并行编译与优化
Parallel Compiler and Optimization

计算机研究所编译系统室 方建滨

Lecture Fifteen: Loop Transformation (II)

第十五课：循环变换（二）

2024-05-16

学习内容

- **1. 循环变换简介** → 理解循环重要性、循环变换概念及循环变换目的与分类
- **2. 简单循环变换** → 掌握几种典型的简单循环变换
- **3. 高级循环变换** → 掌握几种典型的高级循环变换
- **4. 课堂小结与作业**

循环变换的合法性

- 合法的变换：只有变换后的程序与原循环保持**等价的**变换才是**合法的** (valid)
  - 并不是所有循环变换都是合法的
- 等价的变换：如果在原循环中语句S和语句T之间存在依赖关系，且变换后的程序仍**保持这种依赖关系**，那么变换后的程序就与原循环等价(**equivalent**)

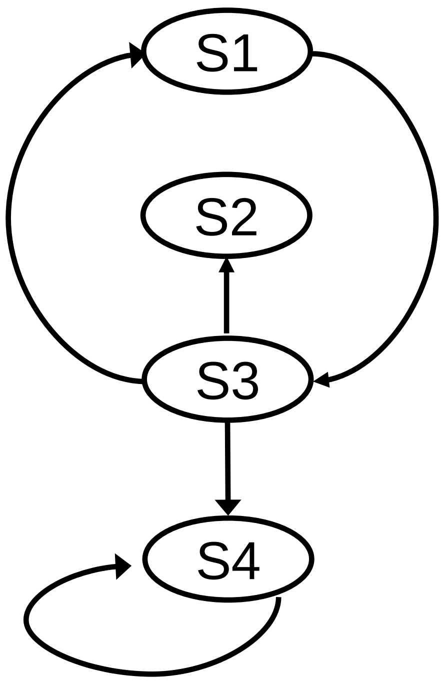
```
L:   do I = 1,N
S1:      A(I) = D(I) * 2
S2:      C(I) = B(I) +A(I)
     enddo
```

```
L1:  do I = 1,N
S1:      A(I) = D(I) * 2
     enddo
L2:  do I = 1,N
S2:      C(I) = B(I) +A(I)
     enddo
```

数据依赖图

- 基本块内数据依赖关系可以用有向无环图（DAG）表示，称为**数据依赖图**
  - 结点表示语句
  - 有向边表示一个数据依赖边起点是依赖源语句
    - 边终点是依赖槽语句

```
        do I=4, 100
S1:       A(I)=B(I-2)+1
S2:       C(I)=B(I-1)+F(I)
S3:       B(I)=A(I-1)+2
S4:       D(I)=D(I-1)+B(I-1)
        enddo
```



3. 高级循环变换

- 3.1 幺模变换 Unimodular transformation
  - 3.1.1 循环交换 Loop interchange
  - 3.1.2 循环置换 Loop permutation
  - 3.1.3 循环反转 Loop reversal
- 3.2 循环合并 Loop fusion
- 3.3 循环分布 Loop distribution
- 3.4 循环分块 Loop tiling

高级循环变换不一定是合法的，故需要判定循环变换的合法性

幺模变换

变换矩阵 (Transformation Matrix)

```
L1  do i=1, 10
L2     do j=2, 10
S         A(i, j) = A(i, j-1)+B(i)
       enddo
    enddo
```

```
L2  do j=2, 10
L1     do i=1, 10
S         A(i,j) = A(i, j-1)+ B(i)
       enddo
    enddo
```

$$(L1, L2) \cdot \boxed{\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}} = (L2, L1)$$

transformation matrix

幺模变换

- A unimodular matrix is a **square matrix** with all **integral** components and with a **determinant of 1** or **-1**

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \qquad \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \qquad \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

幺模变换

- A unimodular transformation represented by the matrix U is legal when applied to a loop nest with a set of non-negative distance vectors D, iff

  for each $\vec{d} \in D$, satisfying $\mathrm{U} \cdot \vec{d} \geq \vec{0}$

- 典型的幺模变换
  - 循环交换、循环置换
  - 循环反转、循环扭曲 (loop skewing)

Michael E. Wolf, Monica S. Lam: A Data Locality Optimizing Algorithm. PLDI 1991: 30-44

什么是循环交换?

- 循环交换(loop interchange)是交换相邻两层循环的次序

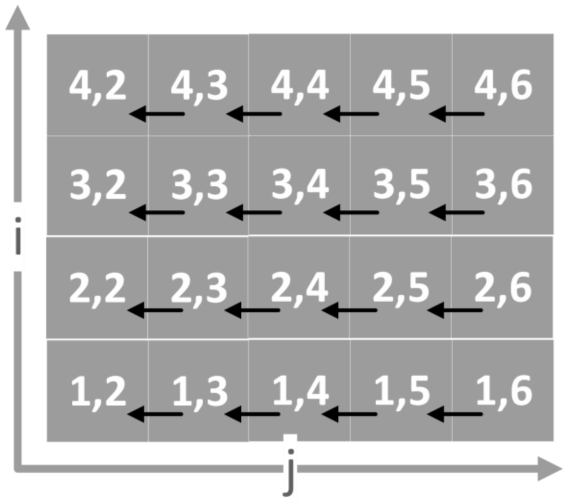
L_1 do i=1, *100*
L_2 do j=2, *10*
$A(i, j) = A(i, j\text{-}1)+B(i)$
enddo
enddo

L_2 do j=2, *10*
L_1 do i=1, *100*
$A(i,j) = A(i, j\text{-}1)+ B(i)$
enddo
enddo

$$(L1, L2) \cdot \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} = (L2, L1)$$

循环交换：示例

$$\begin{bmatrix}0 & 1\\1 & 0\end{bmatrix}\cdot\begin{bmatrix}0\\1\end{bmatrix}=\begin{bmatrix}1\\0\end{bmatrix}\geq\begin{bmatrix}0\\0\end{bmatrix}$$



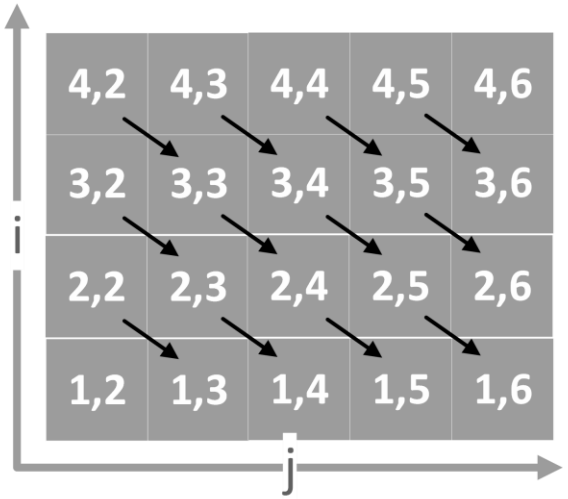

```
L1  do i=1, 10
L2     do j=2, 10
S          A(i, j) = A(i, j-1)+B(i)
        enddo
     enddo
```

$S\ \delta^{f}_{<0,1>}\ S$

```
L2  do j=2, 10
L1     do i=1, 10
S          A(i,j) = A(i, j-1)+ B(i)
        enddo
     enddo
```

循环交换：示例

$$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ -1 \end{bmatrix} = \begin{bmatrix} -1 \\ 1 \end{bmatrix} \ngtr \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$



L_1 do i=2, *10*
L_2 do j=2, *10*
S $A(i, j) = A(i\text{-}1, j+1)+B(i)$
enddo
enddo

L_2 do j=2, *10*
L_1 do i=2, *10*
S $A(i,j) = A(i\text{-}1, j+1)+ B(i)$
enddo
enddo

S $\delta^{f}_{<1,-1>}$ S

循环交换的目的

- **可以改善存储访问的空间局部性**
  - **使得读入至cache的相邻元素能够立即被使用**
- **变换循环使之可以并行化**

```
L1  do i=1, 100
L2      do j=2, 10
            A(i, j) = A(i, j-1)+B(i)
        enddo
    enddo
```

→

```
L2  do j=2, 10
L1      do i=1, 100
            A(i, j) = A(i, j-1)+ B(i)
        enddo
    enddo
```

循环置换

- 循环置换（loop permutation）是一种广义的循环交换，允许一次移动若干个循环，且不要求这些循环是相邻的

L_1 do $I = p1,q1$ L_2 do $J = p2,q2$ L_3 do $K = p3,q3$ $X(I, J, K) =$ $X(I\text{-}3, J\text{-}4, K\text{+}2) + 1$ enddo enddo enddo	L_2 do $J = p2,q2$ L_3 do $K = p3,q3$ L_1 do $I = p1,q1$ $X(I,J,K) =$ $X(I\text{-}3,J\text{-}4,K\text{+}2) + 1$ enddo enddo enddo

$(L_1, L_2, L_3) \rightarrow (L_2, L_3, L_1)$

什么是置换矩阵?

- **置换矩阵P是一类特殊的幺模矩阵，并满足**
  - ① **矩阵的每个元素的值是0或1**
  - ② **矩阵的每行有且只有一个值为1的元素**
  - ③ **矩阵的每列有且只有一个值为1的元素**

$$(L_1, L_2, L_3) \longrightarrow (L_2, L_3, L_1)$$

$$(L_1, L_2, L_3)\begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} = (L_2, L_3, L_1)$$

循环置换：示例

```
    do I1 = p1,q1
      do I2 = p2,q2
        do I3 = p3,q3
S:        X(I1,I2,I3 )
          = X(I1-3, I2-4, I3+2) +1
        enddo
      enddo
    enddo
```

σ=(1,1,-1)

$$P = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}$$

$P\sigma$=(-1,1,1)<0

```
do I3 = p3,q3
   do I2 = p2,q2
      do I1 = p1,q1
         X(I1,I2,I3 )
            = X(I1-3, I2-4, I3+2) +1
      enddo
   enddo
enddo
```

循环置换：示例

```
do I1 = p1,q1
   do I2 = p2,q2
      do I3 = p3,q3
         X(I1,I2,I3 )
       = X(I1-3,I2-4,I3+2) +1
      enddo
   enddo
enddo
```

σ=(1,1,-1)

$$P = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}$$

$P\sigma$=(1,-1,1) > 0

```
do I3 = p3,q3
   do I1 = p1,q1
      do I2 = p2,q2
         X(I1,I2,I3 )
       = X(I1-3,I2-4,I3+2) +1
      enddo
   enddo
enddo
```

什么是循环反转?

- 循环反转（loop reversal）反转一个循环迭代的执行顺序
- 幺模矩阵把要变换循环维度对应的1变为-1

$$\begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} \qquad \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

```
  do i=1,100
    do j=1, 5
S     a(i, j)=a(i-1, j+1)+1
    enddo
  enddo
```

loop reverses →

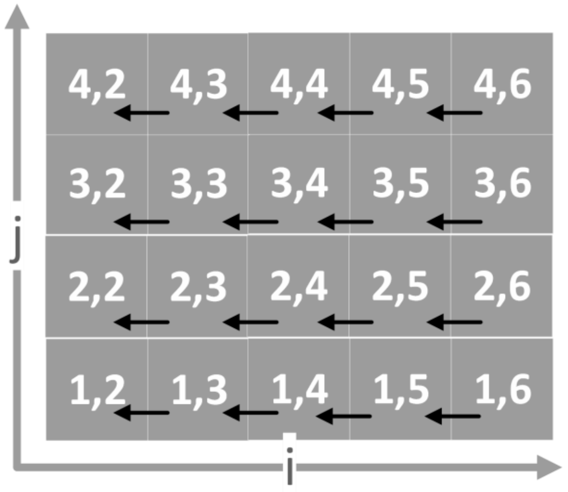
```
  do i=1,100
    do j=5, 1, -1
S     a(i, j)=a(i-1, j+1)+1
    enddo
  enddo
```

$S\ \delta^f_{<1,-1>}\ S$ $\qquad$ $S\ \delta^f_{<1,1>}\ S$

循环反转

```
for i ← 1 by 1 to n do
    for j ← 1 by 1 to n do
        a[i, j] = (a[i-1, j] + a[i+1, j])/2.0
    endfor
endfor
```

$S\ \delta^{f}_{<1,0>}\ S,\ S\ \delta^{a}_{<1,0>}\ S,$



$$\begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} -1 \\ 0 \end{bmatrix}$$

```
for i ← n by -1 to 1 do
    for j ← 1 by 1 to n do
        a[i, j] = (a[i-1, j] + a[i+1, j])/2.0
    endfor
endfor
```

应用实例

```
   do i=1,100
      do j=1, 5
S        a(i, j)=a(i-1, j+1)+1
      enddo
   enddo
```

$S\ \delta^f_{<1,-1>}\ S$

loop reverses

```
   do i=1,100
      do j=5, 1, -1
S        a(i, j)=a(i-1, j+1)+1
      enddo
   enddo
```

$S\ \delta^f_{<1,1>}\ S$

loop interchange

```
do j=5, 1, -1
c$omp parallel
   do i=1,100
      a(i, j)=a(i-1, j+1)+1
   enddo
enddo
```

$S\ \delta^f_{<1,1>} S$ $(S\ \delta^f_{1}\ S)$

什么是循环合并?

- 循环合并（loop fusion）将两个相邻的**具有相同迭代空间的**循环合并成一个循环

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

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

循环合并后

循环合并的合法性判定条件

- **一个循环合并是合法的，仅当**
  1. **循环具有相同的循环上下界，且**
  2. **合并的循环中不存在第一个循环中的语句依赖于第二个循环中语句**

不存在违反原循环中语句顺序的依赖关系

循环合并的合法性：例子

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

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

$$S1 \xrightarrow{\delta^f_=} S2 \qquad S3$$

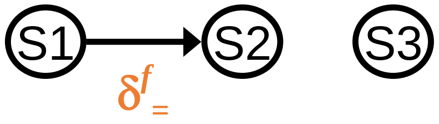
$$S1 \xrightarrow{\delta^f_=} S2 \xleftarrow{\delta^a_1} S3$$

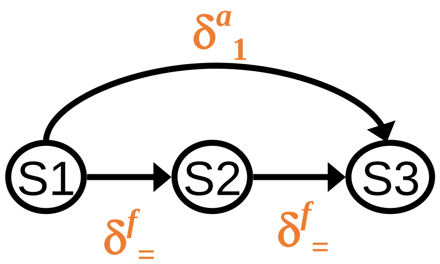
循环合并的合法性：例子

```
      do  i = 1, N
S1        A(i) = B(i) + 1
S2        C(i) = A(i) / 2
      enddo
      do i = 1, N
S3        B(i-1) = C(i)
      enddo
```

After loop fusion:

```
      do  i = 1, N
S1        A(i) = B(i) + 1
S2        C(i) = A(i) / 2
S3        B(i-1) = C(i)
      enddo
```





循环合并的作用

- 减少循环控制开销
- 减少多线程间同步开销
- 增大循环体，为其它优化提供更多机会
  - 标量优化、指令调度等
- 副作用：
  - 循环体增大，降低指令Cache局部性
  - 可能增加寄存器分配的压力

什么是循环分布？

- 循环分布loop distribution（或分裂fission）是循环合并的逆过程，是将一个包含多条语句的循环分裂成具有相同迭代空间的两个循环，其中一个循环包含原循环中的一些语句，第二个循环则包含另外一些语句

```
      do I=4, 100
S1:     A(I)=B(I-2)+1
S2:     C(I)=B(I-1)+F(I)
S3:     B(I)=A(I-1)+2
      enddo
```

→

```
      do I=4, 100
S1:     A(I)=B(I-2)+1
S2:     C(I)=B(I-1)+F(I)
      enddo
      do I=4, 100
S3:     B(I)=A(I-1)+2
      enddo
```

循环分布的合法性

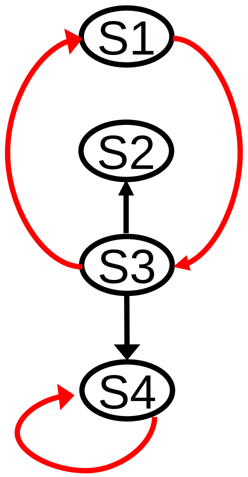
一个循环分布是合法的

- 判定方法1：该变换没有破坏原循环依赖图的**圈**
- 判定方法2：在原循环的**数据依赖图**中形成**强连通分支**的那些语句仍然在循环分布后的**同一个循环**中

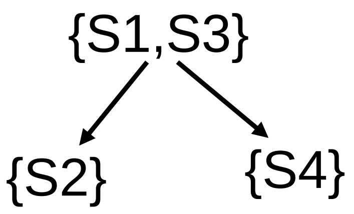
注：在有向图G中，如果任意两个不同的顶点相互可达，则称该有向图是强连通的。有向图G的极大强连通子图称为G的强连通分支。

循环分布

```
        do I=4, 100
S1:       A(I)=B(I-2)+1
S2:       C(I)=B(I-1)+F(I)
S3:       B(I)=A(I-1)+2
S4:       D(I)=D(I-1)+B(I-1)
       enddo
```



strong connected components




```
L1:   do I=4, 100
S1:      A(I)=B(I-2)+1
S3:      B(I)=A(I-1)+2
      enddo
```

```
L2:   do I=4, 100
S2:      C(I)=B(I-1)+F(I)
      enddo
```

```
L3:   do I=4, 100
S4:      D(I)=D(I-1)+B(I-1)
      enddo
```

L1, L2, L3 or L1, L3, L2

循环分布的步骤

- 给定循环L语句的依赖关系图G(V,δ)，循环分布的步骤包括：
  - 找出依赖图中所有的最大连通分量，并由此得到
    凝聚图condensation，记为G' (ℂ, ≤)
    ① ℂ ={ C | C is a strongly connected components of G},
    ② ≤ is the partial order in ℂ
  - 每个连通分量C_i形成一个循环L_i，循环中只包含属于该连通分量的语句
  - 找出循环分布后各个循环{L1, L2, …, Ln}的**合法排列顺序**，并按此顺序安排循环

凝聚图

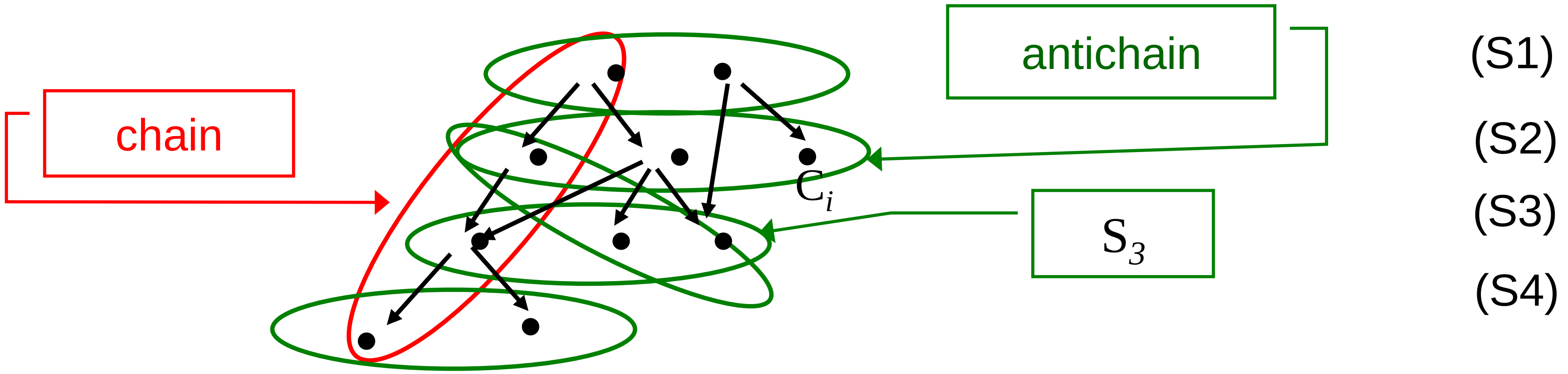
- **一个图G(V, E)的凝聚图G'(V', E')是一个有向图**
  - **将图G中的每一个强连通分支替换为G'一个结点**
  - **对于一个强连通分支的有序对(C_1, C_2)，如果集合$\{(u, v)\in E: u\in C_1, v\in C_2\}$不为空，那么将它替换为从$C_1$到$C_2$的一条边**

By Tarjan algorithm

Alan Gibbons. *Algorithmic Graph Theory*. Cambridge University Press, New York,New York. 1985

循环的合法顺序

- 寻找循环$\{L_1, L_2, \ldots, L_n\}$的合法排列顺序
  - 在凝聚图中，链(chain)的结点之间存在≤关系，反链(antichain)的任何两个结点之间不存在≤关系
  - 给定凝聚图G’，按照最多反链(S1, S2, ..., Sm)的方式组织强连通分支，其中m是图G’中最长链的长度，并满足：
    1. ① 如果i<j，就不会有S_i的前辈在S_j中
    2. ② 每一个S_i至少只有一个直接前辈在S_{i-1}中

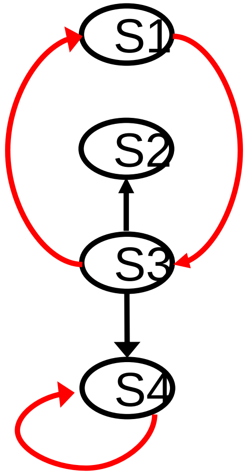


循环的合法顺序

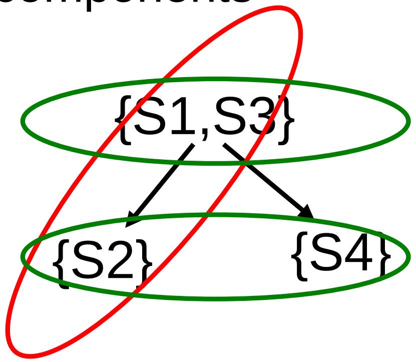
- 一个循环分布是合法的循环变换，当该变换**按如下方式**作用于循环L并改变语句的串行执行顺序
  - 同时开始执行反链S_1中分量对应的所有循环
  - 对于$1<i\leq m$，当反链S_i中分量C在反链S_{i-1}中的所有直接前辈完成执行时，开始执行分量C对应的循环
- 按凝聚图最大反链的层次依次执行各强连通分量
  - 层与层之间按从高到低的顺序执行
  - 同层的在层内可以按任意顺序执行

循环分布

```
        do I=4, 100
S1:       A(I)=B(I-2)+1
S2:       C(I)=B(I-1)+F(I)
S3:       B(I)=A(I-1)+2
S4:       D(I)=D(I-1)+B(I-1)
        enddo
```



strong connected components




```
L1:   do I=4, 100
S1:       A(I)=B(I-2)+1
S3:       B(I)=A(I-1)+2
      enddo
```

```
L2:   do I=4, 100
S2:       C(I)=B(I-1)+F(I)
      enddo
```

```
L3:   do I=4, 100
S4:       D(I)=D(I-1)+B(I-1)
      enddo
```

L1,L2,L3 or L1,L3,L2

循环分布的作用

- 并行化循环
- 向量化循环

循环分块

- 循环分块等价于先进行循环分段，再将分段的循环交换至最外层

  Loop tiling = strip mine and loop interchange

- 循环分块的对象是嵌套循环

循环分块

srtip mine

interchange srtip loops $L\text{-}I_s$ and $L\text{-}J_s$
to outmost level

```
do  I=1,m
    do  J=1,n
        H(I, J)
    enddo
enddo
```

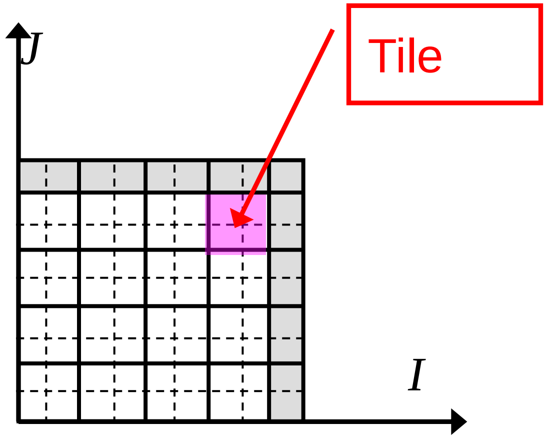
```
do  Is= 1, m, s
  do  I=Is, min(m, Is+s-1 )
     do  Js=1, n, t
         do  J=Js, min(n, Js+t-1)
             H(I, J)
         enddo
     enddo
  enddo
enddo
```

```
do  Is= 1, m, s
  do  Js=1, n, t
     do  I=Is, min(m, Is+s-1 )
        do  J=Js, min(n, Js+t-1)
           H(I, J)
        enddo
     enddo
  enddo
enddo
```

循环分块

例子

```
do  I=0, 5
    do  J=0, 8
          A(I, J)=A(I, J)+1
    enddo)
enddo
```

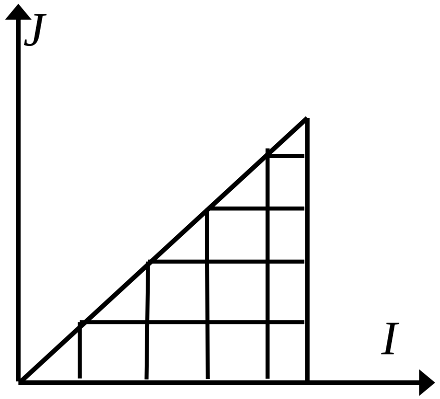



```
do  Is= 0, 5, 2
   do  Js=0, 8, 2
      do  I=Is, min(5, Is+1)
          do  J=Js, min(8, Js+1)
             A(I, J)=A(I, J)+1
          enddo
      enddo
   enddo
enddo
```

循环分块

三角循环嵌套

```
do  I=1, m
    do  J=I, n
        H(I, J)
    enddo
enddo
```




```
do  Is= 1, m, s
   do  Js=1, n, s
      do  I=Is, min(m, Is+s-1)
         do  J= max(Js, I), min(n, Js+s-1)
              H(I,J)
         enddo
      enddo
   enddo
enddo
```

循环分块的合法性

- **如果循环L_i和L_j循环是可置换的，那么循环L_i到L_j循环是可分块的**
- **循环L_i到L_j循环是可置换的，当且仅当**

所有的依赖向量都是正的，且

对于每个依赖向量，

① $(d_1,\ldots,d_{i-1})$为正；或，

② 要求di到dj不为负

数据分块

例子

```
do  i=1, n
   do  j=1, n
       Z(i,j)=0.0
       do k=1, n
           Z(i, j)=Z(i, j)
                   +X(i,k)*Y(k, j)
         enddo
    enddo
enddo
```

```
      do 100  i=1, n
         do 100 j=1, n
             Z(i,j)=0.0
100 continue

   do i=1, n
      do j=1, n
         do k=1, n
            Z(i, j)=Z(i, j)
                  +X(i,k)*Y(k, j)
          enddo
      enddo
   enddo
```

数据分块

例子

```
do  i=1, n
   do  j=1, n
      do k=1, n
           Z(i, j)=Z(i, j)
                   +X(i,k)*Y(k, j)
         enddo
    enddo
enddo
```

```
        do 200  ii= 1, n, B
        do 200  jj=1, n, B
        do 200  kk=1, n, B
            do 200  i=ii, ii+B-1
            do 200  j=jj, jj+B-1
            do 200  k=kk, kk+B-1
                   Z(i, j)=Z(i, j)
                            +X(i,k)*Y(k, j)
200 continue
```

如何选择·分块大小?

循环分块的目的

- 块之间具有并行性
- 块内部的向量化
- 块内部的数据重用
- 块之间的数据重用

课堂小结

- 针对给定程序代码进行**循环编译优化**的步骤
  - **识别**程序中的循环（回顾）
  - **分析**一个循环是否可被并行化/向量化/循环分块等
    - 如果不能，是否有合适的**循环变换**改变循环形式
  - 循环优化目标：并行化/向量化/循环分块等

课堂小结

- 循环变换关键是找寻一个新的**合适的、合法的**迭代执行序列
  - 合适的：根据收益模型
  - 合法的：不违背依赖关系，保持方向向量为“正”

开放问题

- 循环变换的代价/好处
- 循环变换相互影响，如何选择?
- 建议手工改写程序，测试各种循环变换的的优化结果

参考资料

- 龙书第11章
- Optimizing Compilers Modern Architecture, chap5~6, chap 8, chap9.1~9.4
- Utpal Banerjee, Loop Transformations for Restructuring Compilers, Kluwer A. Publishers,1993

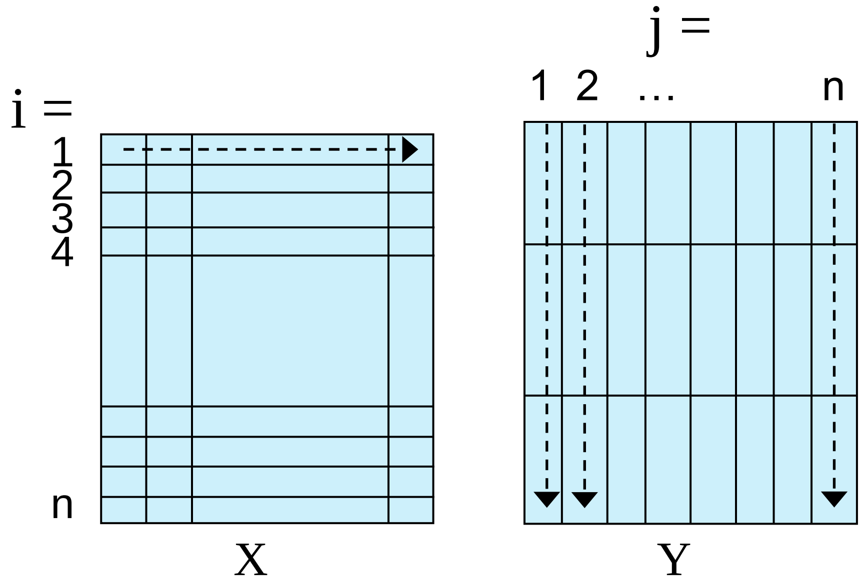
Backup Slides

数据分块

例子

```
do  i=1, n
   do  j=1, n
      do k=1, n
           Z(i, j)=Z(i, j)
                   +X(i,k)*Y(k, j)
         enddo
    enddo
enddo
```

suppose cache line size：c
n divies c



cache misses: total: $n^2/c + n^3/c$

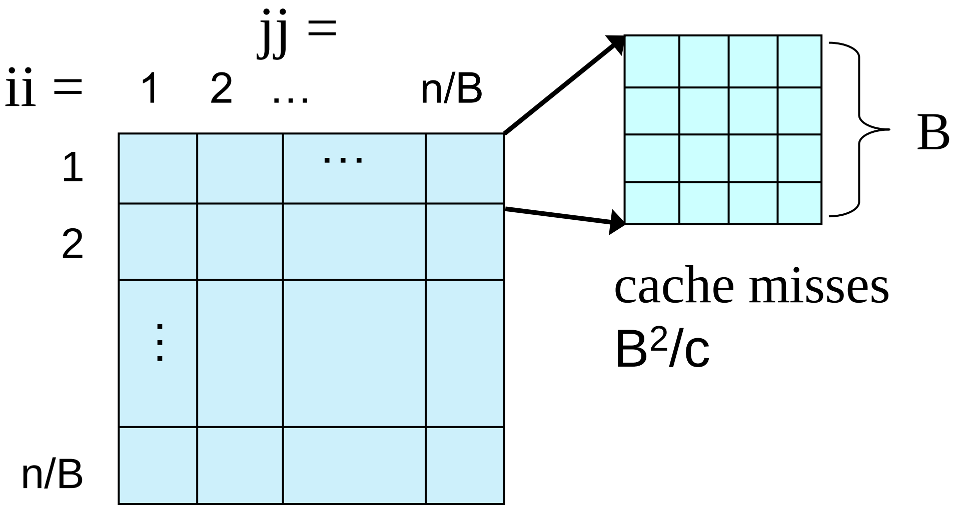
X	miss	Y	miss	
a elem.		a column	n/c	
a line	n/c	n column	n^2/c	
n line	n^2/c	n*n column	n^2/c	n^3/c

数据分块

例子

```
do 10  ii= 1, n, B
do 10  jj=1, n, B
do 10  kk=1, n, B
    do 10  i=ii, ii+B-1
    do 10  j=jj, jj+B-1
    do 10  k=kk, kk+B-1
        Z(i, j)=Z(i, j)
            +X(i,k)*Y(k, j)
10 continue
```

B(ii, jj, kk)



tatol cache misses (tiling)：

$$(n^3/B^3) \times (2B^2/c) = 2n^3/Bc$$

total cache misses (not tiling) :

$$n^2/c + n^3/c \text{ (or } n^2/c)$$