Reestruturação de Programas em Sistemas Vetoriais

Tópicos:

- Técnicas de Otimização
- Notação para Dependência
- Vetorização de Loops
- Paralelização de Loops

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

Técnicas de Otimização

- Sistemas com um único processador:
  - Vetorização
- Sistemas com mais de um processador:
  - Vetorização e / ou Paralelização

Dependências de dados:

- Afetam tanto paralelização como vetorização, porém...
- Cada *tipo* de dependência tem efeitos distintos!

  (Ex: Anti-dependências impedem paralelização, mas não impedem necessariamente vetorização)

Notação para Dependência

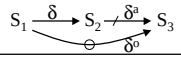
Dependência: Relação no conjunto de comandos

Exemplo:

S_1: A = B + C
S_2: D = A + 2
S_3: A = E + F

- Dependência de fluxo: $S_1\ \delta\ S_2$
- Antidependência: $S_2\ \delta^a\ S_3$
- Dependência de saída: $S_1\ \delta^o\ S_3$

Grafo de Dependências:



Notação para Dependência (Casos com Loops)

Exemplo-1: $S_1\ \delta\ S2 \rightarrow S_1\ \delta_{=}\ S_2$

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

Exemplo-2: $S_1^i\ \delta\ S_2^{i+1} \rightarrow S_1\ \delta_{<}\ S_2$

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

Exemplo-3: $S_2^i\ \delta^a\ S_1^{i+1} \rightarrow S_2\ \delta^a_{<}\ S_1$

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

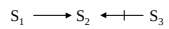

Vetorização de Loops

Regra: Se não há ciclos no Grafo de Dependências, o loop pode ser vetorizado

Exemplo:

```
      do i=1,N
S1:     A(i) = B(i)
S2:     C(i) = A(i) + B(i)
S3:     E(i) = C(i+1)
      enddo
```

Grafo de Dependências:



Novo código após Vetorização:

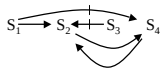
```
S1:     A(1:N) = B(1:N)
S3:     E(1:N) = C(2:N+1)
S2:     C(1:N) = A(1:N) + B(1:N)
```


Vetorização de Loops (cont.)

Outro Exemplo:

```
      do i=2,N
S1:     A(i) = B(i)
S2:     C(i) = A(i) + B(i-1)
S3:     E(i) = C(i+1)
S4:     B(i) = C(i) + 2.0
      enddo
```

Grafo de Dependências:



Novo código após Vetorização:

```
S1:     A(2:N) = B(2:N)
S3:     E(2:N) = C(3:N+1)
      DO i=2,N
S2:     C(i) = A(i) + B(i-1)
S4:     B(i) = C(i) + 2.0
      ENDDO
```

Vetorização de Loops (cont.)

Operações de Redução:

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

Estas operações são reconhecidas por alguns compiladores!

Novo código após Vetorização:

```
S1:     A(1:N) = B(1:N) + C(1:N)
S3:     ASUM = ASUM + SUM(A(1:N))
```

`SUM()`: Função implantada em bibliotecas do sistema

Paralelização de Loops

- **Esquema básico de paralelização**: Dividir as iterações do loop pelos vários processadores
- **Estratégia**: Dividir igualmente o trabalho pelos processadores, minimizando o tempo de sincronização necessária
- **Caso Ideal**: Iterações independentes (não há dependências, ou todas elas são do tipo $\delta_{=}$)
- **Outros Casos**: Iterações dependentes → deve haver comunicação/sincronização entre os processadores

Paralelização de Loops (cont.)

Exemplo:

```
    DO i=1,N
       DO j=2,N
S1:       A(i,j) = B(i,j) + C(i,j)
S2:       C(i,j) = D(i,j) / 2
S3:       E(i,j) = A(i,j-1)**2 + E(i,j-1)
       ENDDO
    ENDDO
```

Dependências:

$S_1\ \delta^a_{=,=}\ S_2$ (devido a C)
$S_1\ \delta_{=,<}\ S_3$ (devido a A)
$S_3\ \delta_{=,<}\ S_3$ (devido a E)

Obs: Todas as dependências são $\delta_{=}$ na direção `i`

∴ Loop `i` pode ser paralelizado!

Paralelização de Loops (cont.)

Outro Exemplo:

```
    DO i=2,N
S1:       A(i) = B(i) + C(i)
S2:       C(i) = D(i) * 2
S3:       E(i) = C(i) + A(i-1)
    ENDDO
```

Dependências:

$S_1\ \delta^a_{=}\ S_2$(devido a C), $S_2\ \delta_{=}\ S_3$(devido a C), $S_1\ \delta_{<}\ S_3$(devido a A)

Obs: Por causa da dependência $\delta_{<}$ as iterações são **dependentes**!

→ Possível Corpo do loop para execução paralela:

```
S1:  A(i) = B(i) + C(i)
     signal(i)
S2:  C(i) = D(i) * 2
     if (i>2) wait(i-1)
S3:  E(i) = C(i) + A(i-1)
```