智能软件与工程学院
School of Intelligent Software and Engineering

代码优化
Code Optimization

介绍

钮鑫涛

资料

- 本讨论班的课件大多移植于CMU的课程15-745:
  - Optimizing Compilers for Modern Architectures
- 一些课本:
  - Compilers: Principles, Techniques, & Tools （龙书）
  - Advanced Backend Code Optimization
  - Engineering a compiler
  - Optimizing compilers for modern architectures a dependence based approach
  - Programming Language Pragmatics

程序优化的重要性

- 程序优化是编译技术的非常重要的步骤
  - 优化技术是目前编译方向最为主流的研究方向
  - 很多open problems有待解决
- 优化技术在工程领域非常重要
  - 高质量的代码可以为企业带来巨大效益！
    - 目前能够脱颖而出的大模型公司，其高效的底层工程实现才能实现盈利

编译器的工作

- 1. 将输入程序翻译为目标语言下的语义等价的程序
  - 比如将C翻译为汇编语言或者Java语言！
- 2. 优化代码
  - 性能更好的代码
    - size、速度、能源、或者鲁棒性

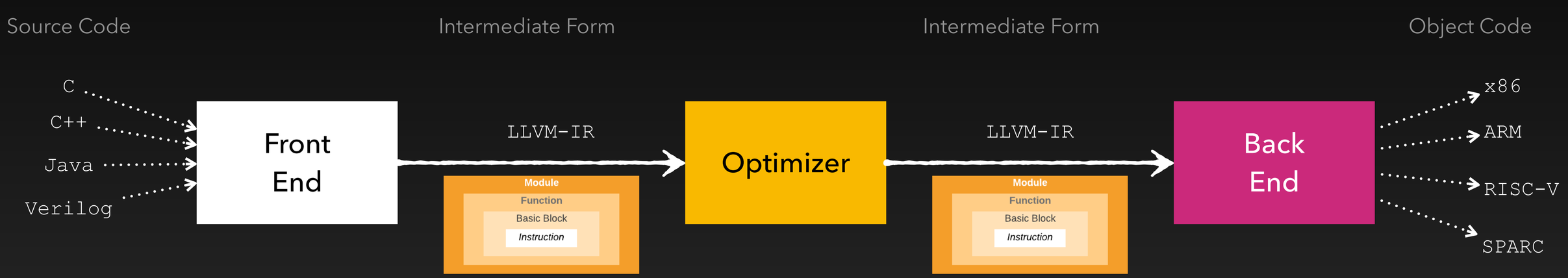
编译器怎么提升性能?

Execution time = Operation count * Machine cycles per operation

- 最小化操作的数量
  - 比如减少运算操作，内存访问
    - 深度学习编译器里有个优化操作就是“合并”算子，减少计算
- 将昂贵的操作替换为更简单的 (Strength reduction)
  - 比如将需要4个cycles的乘法替换为之需要一个cycle的位移
- 最小化cache的miss
  - 利用好locality
- 将工作并行化
  - e.g., Single Instruction Multiple Data (SIMD)
  - Loop unroll
  - 但处理器下的流水线的优化（重排指令）

编译器的结构

编译器结构



- 优化大部分是在中间层开展的
- 中间层的语言介于高级语言和底层语言之间
  - 相比于高级语言，有多特性方便分析和优化（比如SSA）
- 中间层的引入也让编译过程更加的模块化

编译器优化的成分

- 明确优化问题
  - 找到能够优化的机会
- 合适的表示
  - 做相关优化操作时代码的最合适的表示
    - 控制流程图（Control-flow graph）、Def/use chains、SSA(Static Single Assignment)…
- 分析
  - 分析出是否可以进行安全的转换、转换是否值得
    - Control-flow analysis, Dataflow analysis, Pointer analysis:
- 代码转换(Code Transformation)
- 评估

优化的种类

- Peephole
- Local
- Global
- Loop
- Interprocedural, whole-program or link-time
- Machine code
- Data-flow
- SSA-based
- Functional language
- …

优化的种类

- Bounds-checking elimination
- Dead code elimination
- Inline expansion or macro expansion
- Jump threading
- Macro compression
- Reduction of cache collisions
- Stack height reduction

表示的基础：三地址码

- 三地址码（Three-address code）：最多三个操作数
  - A := B op C
    - Left Hand Side (LHS): name of variables, e.g., x, A[t]
    - Right Hand Side (RHS) : value
  - 典型的指令：
    - A := B op C
    - A := unaryop B
    - A := B
    - GOTO s
    - IF A relop B GOTO s
    - CALL f
    - RETURN

优化的例子

优化的例子：冒泡排序

- 冒泡排序BubbleSort就是对一个数组A进行排序：
  - A的每个元素占据四个字节
  - A的元素的索引下标从1到n
  - A[j] 在 &A + 4*[j - 1]地址下存储

```
for (i = n - 1; i >= 1; i--)
{
    for (j = 1; j <= i; j++)
        if (A[j] > A[j + 1])
        {
            temp = A[j];
            A[j] = A[j + 1];
            A[j + 1] = temp;
        }
}
```

翻译后的中间码 (Pseudo)

Outer loop

Inner loop

```
    i := n-1
L5: if i<1 goto L1
    j := 1
L4: if j>i goto L2
    t1 := j-1
    t2 := 4*t1
    t3 := A[t2]     ;A[j]
    t4 := j+1
    t5 := t4-1
    t6 := 4*t5
    t7 := A[t6]     ;A[j+1]
    if t3<=t7 goto L3
    t8 := j-1
    t9 := 4*t8
    temp := A[t9]     ;temp:=A[j]
    t10 := j+1
    t11:= t10-1
    t12 := 4*t11
    t13 := A[t12]     ;A[j+1]
    t14 := j-1
    t15 := 4*t14
    A[t15] := t13     ;A[j]:=A[j+1]
    t16 := j+1
    t17 := t16-1
    t18 := 4*t17
    A[t18]:=temp      ;A[j+1]:=temp
L3: j := j+1
    goto L4
L2: i := i-1
    goto L5
L1:
```

表示：基本块（basic block）

- 基本块：连续的3地址码
  - 这个代码块只有第一句可以从外部抵达（中间的代码不可以被跳转到或者分支到达）
  - 只要第一句被执行，那么该块的其余语句都会被连续的执行（中间没有分支或者终止，除非是最后一句）
- 我们需要基本块最大化：
  - 除非破快基本块的性质，否则无法再变大
- 在基本块内做的优化即为本地优化（ **local** optimizations）

寻找基本块

基本块：第一句进入，最后一句出 Jump targets start a block, and jumps end a block)

```
    i := n-1                                  B1
L5: if i<1 goto L1                            B2
    j := 1                                    B3
L4: if j>i goto L2                            B4
    t1 := j-1
    t2 := 4*t1
    t3 := A[t2]     ;A[j]
    t4 := j+1
    t5 := t4-1                                B5
    t6 := 4*t5
    t7 := A[t6]     ;A[j+1]
    if t3<=t7 goto L3
```

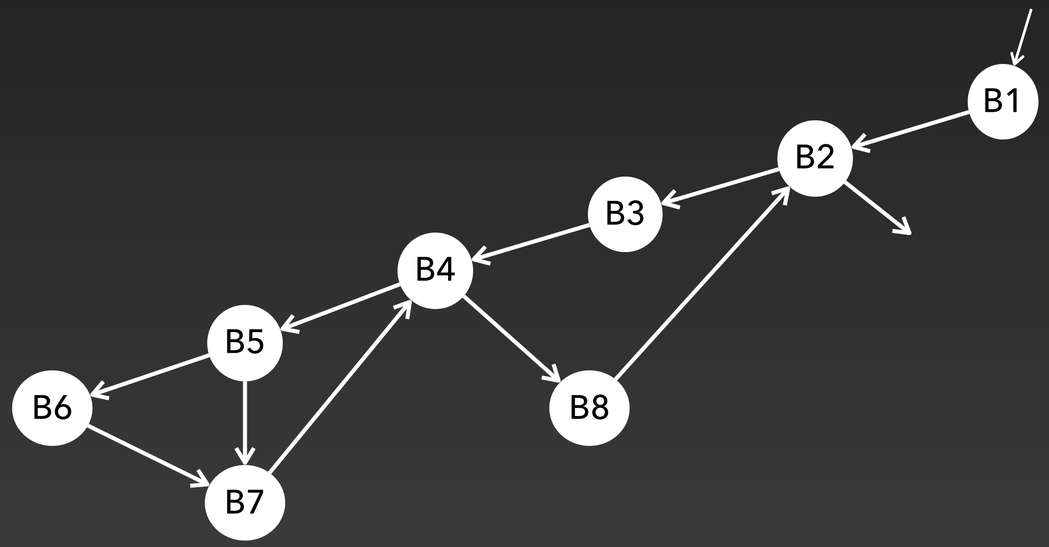
```
    t8 := j-1
    t9 := 4*t8
    temp := A[t9]     ;temp:=A[j]
    t10 := j+1
    t11:= t10-1
    t12 := 4*t11
    t13 := A[t12]     ;A[j+1]                 B6
    t14 := j-1
    t15 := 4*t14
    A[t15] := t13     ;A[j]:=A[j+1]
    t16 := j+1
    t17 := t16-1
    t18 := 4*t17
    A[t18]:=temp      ;A[j+1]:=temp
L3: j := j+1                                  B7
    goto L4
L2: i := i-1                                  B8
    goto L5
L1:
```

控制流图（Control Flow Graph）

- 节点：基本块
- 边：$B_i \rightarrow B_j$ 当切仅当 B_j 在某些执行下紧跟在B_i之后
  - 要么B_j的第一条指令是B_i最后一句跳转的目标
  - 要么B_j就是天然的跟在B_i后面，其中B_i不是以无条件跳转结尾
- 程序首个语句所在的块叫做start或者entry节点

控制流图（Control Flow Graph）

```
    i := n-1                                  B1
L5: if i<1 goto L1                            B2
    j := 1                                    B3
L4: if j>i goto L2                            B4
    t1 := j-1
    t2 := 4*t1
    t3 := A[t2]     ;A[j]
    t4 := j+1
    t5 := t4-1                                B5
    t6 := 4*t5
    t7 := A[t6]     ;A[j+1]
    if t3<=t7 goto L3
```




```
    t8 := j-1
    t9 := 4*t8
    temp := A[t9]     ;temp:=A[j]
    t10 := j+1
    t11:= t10-1
    t12 := 4*t11
    t13 := A[t12]     ;A[j+1]                 B6
    t14 := j-1
    t15 := 4*t14
    A[t15] := t13     ;A[j]:=A[j+1]
    t16 := j+1
    t17 := t16-1
    t18 := 4*t17
    A[t18]:=temp      ;A[j+1]:=temp
L3: j := j+1                                  B7
    goto L4
L2: i := i-1                                  B8
    goto L5
L1:
```

优化的种类

- 算法优化： 请熟读算法导论以及STOC、FOCS…的paper，但如果别人写出算法复杂度高的程序，一般而言，难以通过编译器优化来优化算法
- 代数优化（Algebraic optimization）: 如 A := B + 0 → A := B
- 本地优化: 在一个基本块内进行优化（across instructions）
- 全局优化（Global optimizations）： 在一个控制流图中优化（across basic blocks）
- 过程间优化 （inter-procedural）： 在一个程序内优化（across procedures，多个控制流图）

本地优化

- 在一个基本块内进行的分析和代码转换
- 不需要考虑控制流信息
- 典型的本地优化例子：
  - 本地公共子表达式消除（Common subexpression elimination）
    - 分析：在一个基本块内被多次求值的同一个表达式
    - 转换：替换为单个的计算
  - 本地常量折叠（Constant folding）或者 消除
    - 分析： 表达式的求值可以直接在编译阶段完成
    - 转换：将求值表达式直接替换为编译阶段计算的常量
  - 死代码消除（dead code elimination）

本地优化

```
    i := n-1
L5: if i<1 goto L1
    j := 1
L4: if j>i goto L2
    t1 := j-1
    t2 := 4*t1
    t3 := A[t2]    ;A[j]
    t4 := j+1
    t5 := t4-1
    t6 := 4*t5
    t7 := A[t6]    ;A[j+1]
    if t3<=t7 goto L3
```

B5

冗余的计算

```
    t8 := j-1
    t9 := 4*t8
    temp := A[t9]      ;temp:=A[j]
    t10 := j+1
    t11:= t10-1
    t12 := 4*t11
    t13 := A[t12]      ;A[j+1]
    t14 := j-1
    t15 := 4*t14
    A[t15] := t13      ;A[j]:=A[j+1]
    t16 := j+1
    t17 := t16-1
    t18 := 4*t17
    A[t18]:=temp       ;A[j+1]:=temp
L3: j := j+1
    goto L4
L2: i := i-1
    goto L5
L1:
```

本地优化

```
    i := n-1
L5: if i<1 goto L1
    j := 1
L4: if j>i goto L2
    t1 := j-1
    t2 := 4*t1
    t3 := A[t2]     ;A[j]
    t6 := 4*j
    t7 := A[t6]     ;A[j+1]
    if t3<=t7 goto L3
```

B5

```
    t8 := j-1
    t9 := 4*t8
    temp := A[t9]       ;temp:=A[j]
    t10 := j+1
    t11:= t10-1
    t12 := 4*t11
    t13 := A[t12]       ;A[j+1]
    t14 := j-1
    t15 := 4*t14
    A[t15] := t13       ;A[j]:=A[j+1]
    t16 := j+1
    t17 := t16-1
    t18 := 4*t17
    A[t18]:=temp        ;A[j+1]:=temp
L3: j := j+1
    goto L4
L2: i := i-1
    goto L5
L1:
```

本地优化

```
    i := n-1
L5: if i<1 goto L1
    j := 1
L4: if j>i goto L2
    t1 := j-1
    t2 := 4*t1
    t3 := A[t2]    ;A[j]
    t6 := 4*j
    t7 := A[t6]    ;A[j+1]
    if t3<=t7 goto L3
```

```
    t8 := j-1
    t9 := 4*t8
    temp := A[t9]      ;temp:=A[j]
    t10 := j+1
    t11:= t10-1
    t12 := 4*t11
    t13 := A[t12]      ;A[j+1]
    t14 := j-1
    t15 := 4*t14
    A[t15] := t13      ;A[j]:=A[j+1]
    t16 := j+1
    t17 := t16-1
    t18 := 4*t17
    A[t18]:=temp       ;A[j+1]:=temp
```

B6

```
L3: j := j+1
    goto L4
L2: i := i-1
    goto L5
L1:
```

本地优化

```
    i := n-1
L5: if i<1 goto L1
    j := 1
L4: if j>i goto L2
    t1 := j-1
    t2 := 4*t1
    t3 := A[t2]    ;A[j]
    t6 := 4*j
    t7 := A[t6]    ;A[j+1]
    if t3<=t7 goto L3
```

```
    t8 := j-1
    t9 := 4*t8
    temp := A[t9]     ;temp:=A[j]
    t12 := 4*j                                  B6
    t13 := A[t12]     ;A[j+1]
    A[t9] := t13     ;A[j]:=A[j+1]
    A[t12]:=temp      ;A[j+1]:=temp
L3: j := j+1
    goto L4
L2: i := i-1
    goto L5
L1:
```

（Intraprocedural）全局优化

- 全局优化 VS 本地优化
  - 全局的公共子表达式消除
  - 全局的常量传播
  - 死代码消除
- 循环优化（Loop optimizations）
  - 循环展开（Loop unroll）、循环融合（Loop fusion）
  - 代码移动（Code motion）：比如循环不变式外提(Loop-Invariant Code Motion, LICM)
  - 归纳变量消除（Induction variable elimination）
- 其他控制结构：
  - 消除流程图中并行路径上相同代码的副本，以减少代码体积

全局（横跨多个基本块）优化例子

```
     i := n-1
L5:  if i<1 goto L1
     j := 1
L4:  if j>i goto L2
     t1 := j-1
     t2 := 4*t1
     t3 := A[t2]    ;A[j]
     t6 := 4*j
     t7 := A[t6]    ;A[j+1]
     if t3<=t7 goto L3
```

B5

```
     t8 := j-1
     t9 := 4*t8
     temp := A[t9]     ;temp:=A[j]
     t12 := 4*j
     t13 := A[t12]     ;A[j+1]
     A[t9] := t13     ;A[j]:=A[j+1]
     A[t12]:=temp      ;A[j+1]:=temp
L3:  j := j+1
     goto L4
L2:  i := i-1
     goto L5
L1:
```

B6

公共子表达式！

全局（横跨多个基本块）优化例子

```
    i := n-1
L5: if i<1 goto L1
    j := 1
L4: if j>i goto L2
    t1 := j-1
    t2 := 4*t1
    t3 := A[t2]    ;A[j]
    t6 := 4*j
    t7 := A[t6]    ;A[j+1]
    if t3<=t7 goto L3
```

B5

```
    A[t2] := t7    ;A[j]:=A[j+1]
    A[t6]:=t3      ;A[j+1]:=old_A[j]
L3: j := j+1
    goto L4
L2: i := i-1
    goto L5
L1:
```

B6

公共子表达式！

归纳变量消除(Induction Variable Elimination)

- 直观上
  - 循环索引是归纳变量（即循环的每次迭代中增加或减少固定量的变量，用于计算迭代次数）
  - 一个归纳变量的线性函数映射的变量也是归纳变量 (一般用于访问数组)
- 分析: 侦测到相应的归纳变量
- 优化
  - 强度削弱(strength reduction):
    - 将乘法变为加法或者位移
  - 消除循环索引
    - 替换为别的归纳变量（一般将循环终止条件替换为其他归纳变量的测试）

归纳变量消除

```
    i := n-1
L5: if i<1 goto L1
    j := 1
L4: if j>i goto L2
    t1 := j-1
    t2 := 4*t1
    t3 := A[t2]    ;A[j]
    t6 := 4*j
    t7 := A[t6]    ;A[j+1]
    if t3<=t7 goto L3
    A[t2] := t7    ;A[j]:=A[j+1]
    A[t6]:=t3      ;A[j+1]:=old_A[j]
L3: j := j+1
    goto L4
L2: i := i-1
    goto L5
L1:
```

```
    i := n-1
L5: if i<1 goto L1
    t2 := 0
    t6 := 4
L4: t19 := 4*i
    if t6>t19 goto L2
    t3 := A[t2]    ;A[j]
    t7 := A[t6]    ;A[j+1]
    if t3<=t7 goto L3
    A[t2] := t7    ;A[j]:=A[j+1]
    A[t6]:=t3      ;A[j+1]:=old_A[j]
L3: t2 := t2+4
    t6 := t6+4
    goto L4
L2: i := i-1
    goto L5
L1:
```

Q：会溢出吗?

循环不变式外提(Loop-Invariant Code Motion)

- 分析
  - 在循环内的某个计算，随着循环进行不会改变
- 变换
  - 将这个计算直接提到循环外部

循环不变式外提

```
    i := n-1
L5: if i<1 goto L1
    t2 := 0                                  B3
    t6 := 4
L4: t19 := 4*i                               B4
    if t6>t19 goto L2
    t3 := A[t2]    ;A[j]
    t7 := A[t6]    ;A[j+1]
    if t3<=t7 goto L3
    A[t2] := t7     ;A[j]:=A[j+1]
    A[t6]:=t3       ;A[j+1]:=old_A[j]
L3: t2 := t2+4
    t6 := t6+4
    goto L4
L2: i := i-1
    goto L5
L1:
```

```
    i := n-1
L5: if i<1 goto L1
    t2 := 0
    t6 := 4                                  B3
    t19 := 4*i
L4: if t6>t19 goto L2                        B4
    t3 := A[t2]    ;A[j]
    t7 := A[t6]    ;A[j+1]
    if t3<=t7 goto L3
    A[t2] := t7     ;A[j]:=A[j+1]
    A[t6]:=t3       ;A[j+1]:=old_A[j]
L3: t2 := t2+4
    t6 := t6+4
    goto L4
L2: i := i-1
    goto L5
L1:
```

最终代码

```
    i := n-1
L5: if i<1 goto L1
    t2 := 0
    t6 := 4
    t19 := 4*i
L4: if t6>t19 goto L2
    t3 := A[t2]    ;A[j]
    t7 := A[t6]    ;A[j+1]
    if t3<=t7 goto L3
    A[t2] := t7     ;A[j]:=A[j+1]
    A[t6]:=t3      ;A[j+1]:=old_A[j]
L3: t2 := t2+4
    t6 := t6+4
    goto L4
L2: i := i-1
    goto L5
L1:
```

机器相关优化（Machine Dependent Optimizations）

- 寄存器分配（Register allocation）
- 指令调度（Instruction scheduling）
- 内存层次结构优化（Memory hierarchy optimizations）
- …


Q&A
钮鑫涛