

# 编译优化技术概览

陈文光  
清华大学

# 个人与编译有关的科研与教学经历

- 1995-2000            交互式自动并行化编译器TIPS
- 2006-2010            开源编译器Open64中的优化
- 2008-2012            利用编译器进行程序分析
- 2010 -                高层编程系统（图计算系统）
- 2018-                编译优化与程序分析课程教学

# 编译系统的作用

- 翻译

- 支持高层的编程抽象
- 支持底层的硬件体系结构

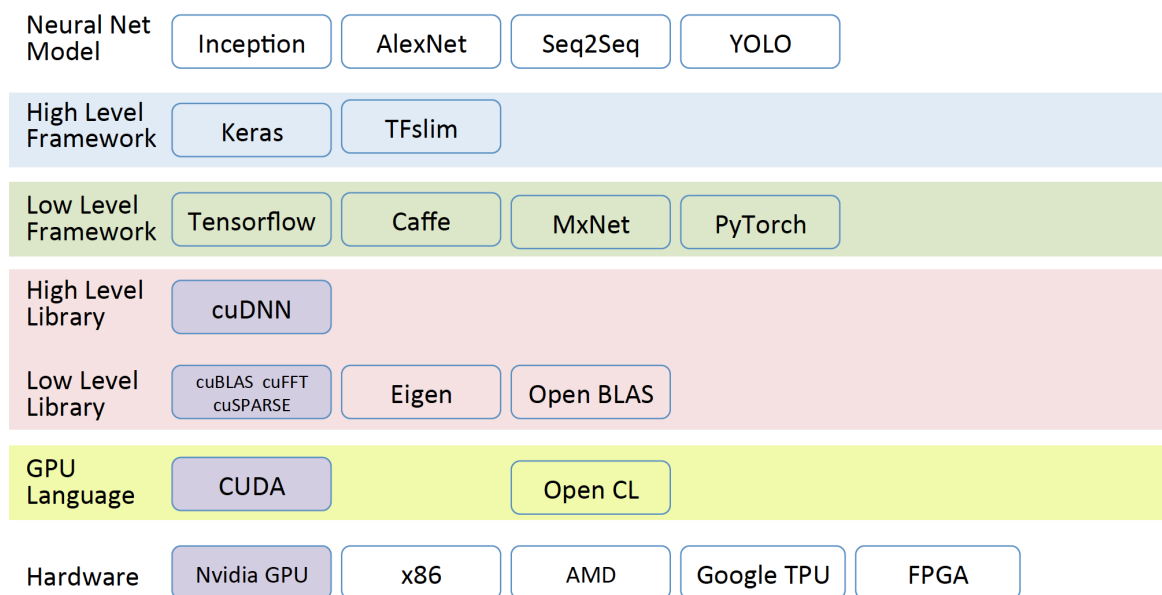
- 理解程序

- 安全性(security)
- 功能正确(safety)

- 优化

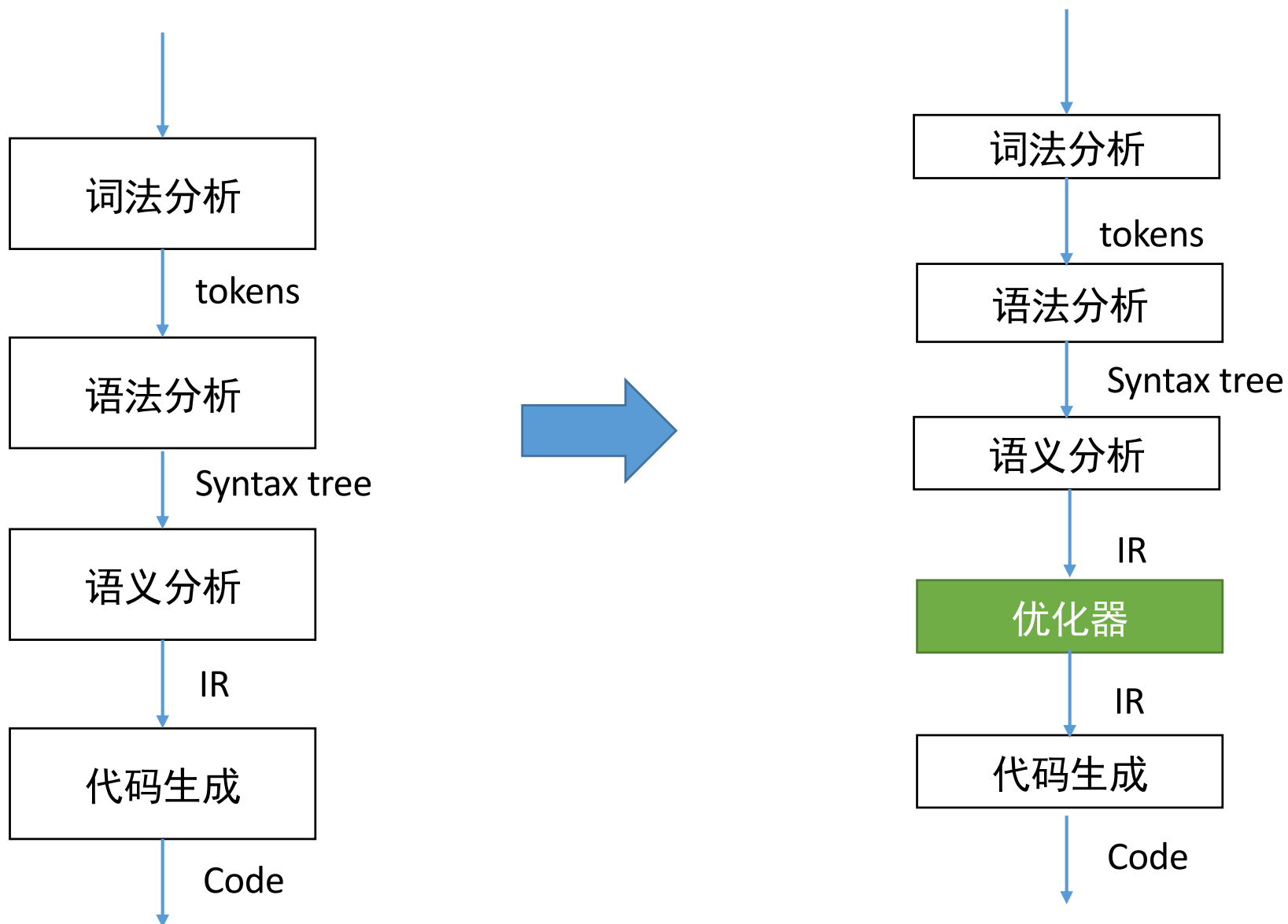
- 更快的执行速度
- 更小的空间

## Levels of Abstraction

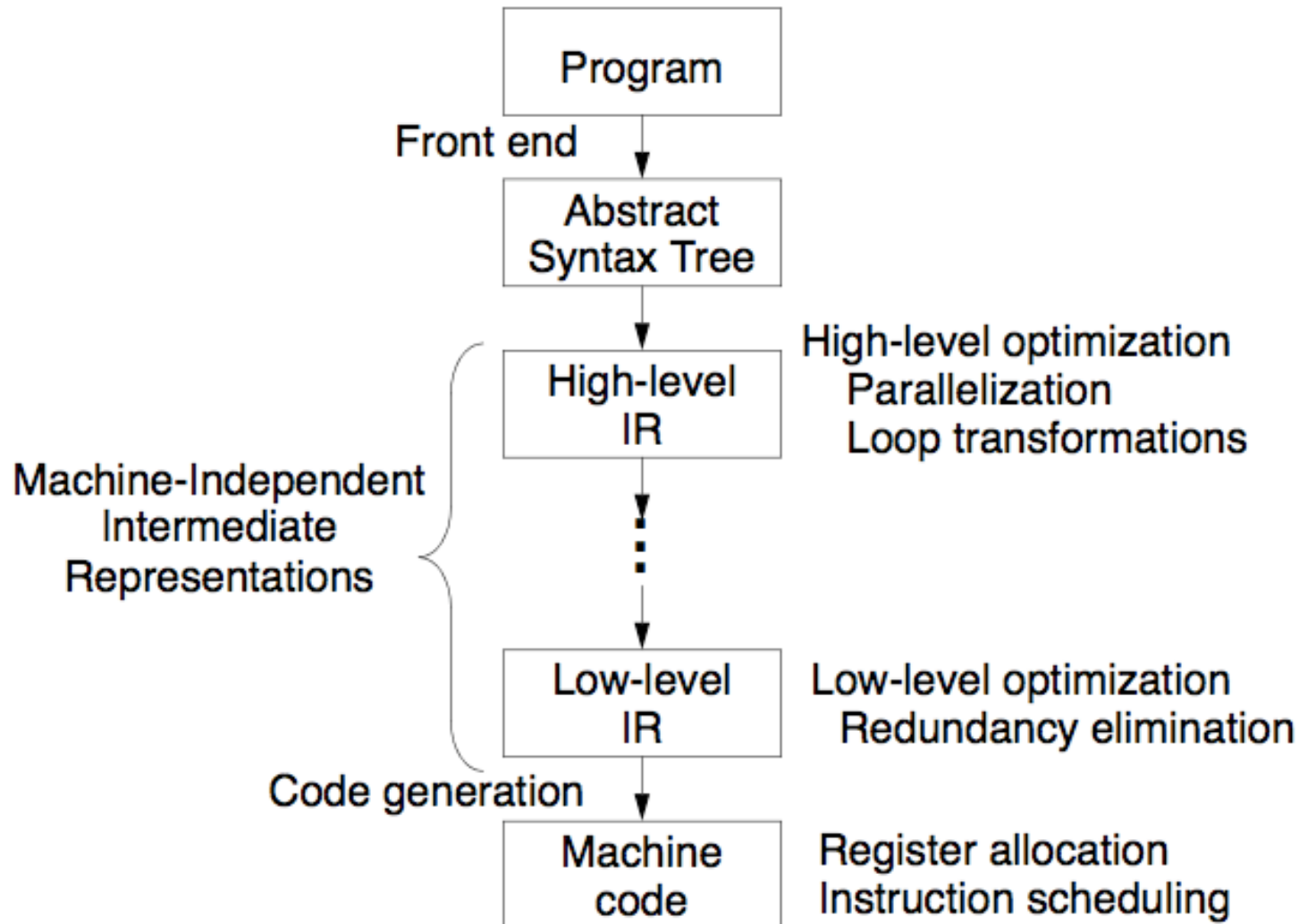


<https://suif.stanford.edu/~courses/cs243/>

# 编译器的架构



# 优化编译器的结构

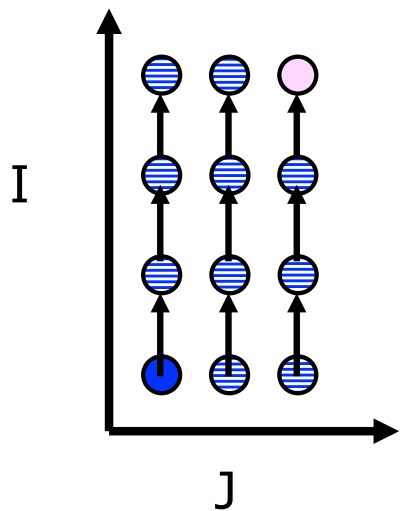


# 编译优化技术

- 体系结构无关
  - 高层优化技术
    - 循环变换 – 优化局部性或并行化
  - 底层优化技术
    - 冗余消除
- 体系结构相关
  - 寄存器分配
  - 指令调度

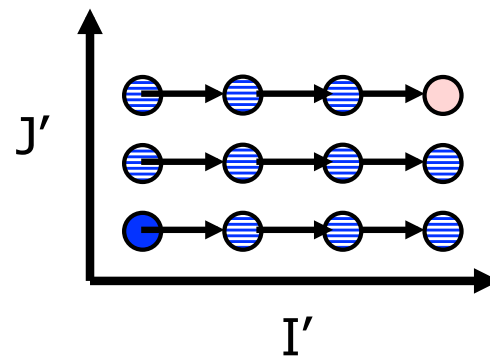
# 高层循环变换 – 循环交换

for I = 1 to 4  
  for J = 1 to 3  
    Z[I,J] = Z[I-1,J]



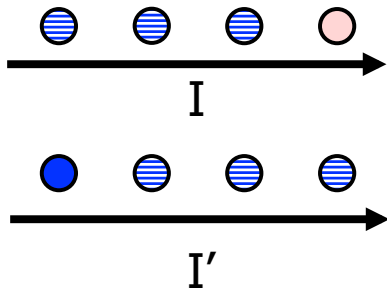
$$\begin{bmatrix} j' \\ i' \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} i \\ j \end{bmatrix}$$

for J' = 1 to 3  
  for I' = 1 to 4  
    Z[I',J'] = Z[I'-1,J']



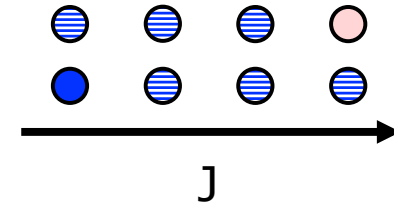
# 高层循环变换 – 循环合并

for I = 1 to 4  
  T[I] = A[I] + B[I] (s1)  
  for I' = 1 to 4  
    C[I'] = T[I'] x T[I'] (s2)



s1:  $[j] = [1] [i]$   
s2:  $[j] = [1] [i']$

for J = 1 to 4  
  T[J] = A[J] + B[J] (s1)  
  C[J] = T[J] x T[J] (s2)



# 高层循环变化－并行化

```
FOR i = 1 to 100  
    A[i] = B[i] + C[i]
```

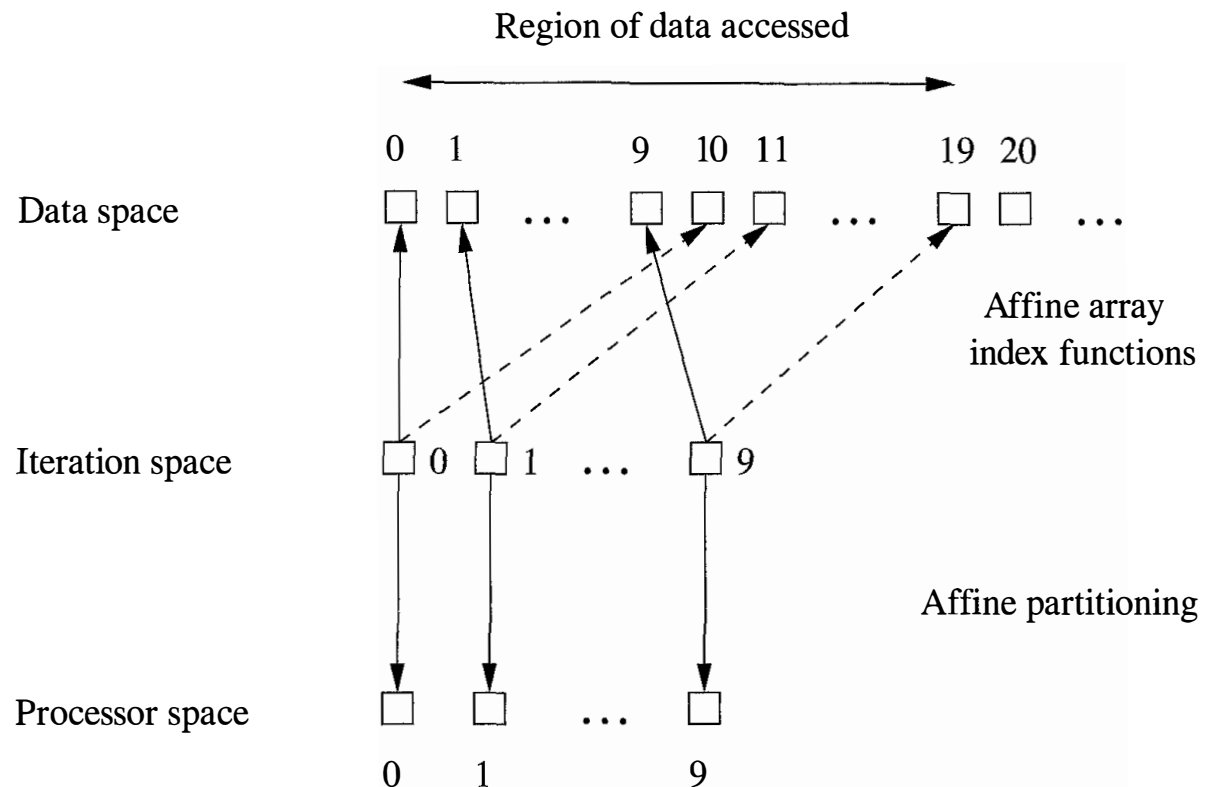
```
FOR i = 11 TO 20  
    a[i] = a[i-1] + 3
```

```
FOR i = 11 TO 20  
    a[i] = a[i-10] + 3
```

# 支持高层循环优化的核心技术

- 依赖性分析
  - 如何判断一个循环没有跨循环迭代的依赖性?
- 仿射 (Affine) 变换技术
  - 三个空间
    - Iteration space
    - data space
    - Processor space

```
float Z[100];  
for (i = 0; i < 10; i++)  
    Z[i+10] = Z[i];
```



# 编译优化技术

- 体系结构无关
  - 高层优化技术
    - 循环变换 – 优化局部性或并行化
  - 底层优化技术
    - 冗余消除
- 体系结构相关
  - 寄存器分配
  - 指令调度

# 体系结构无关的底层优化

- 从高层抽象翻译到底层的过程中引入了冗余
- 优化过程就是消除冗余的过程

```
#include <stdio.h>
```

```
int foo()
```

```
{
```

```
    int a[4];
```

```
    a[0] = 1;
```

```
    a[1] = a[0];
```

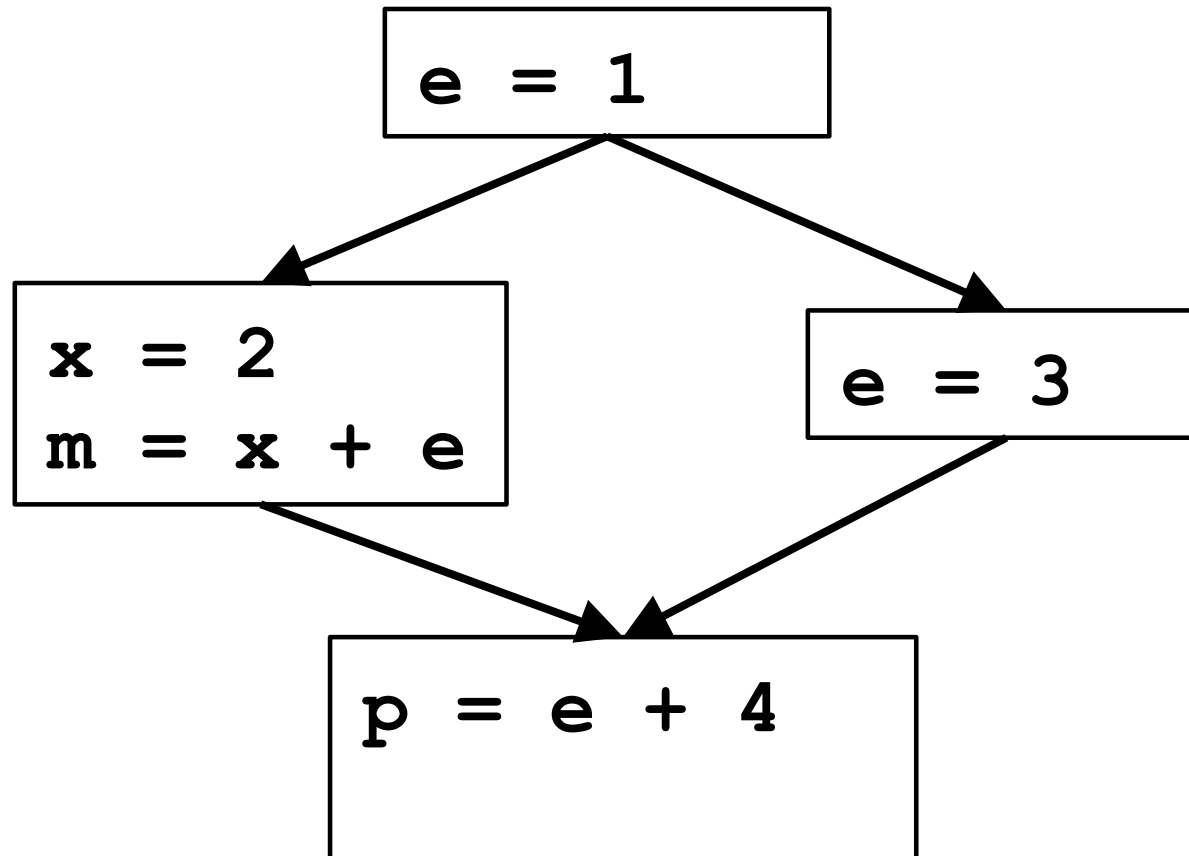
```
    return a[1];
```

```
}
```

```
pushq    %rbp
movq     %rsp, %rbp
subq     $0x30, %rsp
movq     (%rip), %rax
movq     (%rax), %rax
movq     %rax, -0x8(%rbp)
movl     $0x1, -0x20(%rbp)
movl     -0x20(%rbp), %ecx
movl     %ecx, -0x1c(%rbp)
movl     -0x1c(%rbp), %eax
movq     (%rip), %rdx
movq     (%rdx), %rdx
movq     -0x8(%rbp), %rsi
cmpq     %rsi, %rdx
movl     %eax, -0x24(%rbp)
jne      0x49
movl     -0x24(%rbp), %eax
addq     $0x30, %rsp
popq     %rbp
retq
callq    0x4e
```

```
pushq    %rbp
movq     %rsp, %rbp
movl     $0x1, %eax
popq     %rbp
retq
```

# 常数传播



# 公共子表达式消除

```
t6 = 4*i  
x = a[t6]  
t7 = 4*i  
t8 = 4*j  
t9 = a[t8]  
a[t7] = t9  
t10 = 4*j  
a[t10] = x  
goto B2
```

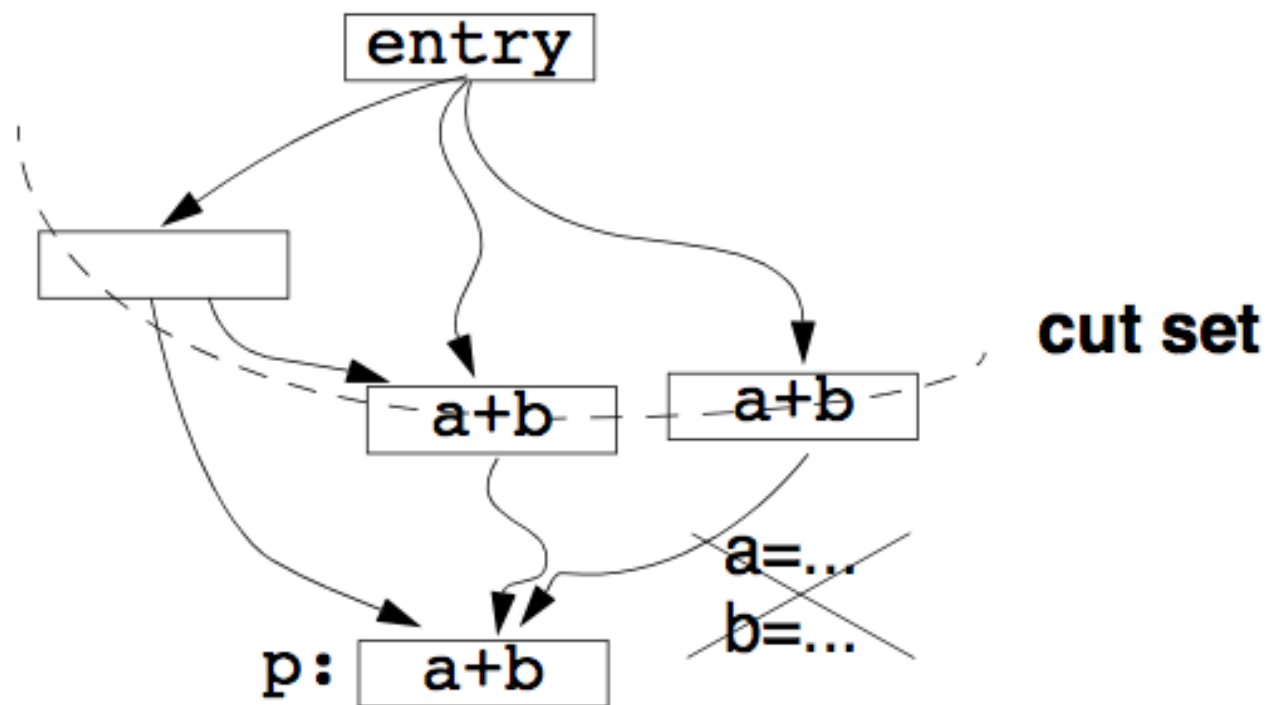
$B_5$



```
t6 = 4*i  
x = a[t6]  
t8 = 4*j  
t9 = a[t8]  
a[t6] = t9  
a[t8] = x  
goto B2
```

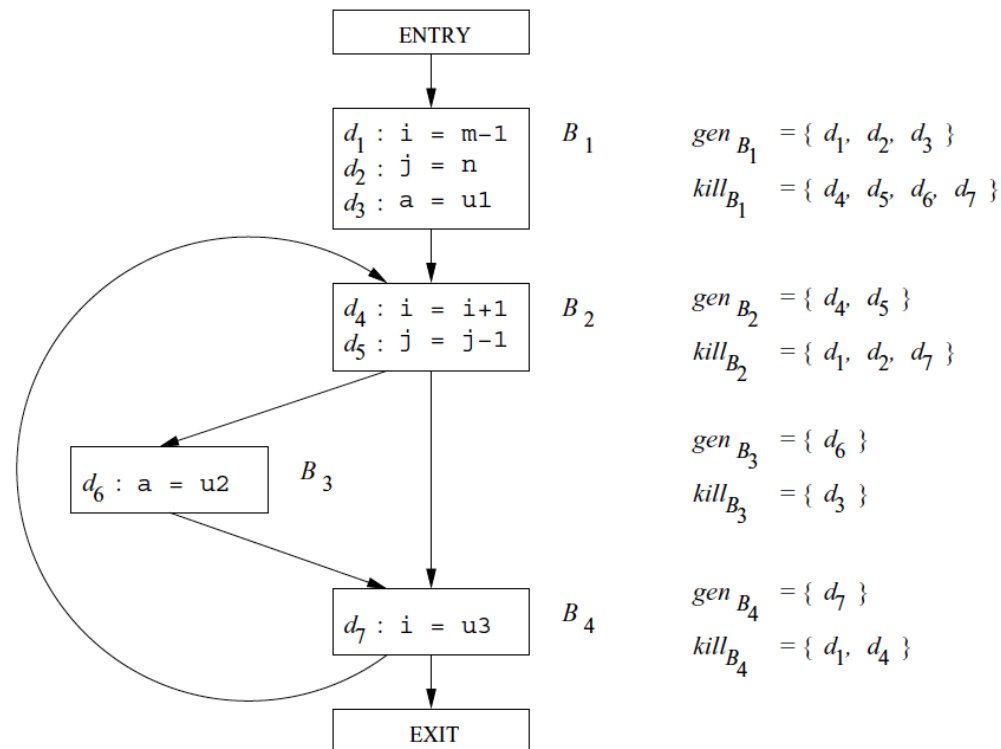
$B_5$

# 部分冗余消除



# 主要支撑技术-数据流分析

- $OUT[ENTRY] = \{\}$
- $OUT[B] = gen_B \cup (IN[B] - kill_B)$
- $IN[B] = \bigcup_{P \text{ a predecessor of } B} OUT[P]$

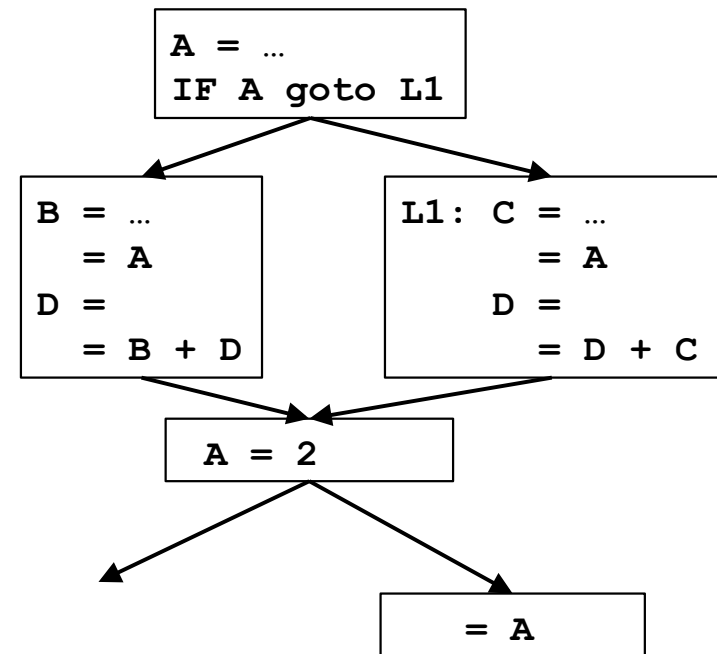
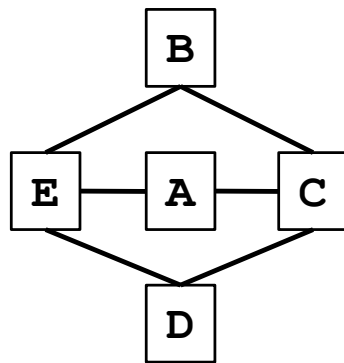


# 编译优化技术

- 体系结构无关
  - 高层优化技术
    - 循环变换 – 优化局部性或并行化
  - 底层优化技术
    - 冗余消除
- 体系结构相关
  - 寄存器分配
  - 指令调度

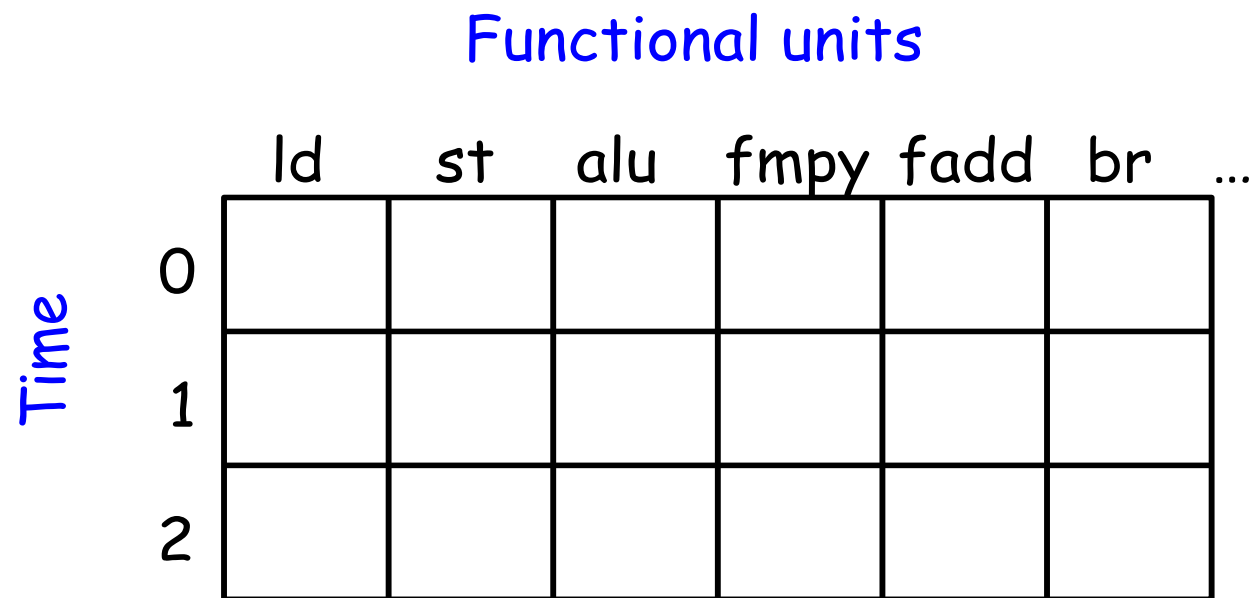
# 体系结构相关的优化

- 寄存器分配
  - 虚拟寄存器
  - 构建Interference graph
  - 图着色算法



# 指令调度

- CPU中的功能部件数
- 是否流水
- 延迟



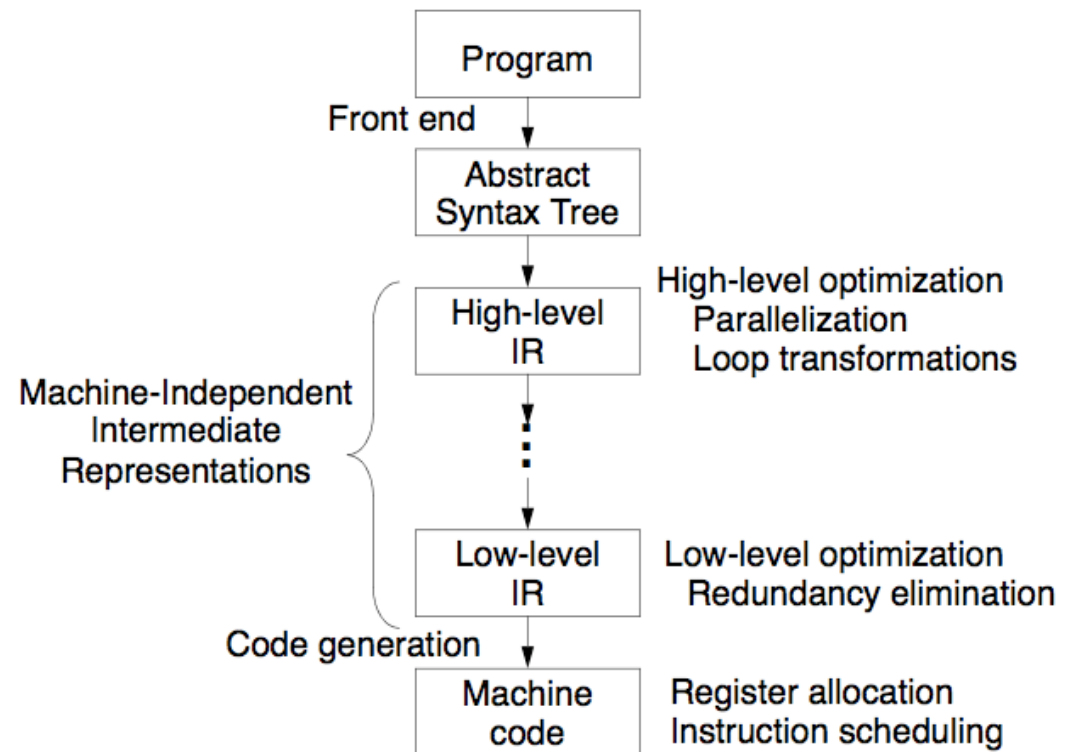
# 程序优化的现状

- CPU上的过程内优化基本成熟
- 过程间优化能力仍然受限
- 面向GPU等新型体系结构的编译优化还有空间

# GPU存储结构对编程优化的挑战

- 消除冗余计算的循环不变量外提可能引起寄存器压力大，从而引发spill，在GPU上开销很大

```
t1 := (100*c)
t2 := (c * 5)
do i := 1, 100
  l := i + t2
  t3 := t1 + 10 * i
  do j := 1, 100
    a(i,j) := t3 + j
  enddo
enddo
```



# Open64

