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 17 Parallelization

第十七课 并行化

2024-05-30

学习内容

- 1. 并行化简介
- 2. 并行化判定条件
- 3. 循环变换与并行化
- 4. 自动并行化编译实现

软件革命：无处不在的并行

David Patterson

ACM&IEEE院士，2017年图灵奖得主，ACM前任主席，提出RISC, RAID, NOW等

"From my perspective, parallelism is the biggest challenge since high-level programming languages. It is the biggest thing in 50 years because industry is betting its future that parallel programming will be useful."

"在我看来，并行性是自从高级程序设计语言出现以来的最大挑战。在今后50年中它都将是最重要的课题，因为工业界已经把自己的未来押宝在'并行程序设计是可行的'这一假定上。"

ACM Queue Interview with John Hennessy and David Patterson, 2007.01

软件革命：无处不在的并行

"Sequentiality is an illusion"

source: https://www.cs.virginia.edu/~skadron/

Kevin Skadron ACM/IEEE Fellow
Harry Douglas Forsyth Professor of Computer Science
Department of Computer Science
School of Engineering and Applied Science
University of Virginia

无处不在的并行

COMP CCRG Open MP

- 指令级并行
  - 超标量流水线： 通过多个功能部件，使处理器每个时钟周期可流出并处理多条指令
- 向量级并行
  - 最早用在60年代的向量机
  - 单指令多数据（SIMD）结构
  - 128位 → 256位→512位→1024位

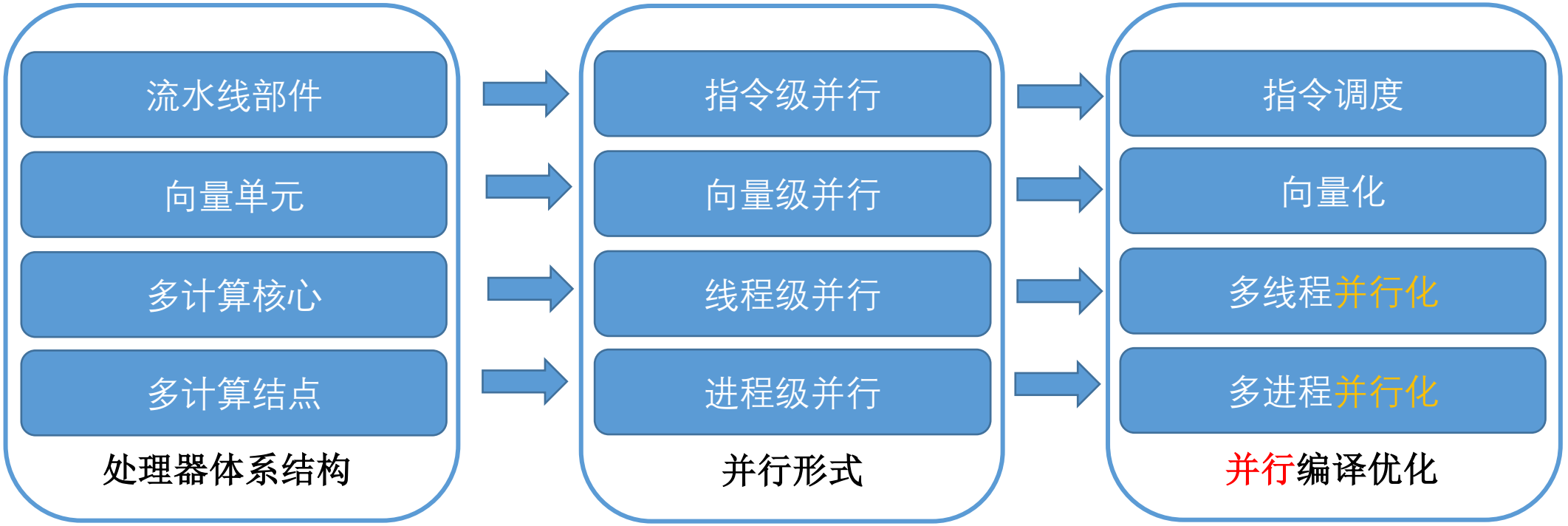
无处不在的并行

- 核心级并行：多核处理器
  - 每个处理器核拥有独立的寄存器文件、流水线及计算单元
  - 通过使用多线程(或者多进程)并行来加速程序运行
- 结点级并行：集群系统或超算系统
  - 通过互连将多处理器连接在一起，共同完成一个计算任务
  - 90年代的Beowulf结构流行至今，使用多进程+多线程

无处不在的并行

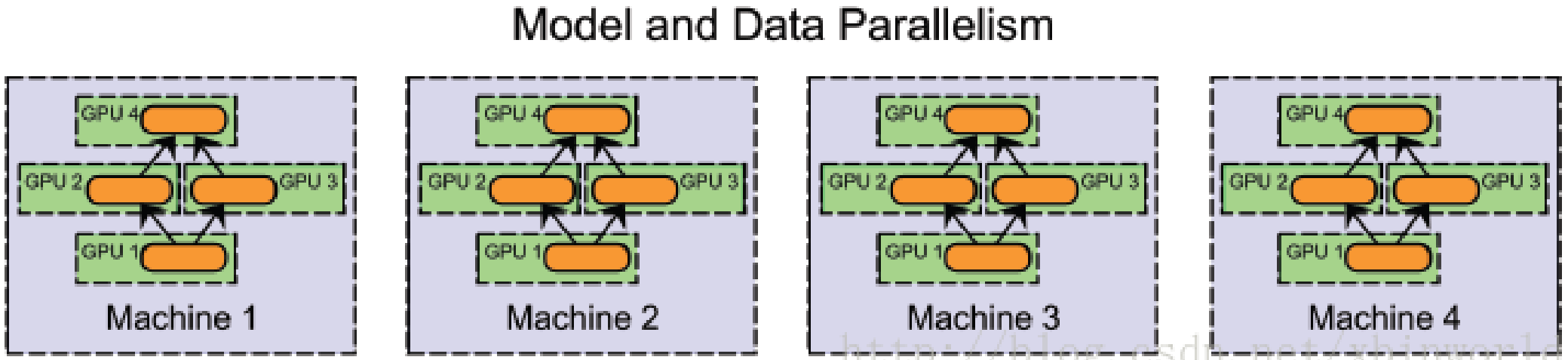
并行形式与编译优化

- 本讲关注狭义并行，包括多线程并行和多进程并行



什么是并行化?

- 并行化就是将程序划分成许多可并发执行子任务并映射到不同线程或进程的过程
- 并行化是为了利用多进程或多线程，从而并发利用不同的计算核心或计算结点



并行化对象

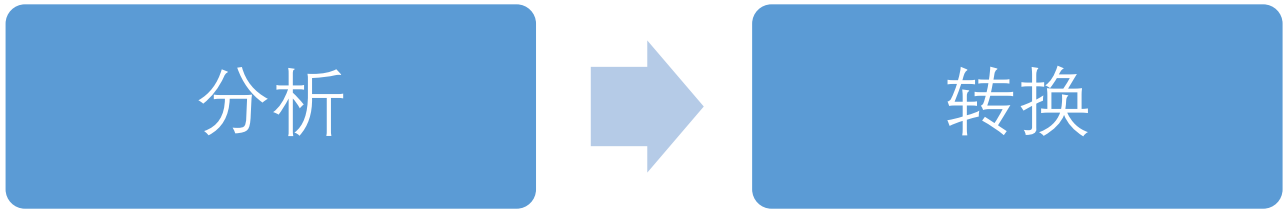
- 并行化对象可以是子程序、循环和语句块
- 循环是并行化重点考虑的对象
  - 循环占程序大部分的运行时间
  - 循环的并行化分析和转换相对容易
- 对嵌套循环选择可并行化的外层循环作并行化
  - 组织循环并行执行的开销一般比组织循环向量执行的开销大
  - 对于向量处理器，同时还可将内层循环向量化

并行化约束

- 多个任务并行执行的前提是不破坏程序中原有的数据依赖和控制依赖关系
  - 若两个任务间不包含任何依赖，则它们可以并行执行
  - 适当的同步通信可保证含有任何依赖关系的两个任务的并行
- 并行化主要针对不含跨迭代依赖关系的循环

循环并行化

- 循环并行化是把循环的迭代分配给各个处理器上去执行
- 串行程序并行化时，首先进行**依赖关系分析**
  - 若可以并行化，则直接进行并行化；
  - 否则需要进行程序转换，力图使之并行化



并行化方式

- 手工并行化 (manual parallelization)
  - 基于源语言或编程接口进行程序并行化
  - 使用并行语言或编译指导命令来描述并行
    - MPI, Pthreads, OpenMP, CUDA, OpenCL, SYCL, …
  - 手工并行化程序一项很耗时且容易出错的任务
- 自动并行化 (auto-parallelization)
  - 编译器自动分析和转换串行程序，并生成并行形式的代码

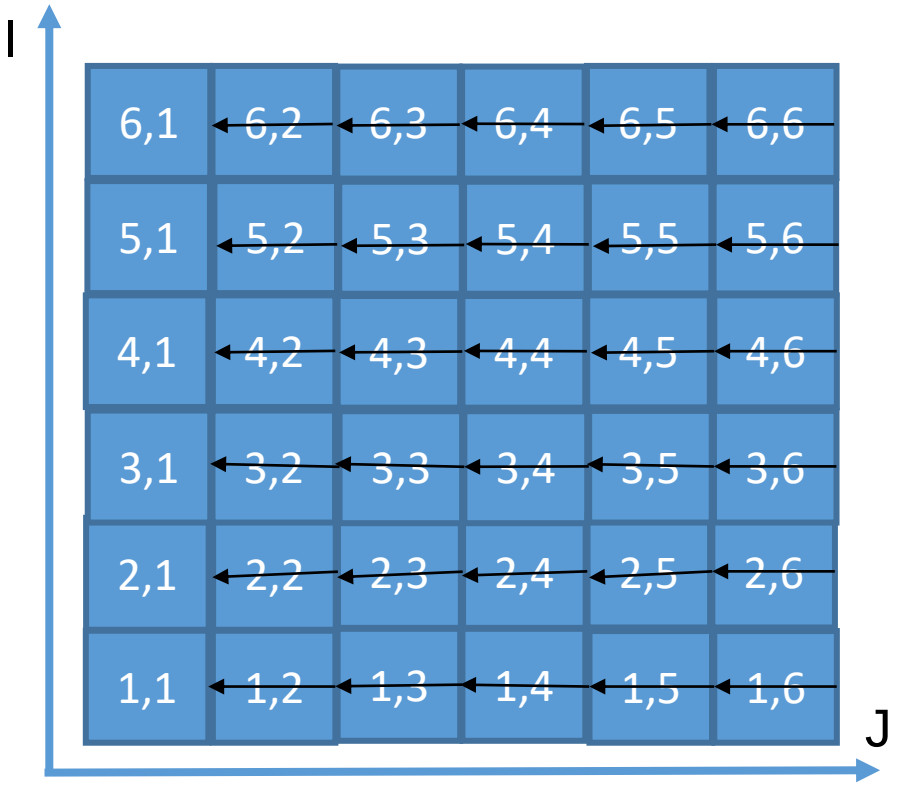
学习内容

- 1. 并行化简介
- 2. 并行化判定条件
- 3. 循环变换与并行化
- 4. 自动并行化编译实现

可并行化循环的定义

- 一个循环是可并行化循环，如果它的各个迭代可按任何次序执行，且结果与串行执行时相同

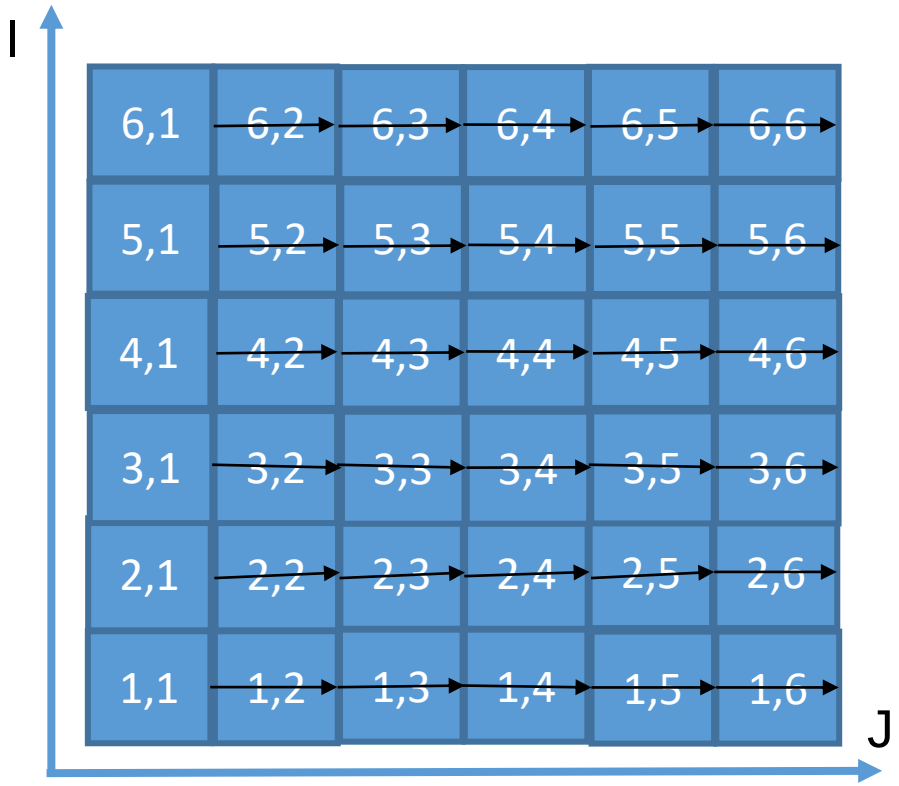
```
L:  do  I = 1, N
        do  J = 1, M
S :         X(I, J) = X(I, J-1) + X(I, J+1)
        enddo
    enddo
```



可并行化循环的定义

- 一个循环是可并行化循环，如果它的各个迭代可按任何次序执行，且结果与串行执行时相同

```
L:  do  I = 1, N
        do  J = 1, M
S :         X(I, J) = X(I, J-1) + X(I, J+1)
        enddo
    enddo
```



可并行化循环的定义

- 一个循环是可并行化循环，如果它的各个迭代可按任何次序执行，且结果与串行执行时相同

```
L:  do  I = 1, N
        do  J = 1, M
S :       X(I, J) = X(I, J-1) + X(I, J+1)
        enddo
    enddo
```

L的内层循环J不能被并行化

L的外层循环I可以被并行化

可并行化循环的判定条件

- 定理：一个循环是可并行化的，当且仅当循环体中不存在依赖方向不为“=”的依赖关系

充分必要性证明

必要性证明

定理：一个循环是可并行化的，当且仅当循环体中不存在依赖方向不为“=”的依赖关系

- **假设循环中有依赖方向不为0的依赖关系，则循环中存在相关迭代对$\{(i, j): j \neq i, p \leq i \leq q, p \leq j \leq q\}$，p、q分别是循环的下、上界。显然这些相关迭代对不能按任意顺序执行，故该循环不能并行化。**

充分性证明

定理：一个循环是可并行化的，当且仅当循环体中不存在依赖方向不为“=”的依赖关系

- **根据假设，循环中只可能有依赖方向为0的依赖关系，即其相关迭代对为{(i, j): j=i, p ≤ i ≤ q}。因为一个迭代分配在一个处理器（或硬件线程）上执行，而这种依赖关系发生在一个迭代内部，因此不论各个迭代的执行次序如何，总能保证结果的正确性。根据可并行化循环的定义，该循环可以并行化。**

嵌套循环可并行化的判定条件

- 定理：嵌套循环$L=(L_1,L_2,\cdots,L_m)$中的第l层循环L_l是可并行化的，当且仅当
  - L中不存在层次为l的依赖关系，或者
  - 不存在方向向量为$\sigma_1=\sigma_2=\cdots\sigma_{l-1}=0$，$\sigma_l=1$，$\sigma_{l+1}=\cdots=\sigma_m=*$的依赖关系

必要性是显然的；
只需证明充分性。

充分性证明

- 在单层循环情况下，如果循环无依赖关系，则可能按任意顺序执行。
- 在多层循环情况下，如果第l层无依赖关系，则L_l的**并行执行**不会破坏L中其他层循环的依赖关系。
  - 假设第L_r层存在依赖关系，其中$1 \le r \le m$，且$r \neq l$，则存在迭代$H(i)\delta H(j)$且$i <_r j$。由于L_r层存在依赖关系，此层**在变换后的循环L'**中仍保持依赖关系，因为在L'中仍有$i <_r j$，即H(i)将在H(j)之前被执行，亦即并行执行不会改变第r层的依赖关系。

嵌套循环可并行化

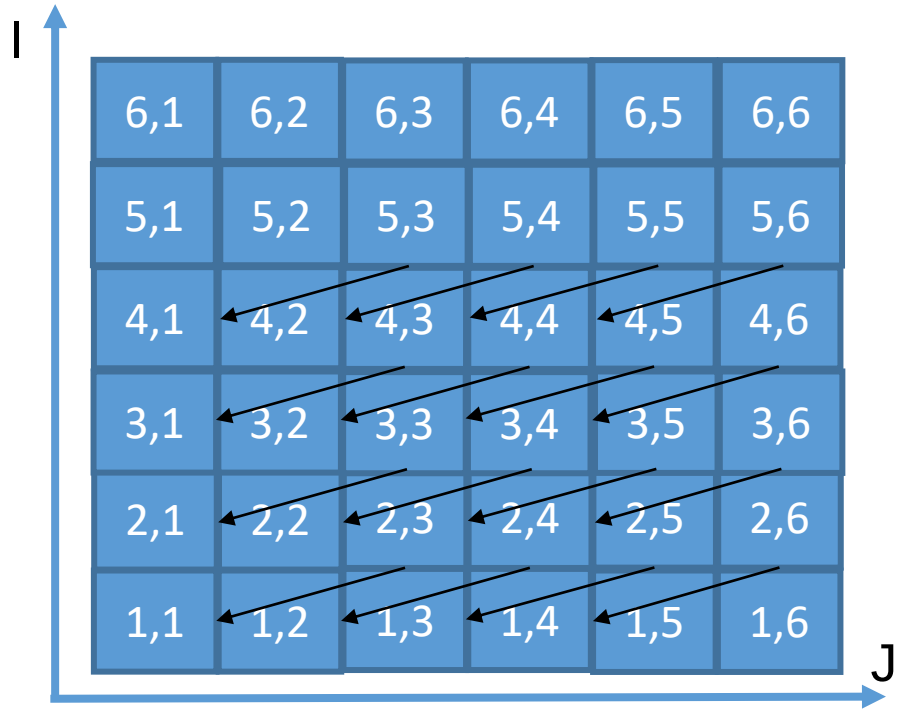
```
L1:  do  I = 1, 4
L2:     do  J = 1, 4
S:        X(I+1, J+2) = Y(I, J) + 1
T:        Y(I+2, J+1) = X(I, J) + 1
        enddo
    enddo
```

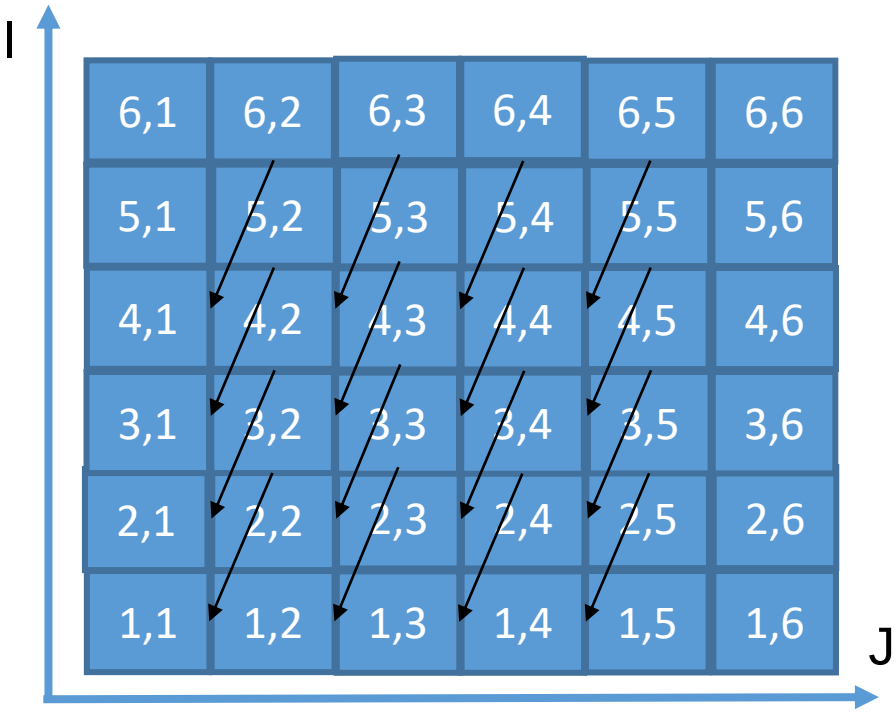
$X: S\delta_{<1,2>}T$ or $S\delta_{<1,1>}T$

$Y: T\delta_{<2,1>}S$ or $T\delta_{<1,1>}S$

外层循环不能并行化

内层循环可以并行化



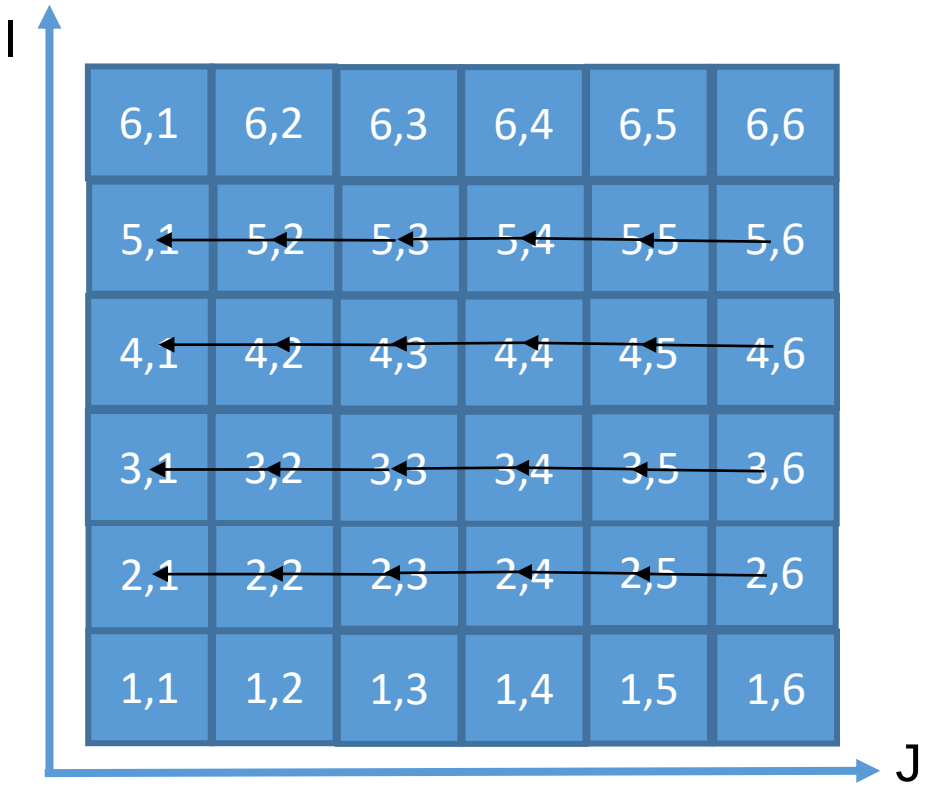


嵌套循环可并行化

```
L1:  do  I = 2, N
L2:     do  J = 2, M
S :       A(I, J) = A(I, J-1) + 1
        enddo
     enddo
```

$A: S\delta_{<0,1>}S$

外层循环可以并行化
内层循环不能并行化



学习内容

- 1. 并行化简介
- 2. 并行化判定条件
- 3. 循环变换与并行化
- 4. 自动并行化编译实现

循环并行化

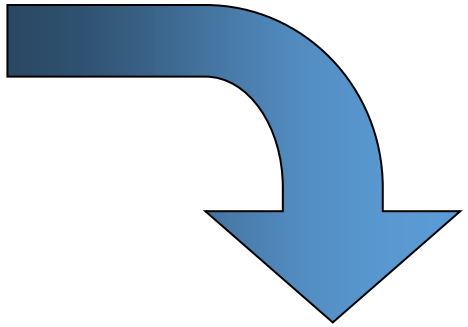
- 当一个循环中有少量的跨迭代依赖时，
  - 通过循环变换、数据私有化等循环优化消除依赖
  - 或者通过同步等机制并行执行这个循环以保持依赖
- 消除依赖的关键在于使用等价的程序变换

循环剥除（1/2）

删除某些数据依赖

```
    do i = 1, n
S:     y(i, n) = y(1, n) + y(n,n)
    enddo
```

i = 1 and i = n 有依赖



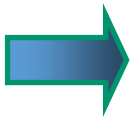
```
    y(1, n) = y(1, n) + y(n,n)
    do i = 2, n-1
S:     y(i, n) = y(1, n) + y(n,n)
    enddo
    y(n, n) = y(1, n) + y(n,n)
```

循环可以并行化

循环剥除（2/2）

消除循环中的分支判断

```
for ( i =1; i < n; i++) {
   if (i == 1)
      a[i] = 0;
   else
      a[i] = b[i-1];
}
```



```
a[1] = 0;
for ( i=2; i < n; i++) {
     a[i] = b[i-1]
 }
```

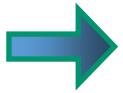
循环可以并行化

循环分裂（1/2）

删除与循环索引变量相关的条件判断

```
do i = 1, 100
   A(i) = B(i) + C(i)
   if i > 10 then
      D(i) = A(i) + A(i-10)
   endif
enddo
```

分裂前



```
do i = 1, 10
   A(i) = B(i) + C(i)
enddo
do i = 11, 100
   A(i) = B(i) + C(i)
   D(i) = A(i) + A(i-10)
enddo
```

分裂后

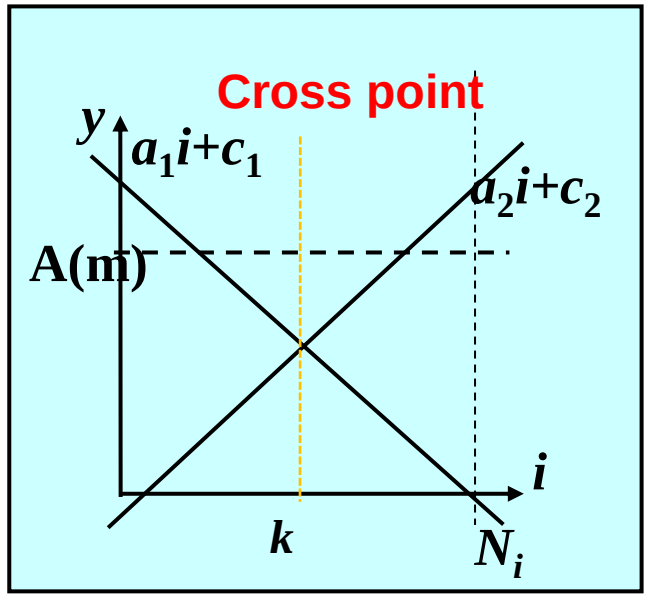
第一个循环可以并行化
第二个循环不能并行化

循环分裂（2/2）

消除数据依赖

```
      do i = 1, n
S:        y(i) = y(n-i+1)
      enddo
```

```
      do i = 1, (n+1)/2
         y(i) = y(n-i+1)
      enddo
      do i = (n+1)/2+1, n
         y(i) = y(n-i+1)
      enddo
```



k=(n+1)/2

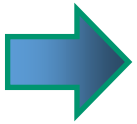
循环可以并行化

循环去开关化

- 循环去开关化是指将循环不变的条件分支移至循环体外

```
do i=1, n
   do j=2, n
      if (t(i) .gt. 0) then
         a(i,j) = a(i, j-1)*t(i)
      else
         a(i,j) = t(i)
      endif
   enddo
enddo
```

变换前



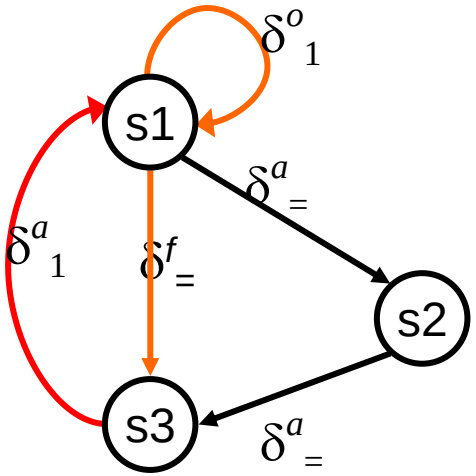
```
do i=1, n
   if (t(i) .gt. 0) then
      do j=2, n
         a(i,j) = a(i, j-1)*t(i)
      enddo
   else
      do j=2, n
         a(i,j) = t(i)
      endo
   endif
enddo
```

变换后

标量扩张

- 标量扩张使得原本不能并行化的循环变得可以被并行化

```
L:    do i=1, N
S1:     T = A[i]
S2:     A[i]=B[i]
S3:     B[i] = T
      enddo
```

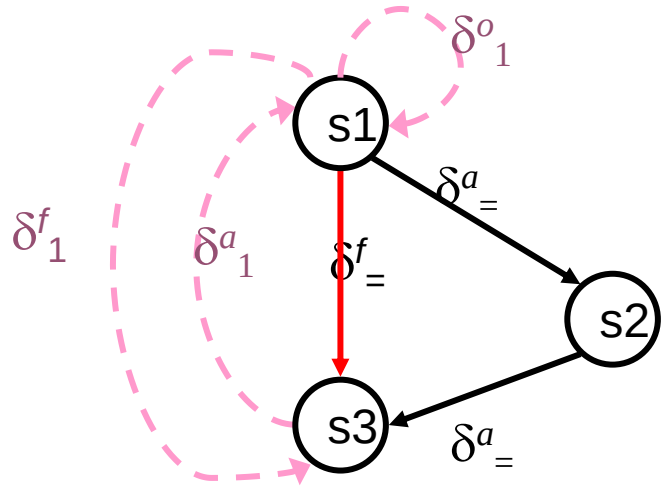


循环L不能被并行化

标量扩张

- 标量扩张使得原本不能并行化的循环变得可以被并行化

```
L′:   do i=1, N
S1:     $T[i] = A[i]
S2:     A[i]=B[i]
S3:     B[i] = $T[i]
      enddo
      T=$T[N]
```



经过标量扩展，循环L’可以被并行化

标量扩张

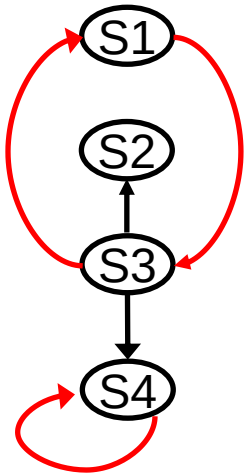
```
      DO I=1,100
S1      A(I)= T
S2      T =B(I)+C(I)
      ENDDO
```

```
      T$(0) = T
      DO I = 1, 100
S1      A(I)  = T$(I-1)
S2      T$(I)= B(I) + C(I)
      ENDDO
      T = T$(100)
```

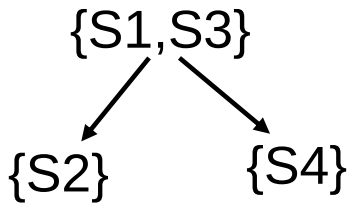
经过标量扩展，循环仍然不能被并行化

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
```

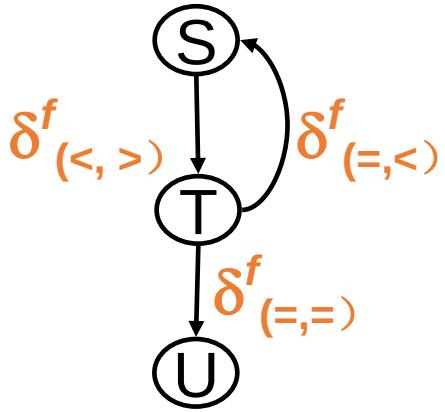
```
L2:   do I=4, 100
S2:     C(I)=B(I-1)+F(I)
      enddo
```

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

L1, L2, L3 or L1, L3, L2

例子：循环分布

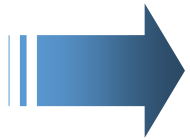
```
L1:   do I1=0, 100
L2:     do I2=0,100
S:        A(I1+1, I2) = B(I1, I2)+C(I1, I2)
T:        B(I1, I2+1) = A(I1, I2+1)+1
U:        D(I1, I2) = B(I1, I2+1)-2
        enddo
      enddo
```



{ S, T }, { U }

例子：循环分布

```
L11:   do I1=0, 100
L12:     do I2=0,100
S:         A(I1+1, I2) = B(I1, I2)+C(I1, I2)
T:         B(I1, I2+1) = A(I1, I2+1)+1
         enddo
       enddo
```

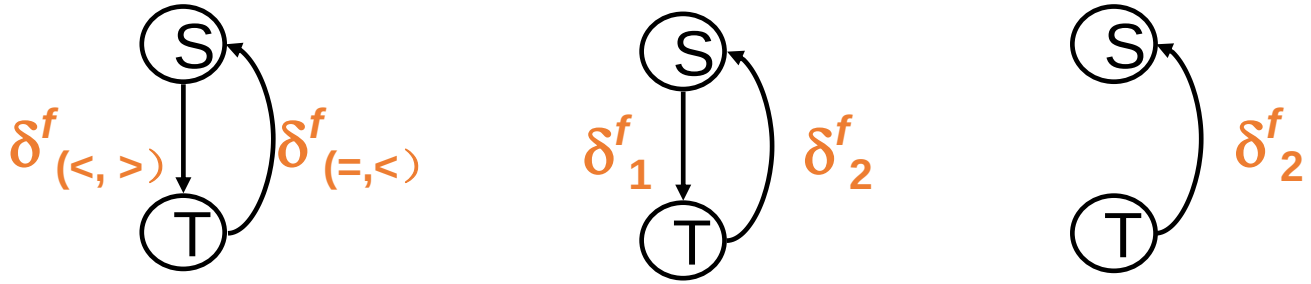


```
L21:   do I1=0, 100
L22:     do I2=0,100
U:         D(I1, I2) = B(I1, I2+1)-2
         enddo
       enddo
```

```
c$omp parallel do
L21:   do I1=0, 100
L22:     do I2=0,100
U:         D(I1, I2) = B(I1, I2+1)-2
         enddo
       enddo
```

例子：循环分布

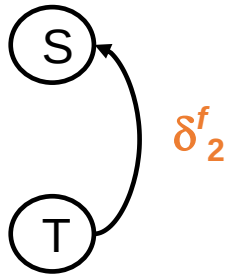
```
L11:   do I1=0, 100
L12:     do I2=0,100
S:         A(I1+1, I2) = B(I1, I2)+C(I1, I2)
T:         B(I1, I2+1) = A(I1, I2+1)+1
         enddo
       enddo
```



两个依赖是在不同的循环层次上，可以按层次分解开来，
分解后不存在相同层次的依赖环，故可进一步进行循环分布

例子：循环分布

```
L11:    do I1=0, 100
c$omp parallel do
L121:     do I2=0,100
T:          B(I1, I2+1) = A(I1, I2+1)+1
          enddo
c$omp parallel do
L122:     do I2=0,100
S:          A(I1+1, I2) = B(I1, I2)+C(I1, I2)
          enddo
        enddo
```



例子：循环分布

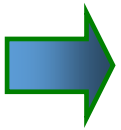
```
      do I1=0, 100
        do I2=0,100
S:        A(I1+1, I2) = B(I1, I2)+C(I1, I2)
T:        B(I1, I2+1) = A(I1, I2+1)+1
U:        D(I1, I2) = B(I1, I2+1)-2
        enddo
      enddo
```

```
      do I1=0, 100
c$omp parallel do
        do I2=0,100
T:        B(I1, I2+1) = A(I1, I2+1)+1
        enddo
c$omp parallel do
        do I2=0,100
S:        A(I1+1, I2) = B(I1, I2)+C(I1, I2)
        enddo
      enddo
c$omp parallel do
      do I1=0, 100
        do I2=0,100
U:        D(I1, I2) = B(I1, I2+1)-2
        enddo
      enddo
```

循环交换

交换相邻的两层循环

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



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

循环交换使外层循环可以并行化

循环反转

反转一个循环迭代的执行顺序

```
   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

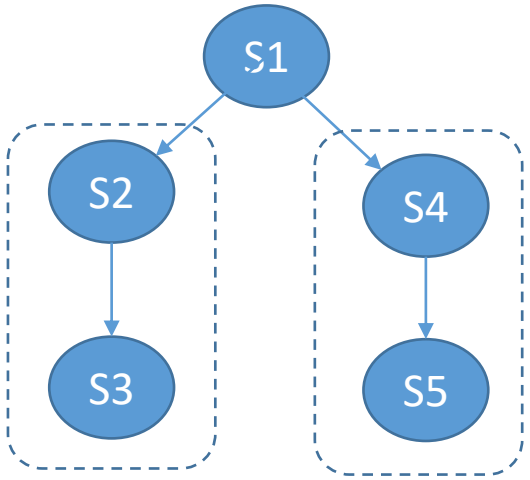
```
do j=5, 1, -1
c$omp parallel
   do i=1,100
      a(i, j)=a(i-1, j+1)+1
   enddo
enddo
```

S $\delta^{f}_{<1,1>}$S (S δ^{f}_{1} S)

非循环代码并行化

- 非循环代码并行化就是将语句序列中的语句划分成多个**可以并行执行的子语句序列**

```
S1: A = 1
S2: B = A +1
S3: C = B + 1
S4: D = A + 1
S5: E = D + F
```



非循环代码并行化技术应用于复合语句、语句基本块及过程调用语句时，可能获得较大粒度的并行。

学习内容

- 1. 并行化简介
- 2. 并行化判定条件
- 3. 循环变换与并行化
- 4. 自动并行化编译实现

自动并行化

- 手工并行化程序一件很耗时的任务，故需要自动并行化
- 自动并行化是并行计算领域最具挑战性的问题
  - 程序代码形式各异
  - 体系结构复杂多样

自动并行化框架

COMP CCRG Open MP

- 自动并行化解决思路
  - 开发编译基础设施来自动生成高效率的并行程序代码
  - 使用模型指导的参数性能调优
- 自动并行化框架
  - Intel C++ compiler
    - 编译选项 /Qparallel
    - 编译指导命令 #pragma parallel
  - 多面体模型：LLVM Polly, PLuTo, PPCG, …
    - C → OpenMP

自动并行化步骤

- 寻找良好的工作共享候选循环
- 依赖关系分析判定该循环是否能被并行化
  - 如果不能，是否存在合适的循环变换
- 划分数据并生成多线程代码

循环并行化的限制

- 循环不能嵌套在已并行化的另一循环内
- 流控制不允许跳出循环
- 用户级子程序不能在循环内被调用
- 循环中不能有I/O语句
- ……

自动并行化的挑战

- 编译器或工具自动并行化非常困难
  - 对于使用间接寻址、指针、递归或间接函数调用的代码，依赖分析非常困难，编译时很难检测到此类依赖
  - 循环迭代次数常常是未知的
  - 在内存分配、I/O和共享变量等方面，存在难以协调对全局资源访问的问题
  - 不规则算法使用依赖于输入的间接访问会干扰编译时分析和优化

基于多面体模型的编译

- 多面体模型：一种仿射程序表示与转换的框架
  - 应用范围广：幺模矩阵只能处理**完美循环嵌套**，而多面体模型对输入程序的约束更少。
  - 表示能力强：幺模矩阵只能表示循环交换、倾斜和反转等优化，而多面体模型还能自动实现包括循环分块、合并、分布等在内的几乎所有循环变换技术
  - 优化空间大：幺模矩阵一次只能实现一种循环变换,而多面体模型允许同时实现多个循环变换

基于多面体模型的编译优化技术仍是国际上编译领域的研究热点！

往年竞赛实现A（1/4）

2022年全国大学生计算机系统能力大赛编译系统设计赛（华为毕昇杯）
全国总决赛获奖名单
[按学校名称拼音排序，队员排名不分先后]

参赛队伍	学校	赛队成员
特等奖		
赫露艾斯塔	清华大学	焦景辉 王建楠 王子元 李欣隆

https://gitlab.eduxiji.net/educg-group-12619-928705/helesta/-/blob/func-inline-2-sub/sub/loop_ops.cpp

往年竞赛实现A（2/4）

COMP CCRG Open MP

实现流程

- 循环的依赖性检查
  - dependent(w) w是循环基本块
- 初始化并行循环副本：为每个线程创建一个循环副本

```
for (size_t i = 1; i <= cnt; ++i)
{
        loops.emplace_back(loops[0]);//创建副本
        loops.back().copy(std::string(":")
                          +std::to_string(i) + ":");
}
```

往年竞赛实现A（3/4）

COMP CCRG Open MP

实现流程

- 创建全局互斥锁和同步变量，以实现线程间的同步
  - Mutex：互斥锁，保护共享资源的并发访问
  - Barrier：进行线程间的同步
- 为每个线程创建并配置循环副本：为每个线程创建新的entry基本块，并配置分支和跳转指令，以实现并行化执行
  - 计算并更新循环上下界
  - 更新entry中的phi节点等

往年竞赛实现A（4/4）

实现流程

- 线程同步：为每个线程创建新的基本块，利用Mutex和Barrier配置同步操作以确保线程之间的正确同步。
- 将线程绑定到指定的核心，可实现线程绑定

```
if (use_bind_core)
      cg.call(bind_core, ScalarType::Void,
            {{cg.lc(0), ScalarType::Int},
            {cg.lc((1 << 4) - 1), ScalarType::Int}});
      cg.jump(next);
      tail->push(std::move(cg.instrs));
```

课堂总结

- 1. 并行化简介（熟悉）
- 2. 并行化判定条件（掌握）
- 3. 循环变换与并行化（掌握）
- 4. 自动并行化编译实现（掌握）

课后作业

- 分析给定循环代码能否被并行化
  - 写出依赖关系，并进行判断
  - 在头歌系统提交

扩展阅读

- Michael E. Wolf, Monica S. Lam: A Loop Transformation Theory and an Algorithm to Maximize Parallelism. IEEE Trans. Parallel Distributed Syst. 2(4): 452-471 (1991)
- 赵捷等，基于多面体模型的编译“黑魔法”，软件学报， 2018 第29卷 第8期 P2371-2396