先进编译实验室
Advanced Compiler

循环优化系列第三讲

循环分布

嘉宾：柴赟达

先进编译实验室
Advanced Compiler

循环分布

- **基础概念**

循环分布，Loop Distribute，将一个循环分解为多个循环，且每个循环都有与原循环相同的迭代空间，但只包含原循环的语句子集，常用于分解出可向量化或者可并行化的循环，进而将可向量化部分的代码转为向量执行。

```
for (int i = 0；i < n；i++) {
   A[i] = i;
   B[i] = 2 + B[i];
   C[i] = 3 + C[i - 1];
}
```

```
for (int i = 0；i < n；i++){
   A[i] = i;
   B[i] = 2 + B[i];
}
for (int i = 0；i < n；i++){
   C[i] = 3 + C[i - 1];
}
```


- **优点：**

① 将一个串行循环转变为多个并行循环

② 实现循环的部分并行化

③ 增加循环优化的范围

- **缺点：**

① 减小并行

② 增加额外的 开销

循环分布

- **循环分布的效果**

```
for(i=1;i<N;i++){
  for(j=1;j<N;j++){
    A[i][j] = B[i][j] + C[i][j];//S1
    D[i][j] = A[i][j-1] * 2;//S2
  }
}
```

循环分布

```
for(i=1;i<N;i++){
  for(j=1;j<N;j++){
    A[i][j] = B[i][j] + C[i][j];//S1
for(i=1;i<N;i++)
  for(j=1;j<N;j++)
    D[i][j] = A[i][j-1] * 2;//S2
```

循环分布

- **和循环交换优化的配合**

若循环不是紧嵌套循环导致无法进行后续优化操作时，可以使用循环分布将循环体变换为紧嵌套循环。

```
for (i = 1; i < N; i++) {
    for (j = 1; j < N; j++) {
        A[i][j] = D;//S1语句
        for (k = 1; k < N; k++) {
            A[i][j] = A[i][j] + B[i][k] * C[k][j];//S2语句
        }
    }
}
```

循环分布

```
for (i = 1; i < N; i++)
    for (j = 1; j < N; j++)
        A[i][j] = D;//S1语句

for (i = 1; i < N; i++)
    for (j = 1; j < N; j++)
        for (k = 1; k < N; k++)
            A[i][j] = A[i][j] + B[i][k] * C[k][j];//S2语句
```

循环分布

- 与循环合并优化的配合

```
for(i=1;i<N;i++){
  A[i] =  B[i] + 1;//S1
  C[i] = A[i] + C[i-1];//S2
  D[i] = A[i] + x;//S3
}
```

循环分布

```
for(i=1;i<N;i++)
  A[i] =  B[i] + 1;//S1
for(i=1;i<N;i++)
  C[i] = A[i] + C[i-1];//S2
for(i=1;i<N;i++)
  D[i] = A[i] + x; //S3
```

循环合并

```
for(i=1;i<N;i++){
  A[i] =  B[i] + 1;//S1
  D[i] = A[i] + x;//S3
}
for(i=1;i<N;i++)
  C[i] = A[i] + C[i-1];//S2
```

循环分布	循环合并
增加循环判断条件执行	减少循环判断条件执行
将一个循环拆分为多个循环	将多个循环拆合并为一个循环

循环分布

- **编译器中的循环分布**

```
#include <stdio.h>
#include <stdlib.h>
#define N 1280
int main(){
        int A[N],B[N],C[N];
        int i;
        for(i=0;i<N;i++){
            B[i] = rand();
            C[i] = rand();
        }
        for(i=1;i<N;i++){
             A[i] = i;
             B[i] = 2 + B[i];
             C[i] = 3 + C[i - 1];
        }
        for(i=0;i<N;i++){
              printf("%d",B[i]);
              printf("%d",A[i]);
              printf("%d",C[i]);
        }
}
```

```
[llvm@2021]$ clang -O1 -mllvm -enable-loop-distribute loop.c -Rpass=loop-distribute    -Rpass-missed=loop-distribute   -Rpass-analysis=loop-distribute -emit-llvm -S
loop.c:16:9: remark: loop not distributed: use -Rpass-analysis=loop-distribute for more info [-Rpass-missed=loop-distribute]
        for(i=0;i<N;i++){
        ^
loop.c:16:9: remark: loop not distributed: no unsafe dependences to isolate [-Rpass-analysis=loop-distribute]
loop.c:11:9: remark: distributed loop [-Rpass=loop-distribute]
        for(i=1;i<N;i++){
        ^
loop.c:7:9: remark: loop not distributed: use -Rpass-analysis=loop-distribute for more info [-Rpass-missed=loop-distribute]
        for(i=0;i<N;i++){
        ^
loop.c:7:9: remark: loop not distributed: no unsafe dependences to isolate [-Rpass-analysis=loop-distribute]
```

编译器	选项
LLVM	-mllvm -enable-loop-distribute
GCC	-ftree-loop-distribution

循环分布

- **编译器中的循环分布**

```
for.body6:                                        ; preds = %for.body6.preheader, %for.body6
  %store_forwarded = phi i32 [ %load_initial, %for.body6.preheader ], [ %add15,
%for.body6 ]
  %indvars.iv56 = phi i64 [ 1, %for.body6.preheader ], [ %indvars.iv.next57, %for.body6 ]
  %arrayidx8 = getelementptr inbounds [1280 x i32], [1280 x i32]* %A, i64 0, i64
%indvars.iv56, !dbg !27
  %3 = trunc i64 %indvars.iv56 to i32, !dbg !28
  store i32 %3, i32* %arrayidx8, align 4, !dbg !28, !tbaa !13
  %arrayidx10 = getelementptr inbounds [1280 x i32], [1280 x i32]* %B, i64 0, i64
%indvars.iv56, !dbg !29
  %4 = load i32, i32* %arrayidx10, align 4, !dbg !29, !tbaa !13
  %add = add nsw i32 %4, 2, !dbg !30
  store i32 %add, i32* %arrayidx10, align 4, !dbg !31, !tbaa !13
  %add15 = add nsw i32 %store_forwarded, 3, !dbg !32
  %arrayidx17 = getelementptr inbounds [1280 x i32], [1280 x i32]* %C, i64 0, i64
%indvars.iv56, !dbg !33
  store i32 %add15, i32* %arrayidx17, align 4, !dbg !34, !tbaa !13
  %indvars.iv.next57 = add nuw nsw i64 %indvars.iv56, 1, !dbg !35
  %exitcond59.not = icmp eq i64 %indvars.iv.next57, 1280, !dbg !36
  br i1 %exitcond59.not, label %for.body23, label %for.body6, !dbg !26, !llvm.loop !37
```

先进编译实验室
Advanced Compiler

```
for.body6.ldist1:                                 ; preds = %for.body, %for.body6.ldist1
  %indvars.iv56.ldist1 = phi i64 [ %indvars.iv.next57.ldist1, %for.body6.ldist1 ], [ 1, %for.body ]
  %arrayidx8.ldist1 = getelementptr inbounds [1280 x i32], [1280 x i32]* %A, i64 0, i64
%indvars.iv56.ldist1, !dbg !26
  %3 = trunc i64 %indvars.iv56.ldist1 to i32, !dbg !27
  store i32 %3, i32* %arrayidx8.ldist1, align 4, !dbg !27, !tbaa !13
  %arrayidx10.ldist1 = getelementptr inbounds [1280 x i32], [1280 x i32]* %B, i64 0, i64
%indvars.iv56.ldist1, !dbg !28
  %4 = load i32, i32* %arrayidx10.ldist1, align 4, !dbg !28, !tbaa !13
  %add.ldist1 = add nsw i32 %4, 2, !dbg !29
  store i32 %add.ldist1, i32* %arrayidx10.ldist1, align 4, !dbg !30, !tbaa !13
  %indvars.iv.next57.ldist1 = add nuw nsw i64 %indvars.iv56.ldist1, 1, !dbg !31
  %exitcond59.not.ldist1 = icmp eq i64 %indvars.iv.next57.ldist1, 1280, !dbg !32
  br i1 %exitcond59.not.ldist1, label %for.body6.preheader, label
%for.body6.ldist1, !dbg !33, !llvm.loop !34

for.body6.preheader:                              ; preds = %for.body6.ldist1
  %load_initial = load i32, i32* %C63, align 16
  br label %for.body6, !dbg !33

for.body6:                                        ; preds = %for.body6.preheader, %for.body6
  %store_forwarded = phi i32 [ %load_initial, %for.body6.preheader ], [ %add15, %for.body6 ]
  %indvars.iv56 = phi i64 [ 1, %for.body6.preheader ], [ %indvars.iv.next57, %for.body6 ]
  %add15 = add nsw i32 %store_forwarded, 3, !dbg !36
  %arrayidx17 = getelementptr inbounds [1280 x i32], [1280 x i32]* %C, i64 0
%indvars.iv56, !dbg !37
  store i32 %add15, i32* %arrayidx17, align 4, !dbg !38, !tbaa !13
  %indvars.iv.next57 = add nuw nsw i64 %indvars.iv56, 1, !dbg !31
  %exitcond59.not = icmp eq i64 %indvars.iv.next57, 1280, !dbg !32
  br i1 %exitcond59.not, label %for.body23, label %for.body6, !dbg !33, !llvm.l
```

循环分布

- **通过pragma码进行循环分布**

如果优化人员想指定某一个循环实现循环分布，则可以通过编译指示语句指定循环#pragma clang loop distribute(enable)开启循环分布优化。

```
#include <stdio.h>
int main( ){
    int i,N;
    N=1024;
    int A[N],B[N],C[N],D[N],E[N];
    for (i = 0; i < N; ++i) {
        A[i] = i;
        B[i] = i + 1;
        D[i] = i + 2;
        E[i] = i + 3;
   }
    #pragma clang loop distribute(enable)
    for (i = 0; i < N; ++i) {
        A[i + 1] = A[i] + B[i];//S1
        C[i] = D[i] * E[i];//S2
     }
     return A[8];
}
```

分享完毕，感谢聆听！

参考文献：

[1] 韩林,徐金龙,李颖颖,王阳.面向部分向量化的循环分布及聚合优化[J].计算机科学,2017,44(02):70-74+81.

[2] 陈梦尧. 面向申威GCC编译系统的循环分布技术研究[D]. 郑州大学,2021.DOI:10.27466/d.cnki.gzzdu.2021.001990.

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