循环优化系列第四讲

循环交换

嘉宾：柴赟达

循环交换

- **基础概念**

当一个循环体中包含一个以上的循环，且循环语句之间不包含其它语句，则称这个循环为紧嵌套循环，交换紧嵌套中两个循环的嵌套顺序是提高程序性能最有效的变换之一。实际上，循环交换是一个重排序变换，仅改变了参数化迭代的执行顺序，但是并没有删除任何语句或产生任何新的语句，所以循环交换的合法性需要通过循环的依赖关系进行判定。

```
for (j = 0; j < N; j++)
    for (k = 0; k < N; k++)
        for (i = 0; i < N; i++)
            A[i][j] = A[i][j] + B[i][k] * C[k][j];
```

循环交换 →

```
for (j = 0; j < N; j++)
    for (i = 0; i < N; i++)
        for (k = 0; k < N; k++)
            A[i][j] = A[i][j] + B[i][k] * C[k][j];
```

- **优点:**

① 增强数据局部性

② 增强向量化和并行化的识别

循环交换

- **循环交换的有利性**

```
for(i=1;i<N;i++)
    for(j=1;j<N;j++)
        A[i][j+1] = A[i][j] + 2;
```

循环交换

```
for(j=1;j<N;j++)
    for(i=1;i<N;i++)
        A[i][j+1] = A[i][j] + 2;
```

```
for (j = 1; j < N; j++)
    for (i = 1; i < M; i++)
        A[i][j] = A[i - 1][j];
```

循环交换

```
for (i = 1; j < M; j++)
    for (j = 1; i < N; i++)
        A[i][j] = A[i - 1][j];
```

i →

j ↓

A[1][1] = A[0][1]	A[2][1] = A[1][1]	A[3][1] = A[2][1]
A[1][2] = A	2][2] = A[1][2]	A[3][2] = A[2][2]
A[1][3] = A	2][3] = A[1][3]	A[3][3] = A[2][3]

j →

i ↓

A[1][1] = A[0][1]	A[1][2] =A[0][2]	A[1][3] = A[0][3]
A[2][1] = A[1][1]	A[2][2] = A[1][2]	A[2][3] = A[1]
A[3][1] = A[2][1]	A[3][2] = A[2][2]	A[3][3] = A[2]

循环交换

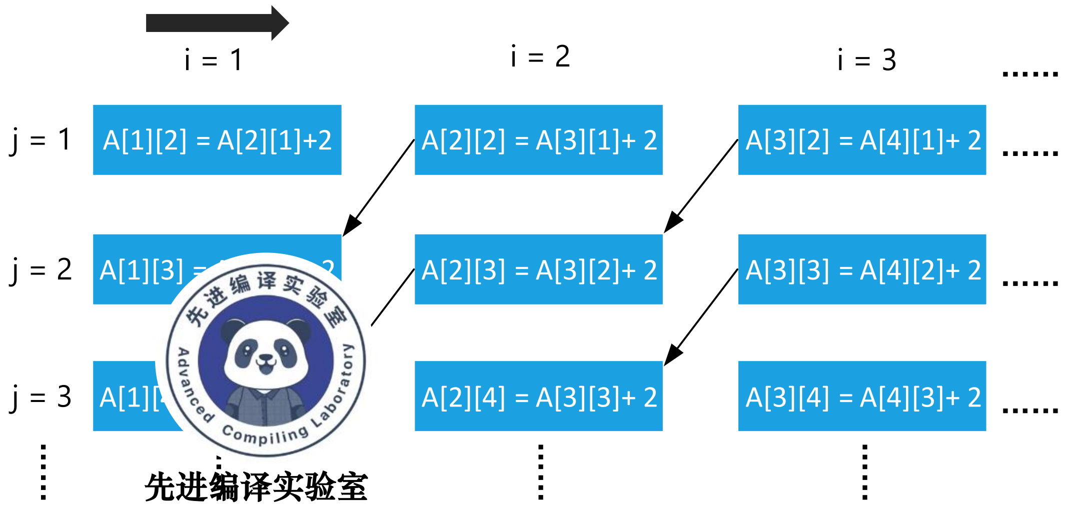
- **循环交换的合法性**

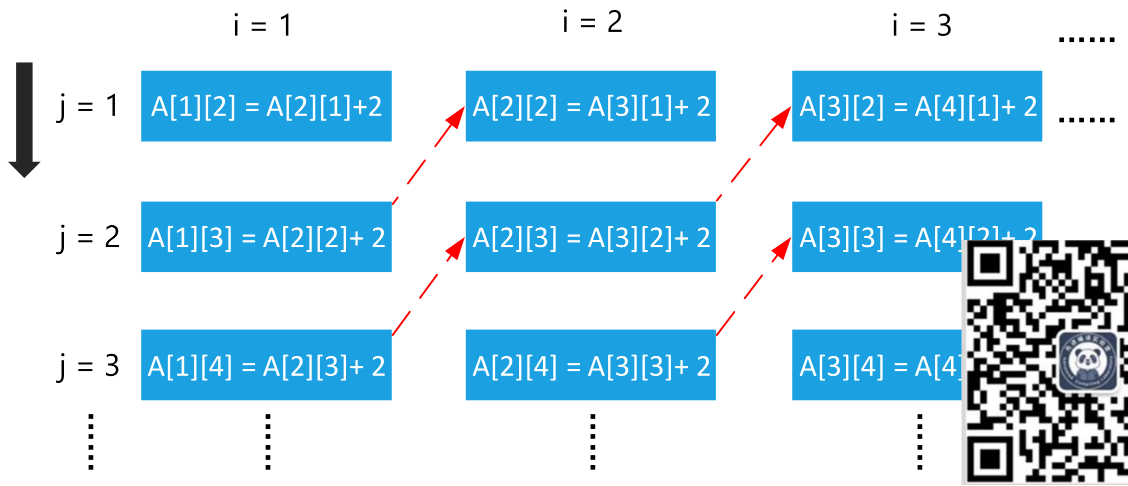
重排序某个依赖端点的循环交换是不合法的，即不要引起依赖关系的反转。

```
for(j=1;j<N;j++)
    for(i=1;i<N;i++)
        A[i][j+1] = A[i+1][j] + 2;
```

循环交换 →

```
for(i=1;i<N;i++)
    for(j=1;j<N;j++)
        A[i][j+1] = A[i+1][j] + 2;
```





循环交换

- **优化效果**

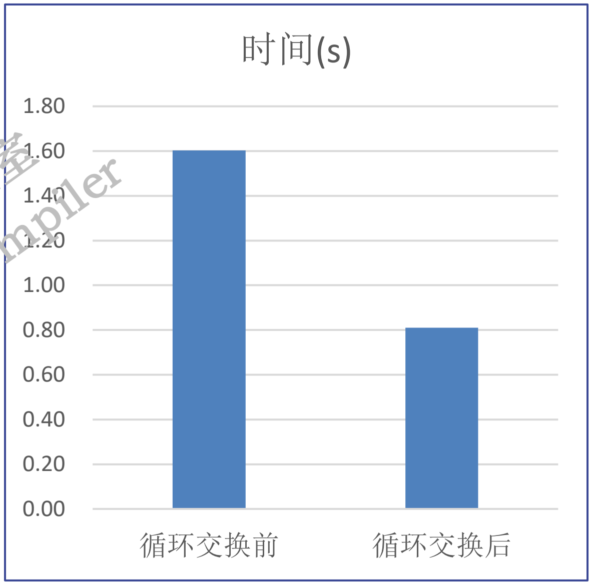
  测试环境：Hygon C86 7185 32-core Processor；x86_64；

  编译器版本：llvm-13

```
#include <stdio.h>
#include <time.h>
int main() {
  clock_t start,finish;
  double total_time;
  const int N = 512;
  double A[N][N], B[N][N], C[N][N];
  int i, j, k;
  for (i = 0; i < N; i++) {
    for (j = 0; j < N; j++) {
      A[i][j] = j;
      B[i][j] = j;
      C[i][j] = i;
    }
  }
  start = clock();
  for (j = 0; j < N; j++)
    for (k = 0; k < N; k++)
      for (i = 0; i < N; i++)
        A[i][j] = A[i][j              C[k][j];
  finish = clock();
  total_time = (dou              /CLOCKS_PER_SEC;
  printf("used time              total_time);
  return A[4][5];
}
```

循环交换

```
#include <stdio.h>
#include <time.h>
int main() {
  clock_t start,finish;
  double total_time;
  const int N = 512;
  double A[N][N], B[N][N], C[N][N];
  int i, j, k;
  for (i = 0; i < N; i++) {
    for (j = 0; j < N; j++) {
      A[i][j] = j;
      B[i][j] = j;
      C[i][j] = i;
    }
  }
  start = clock();
  for (j = 0; j < N; j++)
    for (i = 0; i < N; i++)
      for (k = 0; k < N; k++)
        A[i][j] = A[i][j] + B[i][k] * C[k][j];
  finish = clock();
  total_time = (double)(finish-start)/CLOCKS_PER_SEC;
  printf("used time %f seconds\n",total_time);
  return A[4][5];
}
```



循环交换

- **优化效果**

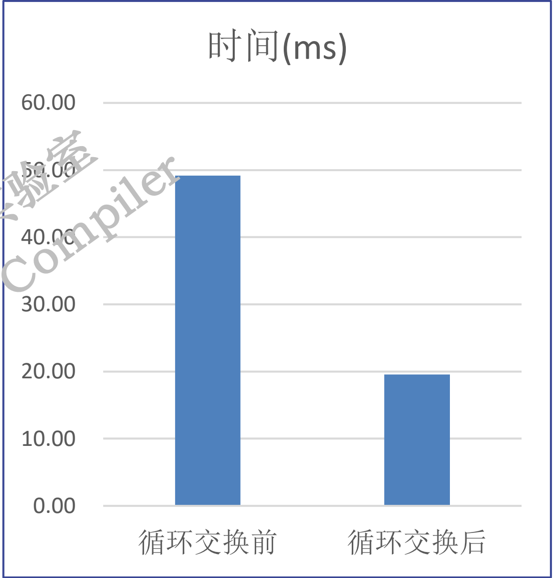
测试环境：Hygon C86 7185 32-core Processor；x86_64；

编译器版本：llvm-13

```
#include <stdio.h>
#include <sys/time.h>
int main() {
  struct timeval  time_start, time_end;
  const int N = 2048;
  double A[N][N], B[N][N], C[N][N];
  int i, j, k;
  for (i = 0; i < N; i++) {
    for (j = 0; j < N; j++) {
      A[i][j] = j;
    }
  }
  gettimeofday(&time_start, NULL);
  for (j = 1; j < N; j++){
    for (i = 1; i < N; i++){
      A[i][j] = A[i-1][j];
    }
  }
  gettimeofday(&time_end, NULL);
  printf("used tir                  " time_end.tv_usec -
time_start.tv_
  return A[4][5
}
```

循环交换 →

```
#include <stdio.h>
#include <sys/time.h>
int main() {
  struct timeval  time_start, time_end;
  const int N = 2048;
  double A[N][N], B[N][N], C[N][N];
  int i, j, k;
  for (i = 0; i < N; i++) {
    for (j = 0; j < N; j++) {
      A[i][j] = j;
    }
  }
  gettimeofday(&time_start, NULL);
  for (i = 1; i < N; i++){
    for (j = 1; j < N; j++){
      A[i][j] = A[i-1][j];
    }
  }
  gettimeofday(&time_end, NULL);
  printf("used time %ld us\n",time_end.tv_usec -
time_start.tv_usec);
  return A[4][5];
}
```



循环交换

- **编译器中的循环交换**

```
#include <stdio.h>
int main() {
  const int M = 200;
  const int N= 300;
  int i, j;
  int A[M][N];
  for (i = 0; i < M; i++){
    for (j = 0; j < N; j++){
        A[i][j] = j;
    }
  }
  for (j = 0; j < N; j++){
    for (i = 0; i < M; i++){
      A[i][j] = A[i][j] + 2;
    }
  }
  return A[3][4];
}
```

```
[llvm@2022] clang test.c -O1  -mllvm -enable-loopinterchange -Rpass=loop-interchange -Rpass-missed=loop-interchange -Rpass-analysis=loop-interchange
test.c:8:5: remark: Interchanging loops is too costly (cost=1, threshold=0) and it does not improve parallelism. [-Rpass-missed=loop-interchange]
    for (j = 0; j < N; j++){
    ^
test.c:13:5: remark: Loop interchanged with enclosing loop. [-Rpass=loop-interchange]
    for (i = 0; i < M; i++){
    ^
```

编译器	选项
LLVM	-mllvm -enable-loopinterchange
GCC	-floop-interchange

循环交换

- **编译器中的循环交换**

```
for.cond12.preheader:                          ; preds = %for.inc6, %for.inc26
  %indvars.iv62 = phi i64 [ %indvars.iv.next63, %for.inc26 ], [ 0, %for.inc6 ]     j
  br label %for.body14, !dbg !28

for.body14:                                    ; preds = %for.cond12.preheader, %for.body14
  %indvars.iv58 = phi i64 [ 0, %for.cond12.preheader ], [ %indvars.iv.next59,
%for.body14 ]                                                                      i
  %2 = mul nuw nsw i64 %indvars.iv58, 300, !dbg !29
  %arrayidx16 = getelementptr inbounds [60000 x i32], [60000 x i32]* %vla48, i64 0, i64
%2, !dbg !29
  %arrayidx18 = getelementptr inbounds i32, i32* %arrayidx16, i64
%indvars.iv62, !dbg !29
  %3 = load i32, i32* %arrayidx18, align 4, !dbg !29, !tbaa !14
  %add = add nsw i32 %3, 2, !dbg !30
  store i32 %add, i32* %arrayidx18, align 4, !dbg !31, !tbaa !14
  %indvars.iv.next59 = add nuw nsw i64 %indvars.iv58, 1, !dbg !32
  %exitcond61.not = icmp eq i64 %indvars.iv.next59, 200, !dbg !33
  br i1 %exitcond61.not, label %for.inc26, label %for.body14, !dbg !28, !llvm.loop !34

for.inc26:                                     ; preds = %for.body14
  %indvars.iv.next63 = add nuw nsw i64 %indvars.iv62, 1, !dbg !36
  %exitcond65.not = icmp eq i64 %indvars.iv.next63, 300, !dbg !37
  br i1 %exitcond65.r                 r.end28, label
%for.cond12.prehe                  lvm.loop !39
```

```
for.body14:                                    ; preds = %for.inc6, %for.body14.split
  %indvars.iv58 = phi i64 [ %4, %for.body14.split ], [ 0, %for.inc6 ]              i
  %3 = mul nuw nsw i64 %indvars.iv58, 300
  %arrayidx16 = getelementptr inbounds [60000 x i32], [60000 x i32]* %vla48, i64 0, i64
%3
  br label %for.cond12.preheader, !dbg !33

for.cond12.preheader:                          ; preds = %for.body14, %for.cond12.preheader
  %indvars.iv62 = phi i64 [ 0, %for.body14 ], [ %indvars.iv.next63, %for.cond12.preheader ]
  %arrayidx18 = getelementptr inbounds i32, i32* %arrayidx16, i64 %indvars.iv62, !dbg !28
  %2 = load i32, i32* %arrayidx18, align 4, !dbg !28, !tbaa !14                     j
  %add = add nsw i32 %2, 2, !dbg !29
  store i32 %add, i32* %arrayidx18, align 4, !dbg !30, !tbaa !14
  %indvars.iv.next63 = add nuw nsw i64 %indvars.iv62, 1, !dbg !31
  %exitcond65.not = icmp eq i64 %indvars.iv.next63, 300, !dbg !32
  br i1 %exitcond65.not, label %for.body14.split, label
%for.cond12.preheader, !dbg !33, !llvm.loop !34

for.body14.split:                              ; preds = %for.cond12.preheader
  %4 = add nuw nsw i64 %indvars.iv58, 1, !dbg !36
  %.not = icmp eq i64 %4, 200, !dbg !37
  br i1 %.not, label %for.end28, label %for.body14, !dbg !38, !llvm.loop !39
```

分享完毕，感谢聆听！

参考文献：

[1] Optimizing Compilers for Modern Architectures: A Dependence-Based Approach [Book Review][J]. Computer,2002,35(4).

[2]