) = B(i,j) + C(i,j)
S2:            D(i,j) = A(i,j)  * 2
             enddo
          enddo
```

Vetorização x Paralelização (cont.)

Exemplo (cont.):

- **1ª Opção**: Vetorizar loop `i` e paralelizar loop `j`
  - \+ Vetorização vai acessar arrays por colunas (*stride*=1)
  - – Devido à dependência, paralelização exige sincronização entre processadores
- **2ª Opção**: Paralelizar loop `i` e vetorizar loop `j`
  - – Vetorização vai acessar arrays por linhas (*stride*>1)
  - \+ Como não há dependência carregada pelo loop `i`, não é necessário haver sincronização entre processadores



Saldo final vai depender dos custos de acesso à memória e de sincronização entre processadores

Vetorização x Paralelização (cont.)

Exemplo (cont.):

- **1ª Opção**: Vetorizar loop `i` e paralelizar loop `j`

```
      doacross j=1,N
S1:         A(1:N,j+1) = B(1:N,j) + C(1:N,j)
            signal(j+1)
            if (j>1) wait(j)
S2:         D(1:N,j) = A(1:N,j) * 2
      enddoacross
```

  - S_1 pode ser executado simultaneamente por todos os processadores
  - O **início** da execução de S_2 ocorre em seqüência pelos processadores

Vetorização x Paralelização (cont.)

Exemplo (cont.):

- **2ª Opção**: Paralelizar loop `i` e vetorizar loop `j`

```
        doall i=1,N
S1:         A(i,2:N+1) = B(i,1:N) + C(i,1:N)
S2:         D(i,1:N) = A(i,1:N) * 2
        enddoall
```

  - Cada processador executa S_1 e depois S_2
  - A seqüência (S_1 ; S_2) pode ser executada simultaneamente por todos os processadores
  - Todos os arrays são acessados por linha (*stride*>1)

Exploração de Paralelismo em Sistemas Vetoriais

- **Objetivo Geral**: Explicitar o paralelismo potencial em casos de loops menos evidentes
- **Estratégia**: Transformar o programa original, de modo a facilitar a paralelização ou a vetorização
- **Justificativa**: Muitas vezes, há *falsas* dependências que impediriam a vetorização ou a paralelização. Após transformar o programa, tais dependências são removidas.

Reconhecimento de Variáveis de Indução

Exemplo:

```
inc = N
do i=1,N
  i2 = 2*i - 1
  X(inc) = Y(i) + Z(i2)
  inc = inc - 1
enddo
```

Def: Variáveis de Indução: variáveis cujos valores ao longo do loop formam uma Progressão Aritmética

Obs: `inc` e `i2` são variáveis de indução

Loop Vetorizado: `X(N:1:-1) = Y(1:N) + Z(1:2*N-1:2)`

Substituição Global

Exemplo:

```
     np1 = N+1
     np2 = N+2
     ...
     do i=1,N
S1:    B(i) = A(np1) + C(i)
S2:    A(i) = A(i) - 1
       do j=2,N
S3:      D(j,np1) = D(j-1,np2) * C(j) + 1
       enddo
     enddo
```

- np1>N : S_1 e S_2 (e iterações do loop `i`) são independentes
- np1 ≠ np2 : iterações do loop `j` são independentes

⟹ Loops podem ser vetorizados ou paralelizados

Análise Semântica

Exemplo:

```
     if (K > 0) then
       do i=1,N
S1:       A(i) = B(i) + A(i+K)
       enddo
     endif
```

- Analizando <u>apenas</u> S_1: $S_1 \ \delta_{<} \ S_1$

  (Loop não pode ser vetorizado nem paralelizado)
- Mas o loop só é executado se K>0; logo, $S_1 \ \boldsymbol{\delta}^{a}_{<} \ S_1$

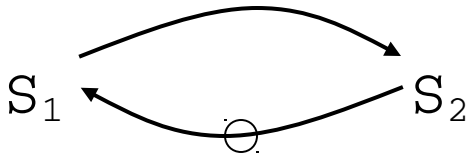
  $\therefore$ Loop **pode** ser vetorizado, mas **não** paralelizado

Eliminação de Anti-dependências ou de Dependências de Saída

Exemplo:

```
     do i=1,N
S1:    A(i) = B(i) + C(i)
S2:    A(i+1) = A(i) + 2 * D(i)
     enddo
```

Grafo de Dependências:

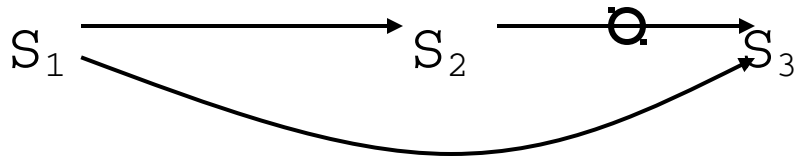


Idéia: Quebrar o ciclo através do uso de um array auxiliar

```
    do i=1,N
S1:    ATEMP(i) = B(i) + C(i)
S2:    A(i+1) = ATEMP(i) + 2 * D(i)
S3:    A(i) = ATEMP(i)
    enddo
```

Eliminação de Anti-dependências ou de Depend. de Saída (cont.)

Novo Grafo de Dependências:



→ Não há ciclos no grafo
∴ Pode haver vetorização

Código Vetorizado:

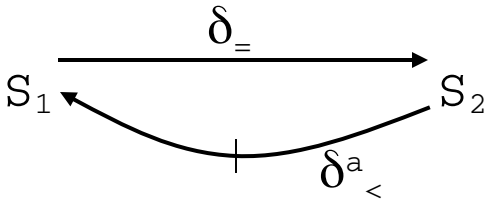
```
S1: ATEMP(1:N) = B(1:N) + C(1:N)
S2: A(2:N+1) = ATEMP(1:N) + 2 * D(1:N)
S3: A(1:N) = ATEMP(1:N)
```

Expansão de Escalares

Exemplo:

```
      do i=1,N
S1:     X = A(i) + B(i)
S2:     C(i) = X ** 2
      enddo
```

Grafo de Dependências:



Solução: Transformar `X` num array (elimina a anti-depend.)

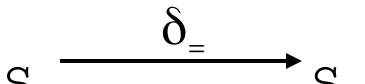
- Não haverá ciclos no grafo → Loop pode ser vetorizado
- Única dependência será $\delta_{=}$ → Loop pode ser paralelizado

Expansão de Escalares (cont.)

Novo Loop Transformado:

```
      do i=1,N
S1:     XTEMP(i) = A(i) + B(i)
S2:     C(i) = XTEMP(i) ** 2
      enddo
      X = XTEMP(N)
```

Grafo de Dependências:



Loop Paralelizado:

```
     doall i=1,N
S1:    XTEMP(i)=A(i)+B(i)
S2:    C(i)=XTEMP(i)**2
     enddoall
     X = XTEMP(N)
```

Loop Vetorizado:

```
XTEMP(1:N)=A(1:N)+B(1:N)
C(1:N)=XTEMP(1:N)**2

X = XTEMP(N)
```