先进编译实验室
Advanced Compiler

循环优化系列第一讲

循环展开和压紧

嘉宾：柴赟达

先进编译实验室
Advanced Compiler

循环优化—循环展开和压紧

基础概念

- 循环展开：英文称loop unrolling，是一种牺牲程序的尺寸来加快程序的执行速度的优化方法，可以由程序员完成，也可由编译器自动优化完成。它通过将循环体内的代码复制多次的操作，进而减少循环分支指令执行的次数，增大处理器指令调度的空间，获得更多的指令级并行。
- 循环压紧：是指调整复制后的语句执行，将原来一条语句复制得到的多条语句合并到一起。
- 优点：
  ① 减少循环分支指令执行的次数；
  ② 获得了更多的指令级并行；
  ③ 增加了寄存器的重用。


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

对j层进行循环展开

```
for (i = 0; i < N; i++) {
    for (j = 0; j < N; j += 4) {
        A[i][j] = A[i][j] + B[i][j] * C[i][j];
        A[i][j + 1] = A[i][j + 1] + B[i][j + 1] * C[i][j + 1];
        A[i][j + 2] = A[i][j + 2] + B[i][j + 2] * C[i][j + 2];
        A[i][j + 3] = A[i][j + 3] + B[i][j + 3] * C[i][j + 3];
    }
}
```

循环优化—循环展开和压紧

循环展开的合法性

- 循环展开需要在不影响原程序语义的情况下进行。

```
for (i = 1; i < N; i++) {
    for (j = 1; j < N; j++) {
        A[i+1][j-1] = A[i][j] +3;
    }
}
```

对i层进行循环展开

```
for (i = 1; i < N; i+=2) {
    for (j = 1; j < N; j++) {
        A[i+1][j-1] = A[i][j] +3;
        A[i+2][j-1] = A[i+1][j] +3;
    }
}
```

针对数组A的依赖分析:

	j=1	j=2	j=3	j=4	…
i=1	A[2][0]=A[1][1]	A[2][1]=A[1][2]	A[2][2]=A[1][3]	A[2][3]=A[1][4]	…
i=2	A[3][0]=A[2][1]	A[3][1]=A[2][2]	A[3][2]=A[2][3]	A[3][3] =A[2][4]	…
i=3	A[4][0]=A[3][1]	A[4][1]=A[3][2]	A[4][2]=A[3][3]	A[4][3]=A[3][4]	…
i=4	A[5][0]=A[4][1]	…=A[4][2]	A[5][2]=A[4][3]	A[5][3]=A[4][4]	…
…	…	…	…	…	

	j=1	j=2	j=3	j=4
i=1	A[2][0]=A[1][1] A[3][0]=A[2][1]	A[2][1]=A[1][2] A[3][1]=A[2][2]	A[2][2]=A[1][3] A[3][2]=A[2][3]	A[2][3]=A[1][4] A[3][3] =A[2][4]
i=3	A[4][0]=A[3][1] A[5][0]=A[4][1]	A[4][1]=A[3][2] A[5][1]=A[4][2]	A[4][2]=A[3][3] A[5][2]=A[4][3]	A[4][3]=A[3][4] A[5][3]=A[4][4]

循环优化—循环展开和压紧

- 循环完全展开：按照迭代次数对循环进行展开，该方法对循环嵌套的最内层循环或者向量化后的循环加速效果更明显。要注意的是并不是循环展开的次数越多获得的程序性能越高，过度地进行循环展开可能会增加寄存器的压力，可能会引起指令缓存区的溢出 。

```
void loop_unroll1(void){
    float a[8];
    for (int i = 0; i < 8; i++)
        a[i] = a[i] + 3;
}
```

```
void loop_unroll1(void){
    float a[8];
    for (int i = 0; i < 8; i+=8)
        a[i] = a[i] + 3;
        a[i+1] = a[i+1] + 3;
        a[i+2] = a[i+2] + 3;
        a[i+3] = a[i+3] + 3;
        a[i+4] = a[i+4] + 3;
        a[i+5] = a[i+5] + 3;
        a[i+6] = a[i+6] + 3;
        a[i+7] = a[i+7] + 3;
}
```

循环优化—循环展开和压紧

循环展开

- 当迭代次数不能被循环展开次数整除的情况下，需要在进行循环展开的同时考虑尾循环的处理。

```
for (i = 0; i < 512; i++) {
  for (j = 0; j < 510; j+=3) {
    A[i][j] = A[i][j] + B[i][j] * C[i][j];
    A[i][j+1] = A[i][j+1] + B[i][j+1] * C[i][j+1];
    A[i][j+2] = A[i][j+2] + B[i][j+2] * C[i][j+2];
  }
  for (j = 510; j < 512; j++) {
    A[i][j] = A[i][j] + B[i][j] * C[i][j];
  }
}
```

```
for (i = 0; i < 512; i++) {
  for (j = 0; j < 504; j+=9) {
    A[i][j] = A[i][j] + B[i][j] * C[i][j];
    A[i][j+1] = A[i][j+1] + B[i][j+1] * C[i][j+1];
    A[i][j+2] = A[i][j+2] + B[i][j+2] * C[i][j+2];
    A[i][j+3] = A[i][j+3] + B[i][j+3] * C[i][j+3];
    A[i][j+4] = A[i][j+4] + B[i][j+4] * C[i][j+4];
    A[i][j+5] = A[i][j+5] + B[i][j+5] * C[i][j+5];
    A[i][j+6] = A[i][j+6] + B[i][j+6] * C[i][j+6];
    A[i][j+7] = A[i][j+7] + B[i][j+7] * C[i][j+7];
    A[i][j+8] = A[i][j+8] + B[i][j+8] * C[i][j+8];
  }
  for (j = 504; j < 512; j++) {
    A[i][j] = A[i][j] + B[i][j] * C[i][j];
  }
}
```

循环优化—循环展开和压紧

循环展开效果

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

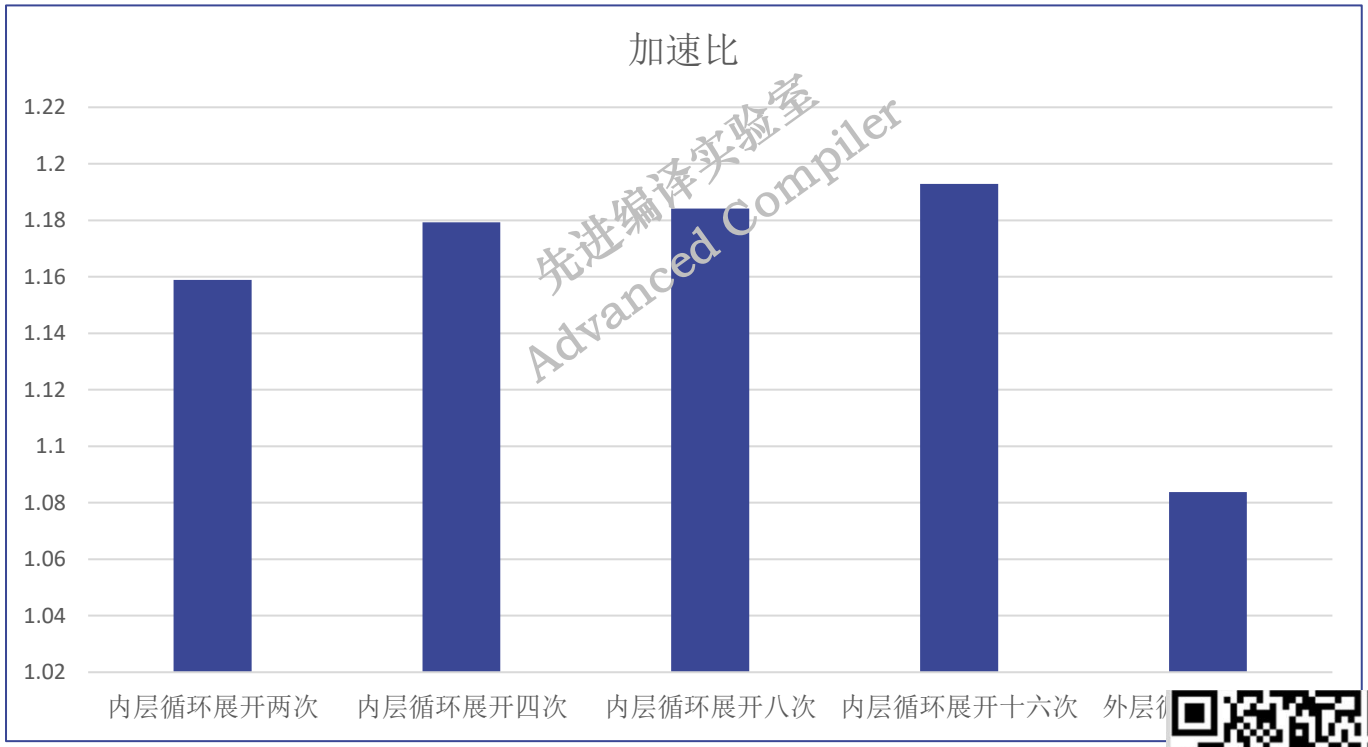
编译器版本：llvm-13

数据规模：512 * 512

基准测试例：

```
#include <stdio.h>
#include <sys/time.h>
int main() {
  const int N = 512;
  double A[N][N], B[N][N], C[N][N];
  int i, j;
  struct timeval  time_start, time_end;
  for (i = 0; i < N; i++) {
    for (j = 0; j < N; j++) {
      A[i][j] = j;
      B[i][j] = j;
      C[i][j] = j;
    }
  }
   gettimeofday(&time_start, NULL);
  for (i = 0; i < N; i++) {
    for (j = 0; j < N; j++) {
      A[i][j] + B[i][j] * C[i][j];

    }
  }
  (&time_end, NULL);
  p  used time %ld us\n",
time_e  usec - time_start.tv_usec);
}
```



循环优化—循环展开和压紧

编译器中的循环展开

```
#include <stdio.h>
#define N 128
int main(){
        float sum = 0;
        float a[N];
        int i,j;
        for(i=0;i<N;i++){
                a[i] = i;
        }
        for(j=0;j<N;j++) {
                sum = sum + a[j];
        }
        printf("sum = %f",sum);
}
```

clang unroll.c -O1 -funroll-loops -emit-llvm -S -Rpass=loop-unroll

```
unroll.c:10:9: remark: unrolled loop by a factor of 8 with a breakout at trip 0 [-Rpass=loop-unroll]
        for(i=0;i<N;i++){
unroll.c:7:9: remark: unrolled loop by a factor of 4 with a breakout at trip 0 [-Rpass=loop-unroll]
    for(j=0;j<N;j++) {
```

```
%indvars.iv = phi i64 [ %indvars.iv.next.7, %for.body4 ], [ 0, %for.body ]
 %sum.020 = phi float [ %add.7, %for.body4 ], [ 0.000000e+00, %for.body ]
 %arrayidx6 = getelementptr inbounds [128 x float], [128 x float]* %a, i64 0, i64 %indvars.iv
 %5 = load float, float* %arrayidx6, align 16, !tbaa !2
 %add = fadd float %sum.020, %5
 %indvars.iv.next = or i64 %indvars.iv, 1
 %arrayidx6.1 = getelementptr inbounds [128 x float], [128 x float]* %a, i64 0, i64 %indvars.iv.next
 %6 = load float, float* %arrayidx6.1, align 4, !tbaa !2
 %add.1 = fadd float %add, %6
 %indvars.iv.next.1 = or i64 %indvars.iv, 2
 %arrayidx6.2 = getelementptr inbounds [128 x float], [128 x float]* %a, i64 0, i64 %indvars.iv.next.1
 %7 = load float, float* %arrayidx6.2, align 8, !tbaa !2
 %add.2 = fadd float %add.1, %7
 %indvars.iv.next.2 = or i64 %indvars.iv, 3
 %arrayidx6.3 = getelementptr inbounds [128 x float], [128 x float]* %a, i64 0, i64 %indvars.iv.next.2
 %8 = load float, float* %arrayidx6.3, align 4, !tbaa !2
 %add.3 = fadd float %add.2, %8
 %indvars.iv.next.3 = or i64 %indvars.iv, 4
 %arrayidx6.4 = getelementptr inbounds [128 x float], [128 x float]* %a, i64 0, i64 %indvars.iv.next.3
 %9 = load float, float* %arrayidx6.4, align 16, !tbaa !2
 %add.4 = fadd float %add.3, %9
 %indvars.iv.next.4 = or i64 %indvars.iv, 5
 %arrayidx6.5 = getelementptr inbounds [128 x float], [128 x float]* %a, i64 0, i64 %indvars.iv.next.4
 %10 = load float, float* %arrayidx6.5, align 4, !tbaa !2
 %add.5 = fadd float %add.4, %10
 %indvars.iv.next.5 = or i64 %indvars.iv, 6
 %arrayidx6.6 = getelementptr inbounds [128 x float], [128 x float]* %a, i64 0, i64 %indva
 %11 = load float, float* %arrayidx6.6, align 8, !tbaa !2
 %add.6 = fadd float %add.5, %11
 %indvars.iv.next.6 = or i64 %indvars.iv, 7
……
```


先进编译实验室
Advanced Compiler

编译器中的循环展开

与循环展开有关的编译选项

选项	功能
-funroll-loops	打开循环展开
-fno-unroll-loops	关闭循环展开
-mllvm -unroll-max-count	为部分和运行时展开设置最大展开计数
-mllvm -unroll-count	确定展开次数
-mllvm -unroll-runtime	使用运行时行程计数展开循环
-mllvm -unroll-threshold	设定循环展开的成本限值
-mllvm -unroll-remainder	允许循环展开后有尾循环
-Rpass=loop-unroll	显示循环展开的优化信息
-Rpass-missed=loop-unroll	显示循环展开失败的信息

循环优化—循环展开和压紧

通过pragma指令实现循环展开

```
#include <stdio.h>
#include <stdlib.h>
int main( ){
  int i,N,sum;
  N=1024;
  sum = 0;
  int A[N],B[N];
  for (i = 0; i < N; ++i) {
    A[i] = i;
    B[i] = rand()%10;
  }
#pragma clang loop unroll(enable)
  for (i = 0; i < N; i++) {
    sum= sum + A[i] + B[i];
  }
  printf("%d\n",s
}
```

```
#include <stdio.h>
#include <stdlib.h>
int main( ){
  int i,N,sum;
  N=1024;
  sum = 0;
  int A[N],B[N];
  for (i = 0; i < N; ++i) {
    A[i] = i;
    B[i] = rand()%10;
  }
#pragma clang loop unroll(full)
  for (i = 0; i < N; i++) {
    sum= sum + A[i] + B[i];
  }
  printf("%d\n",sum);
}
```

```
#include <stdio.h>
#include <stdlib.h>
int main( ){
  int i,N,sum;
  N=1024;
  sum = 0;
  int A[N],B[N];
  for (i = 0; i < N; ++i) {
    A[i] = i;
    B[i] = rand()%10;
  }
#pragma clang loop unroll_count(8)
  for (i = 0; i < N; i++) {
    sum= sum + A[i] + B[i];
  }
  printf("%d\n",sum);
}
```

循环优化—循环展开和压紧

针对向量程序的循环展开

可以针对向量化指令的程序进行循环展开，以进一步挖掘数据级的并行性，提高程序执行的性能。

```
#include <stdio.h>
int main() {
    const int N = 256;
    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] = 1.0;
            B[i][j] = 2.0;
            C[i][j] = 3.0;
        }
    }
    for (i = 0; i < N; i++) {
        for (j = 0; j < N; j++) {
            A[i][j] = A[i][j] + B[i][j] * C[i][j];
        }
    }
    for (i = 0;
        for (j = 0,
            printf("%
}
```

```
#include <stdio.h>
#include <x86intrin.h>
int main(){
  __m256 ymm0,ymm1,ymm2,ymm3,ymm4,ymm5;
  int N = 256;
  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] = 1.0;
      B[i][j] = 2.0;
      C[i][j] = 3.0;
    }
  }
  for(i=0;i<N;i++){
    for(j=0;j<N;j+=4){
      ymm3 = _mm256_loadu_pd(A[i]+j);
      ymm4 = _mm256_loadu_pd(B[i]+j);
      ymm5 = _mm256_loadu_pd(C[i]+j);
      ymm4 = _mm256_mul_pd(ymm4,ymm5);
      ymm3 = _mm256_add_pd(ymm3,ymm4);
      _mm256_storeu_pd(A[i]+j,ymm3);
    }
  }
}
```

```
for (i = 0; i < N; i++) {
    for (j = 0; j < N; j+=8) {
      ymm3 = _mm256_loadu_pd(A[i] + j);
      ymm4 = _mm256_loadu_pd(B[i] + j);
      ymm5 = _mm256_loadu_pd(C[i] + j);
      ymm4 = _mm256_mul_pd(ymm4, ymm5);
      ymm6 = _mm256_add_pd(ymm3, ymm4);
      _mm256_storeu_pd(A[i] + j, ymm6);
      ymm3 = _mm256_loadu_pd(A[i] + j +4);
      ymm4 = _mm256_loadu_pd(B[i] + j
      ymm5 = _mm256_loadu_pd(C[i] + j
      ymm4 = _mm256_mul_pd(ymm4, ym
      ymm6 = _mm256_add_pd(ymm3, ym
      _mm256_storeu_pd(A[i] + j +4, ymm
    }
  }
```

分享完毕，感谢聆听！

先进编译实验室
Advanced Compiler

参考文献：

[1] 高伟,赵荣彩,于海宁,张庆花.循环展开技术在向量程序中的应用[J].计算机科学,2016,43(01):226-231+245.

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

先进编译实验室
Advanced Compiler