有趣有用的编译教学（初探）

陈文光
清华大学/青海大学

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

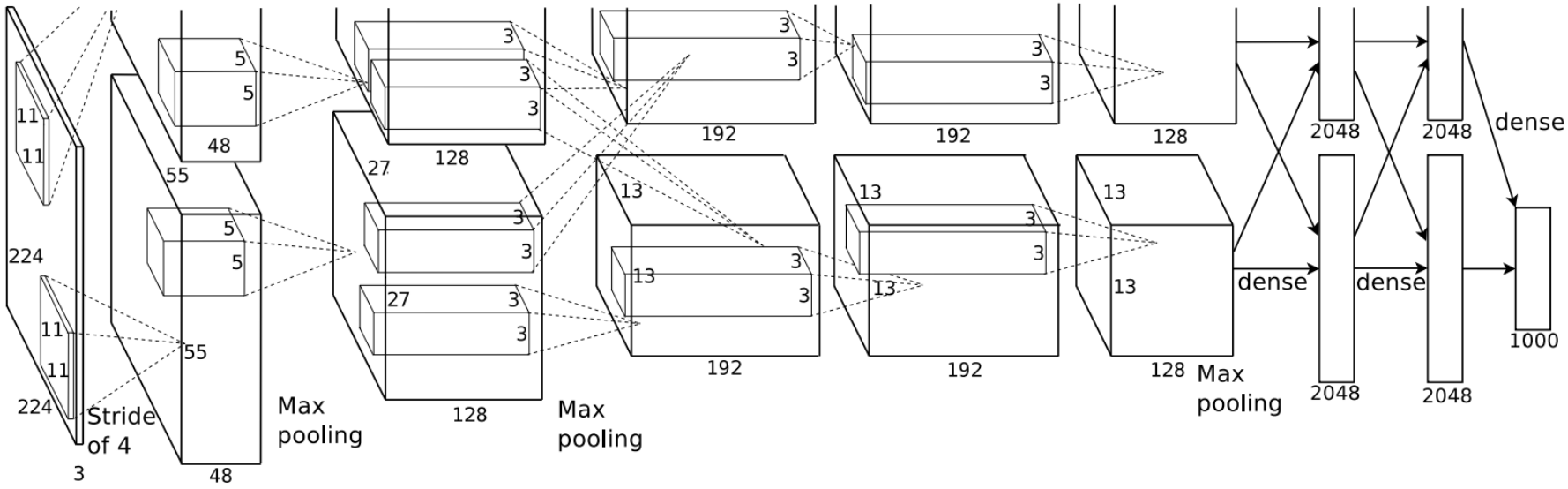
有趣有用 vs 无趣无用？

- 有用即有趣
- 编译的作用是什么？
- 工业界需要什么样的编译人才？

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
- 优化
  - 更快的执行速度
  - 更小的空间
- 理解程序
  - 安全性(security)
  - 功能正确(safety)

新的应用需要新的编程抽象

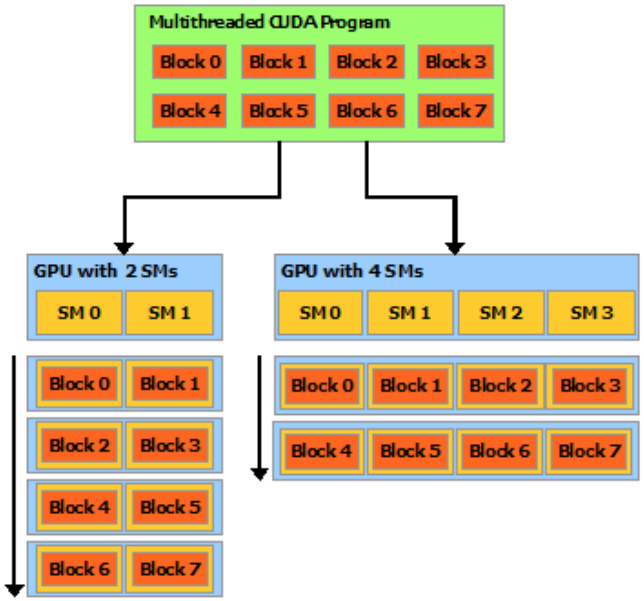


ImageNet: http://www.cs.toronto.edu/~fritz/absps/imagenet.pdf

新的底层硬件

- GPU，FPGA,寒武纪， …
- CUDA

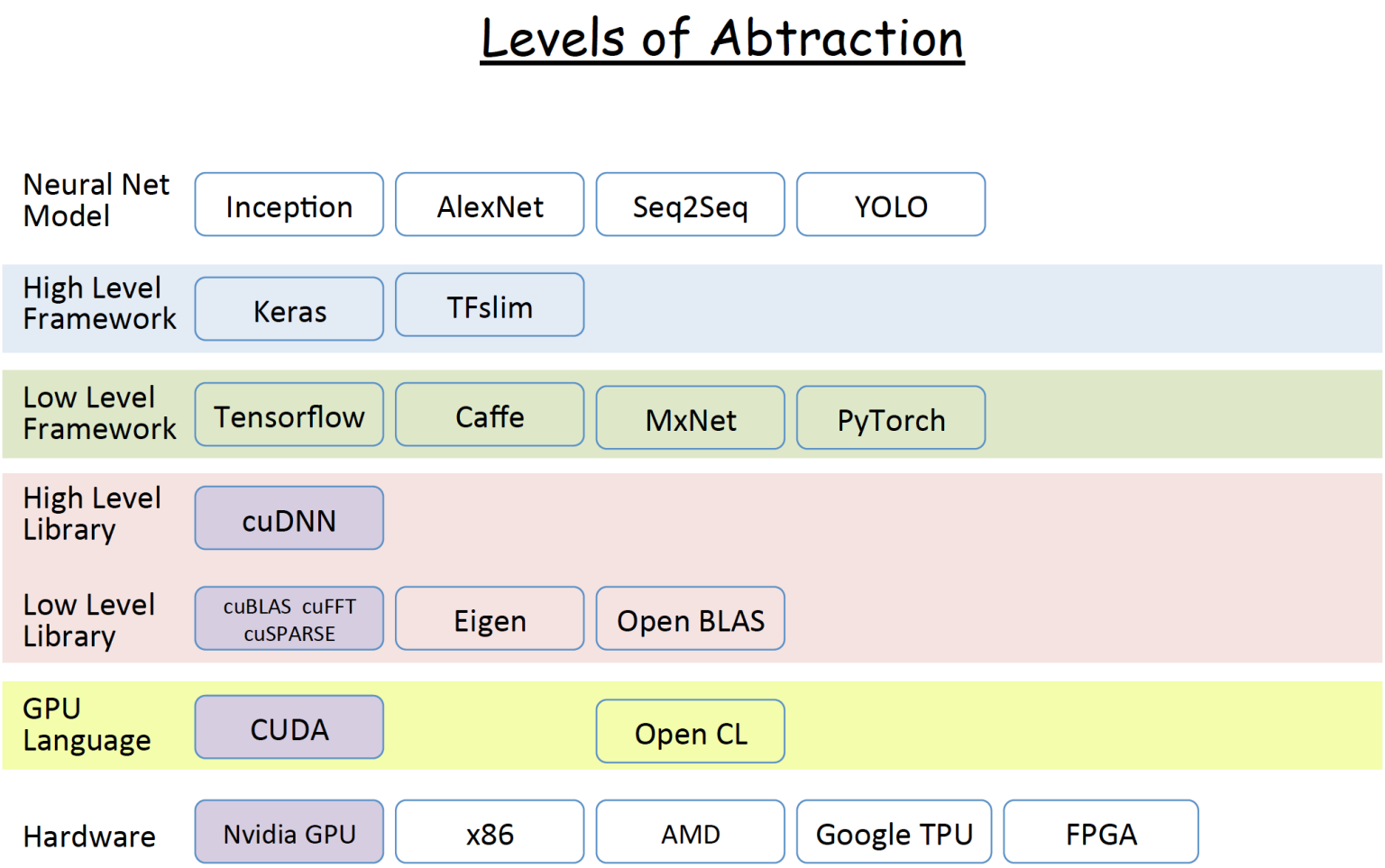
```
__global__
void add(int n, float *x, float *y)
{
  int index = threadIdx.x;
  int stride = blockDim.x;
  for (int i = index; i < n; i += stride)
      y[i] = x[i] + y[i];
}
```




```
int blockSize = 256;
int numBlocks = (N + blockSize - 1) / blockSize;
add<<<numBlocks, blockSize>>>(N, x, y);
```

https://devblogs.nvidia.com/even-easier-introduction-cuda/

多层抽象 - 编程框架与库

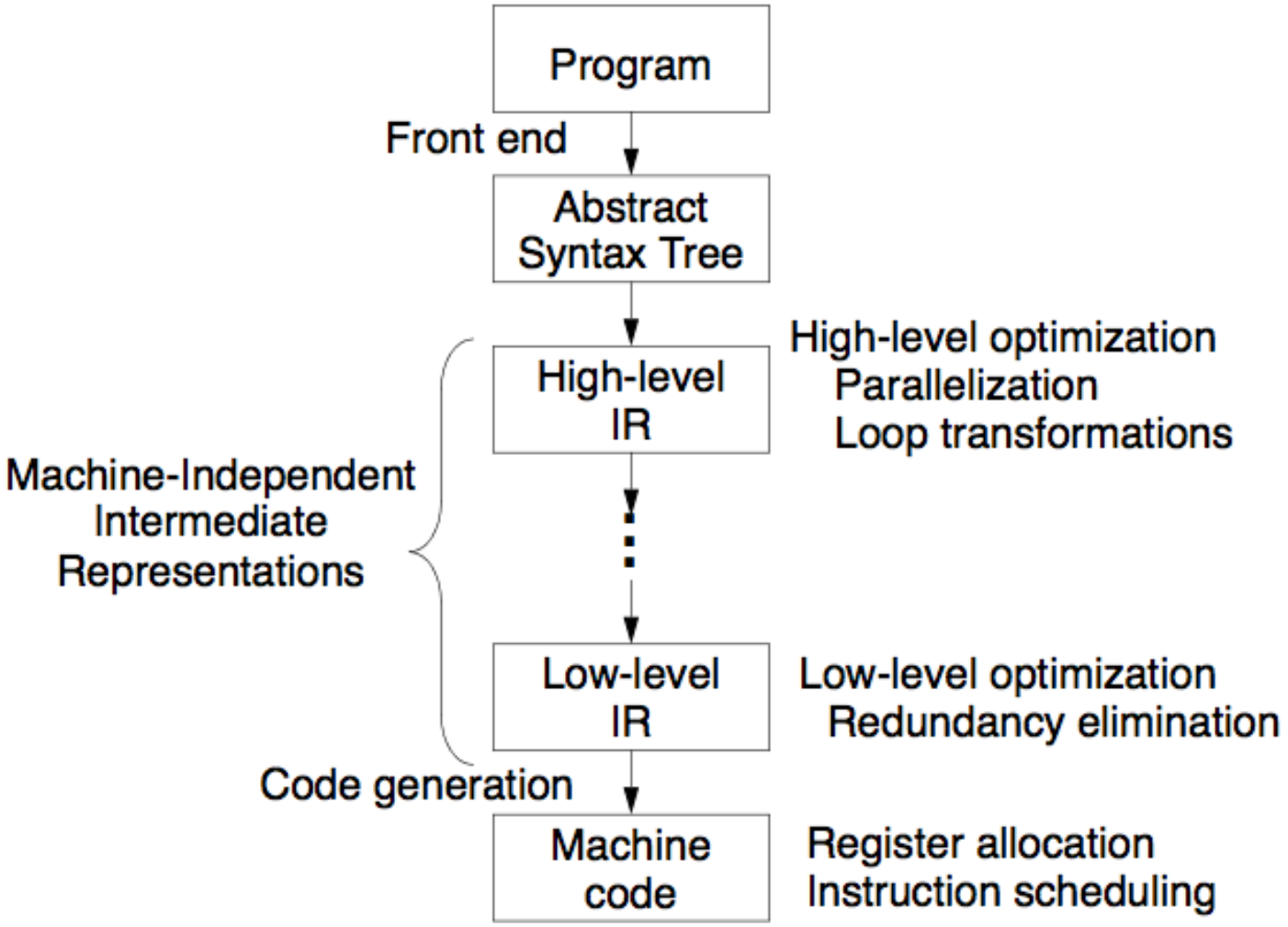


https://suif.stanford.edu/~courses/cs243/

编译系统的作用

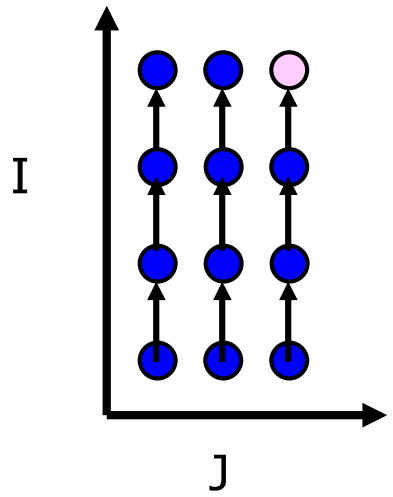
- 翻译
  - 支持高层的编程抽象
  - 支持底层的硬件体系结构
- 优化
  - 更快的执行速度
  - 更小的空间
- 理解程序
  - 安全性(security)
  - 功能正确(safety)

优化编译器的结构


Program
Front end
Abstract
Syntax Tree
High-level optimization
Parallelization
Loop transformations
High-level
IR
Machine-Independent
Intermediate
Representations
Low-level
IR
Low-level optimization
Redundancy elimination
Code generation
Machine
code
Register allocation
Instruction scheduling

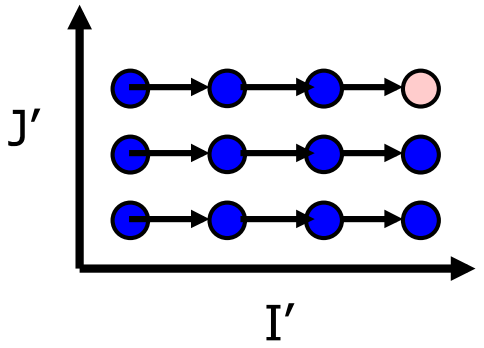
高层循环变换 – 循环交换

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



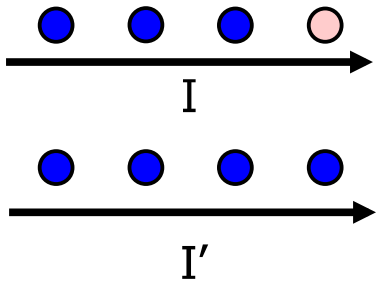
$$\begin{bmatrix} j' \\ i' \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} i \\ j \end{bmatrix}$$

```
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

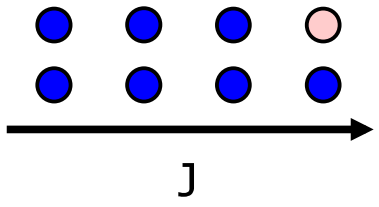
```
for I = 1 to 4
    T[I]= A[I]+B[I]     (s1)
for I' = 1 to 4
    C[I']= T[I'] x T[I'] (s2)
```



s1: $\begin{bmatrix} j \end{bmatrix} = \begin{bmatrix} 1 \end{bmatrix} \begin{bmatrix} i \end{bmatrix}$

s2: $\begin{bmatrix} j \end{bmatrix} = \begin{bmatrix} 1 \end{bmatrix} \begin{bmatrix} i' \end{bmatrix}$

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



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
```

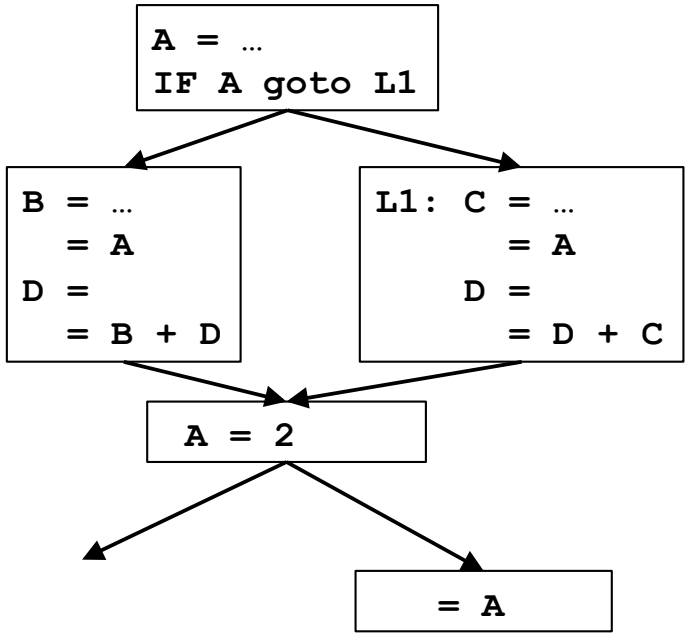
例子：循环不变量外提

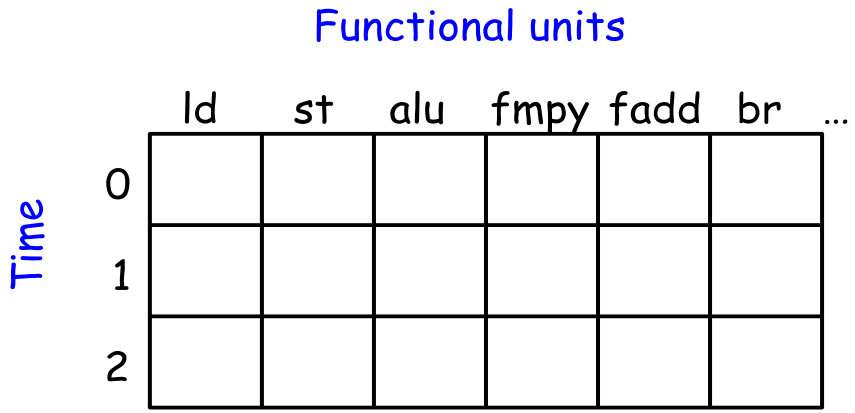
```
do i := 1, 100
    l := i + (c * 5)
    do j := 1, 100
        a(i.j) := 100*c  + 10*i +j
    enddo
enddo
```

```
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
```

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





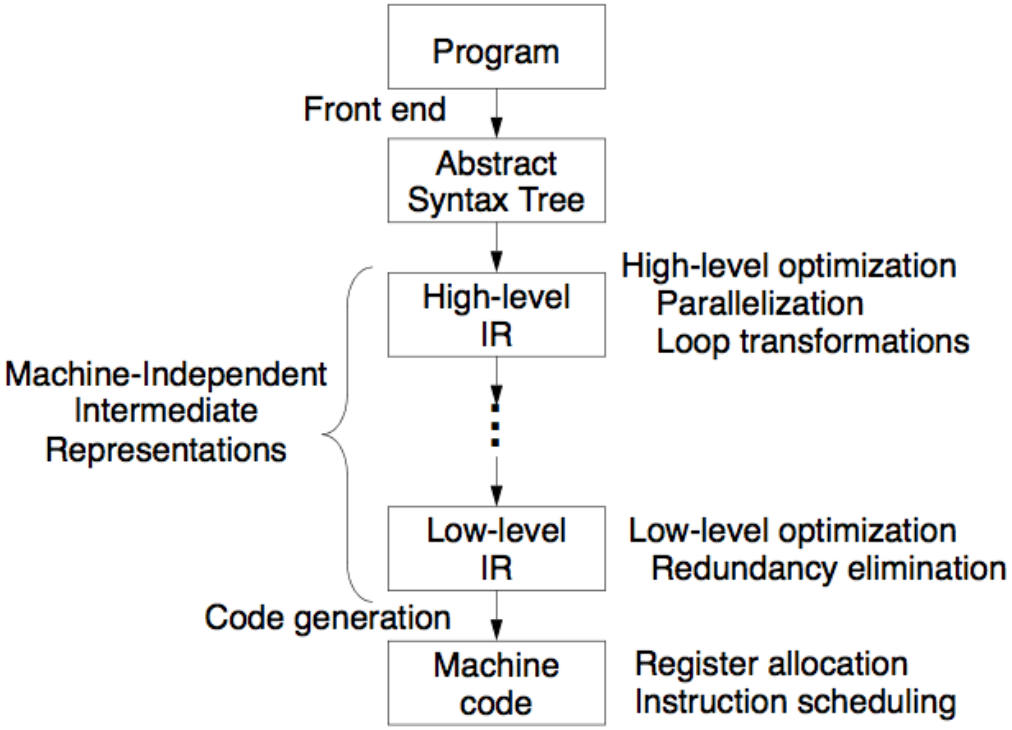
程序优化的现状

- 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

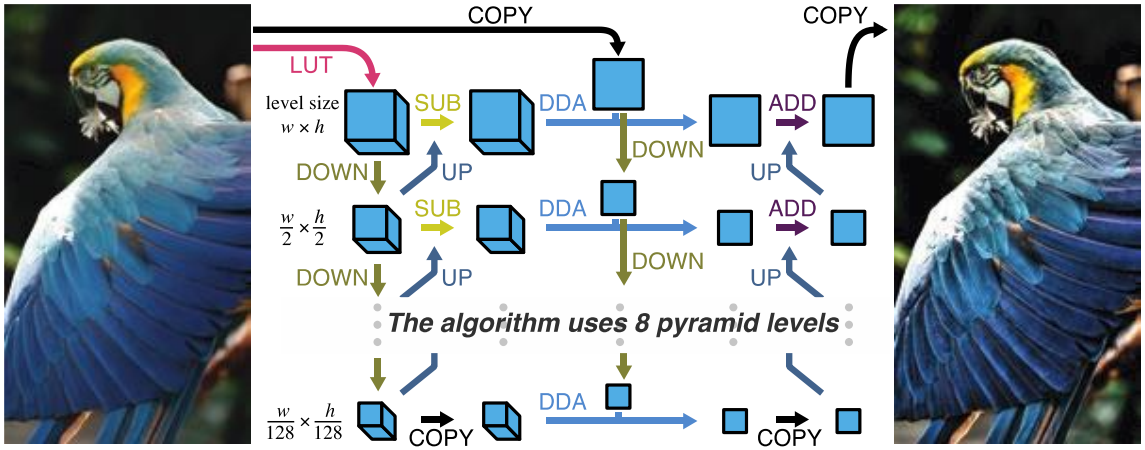
- 消除冗余计算的循环不变量外提可能引起寄存器压力大，从而引发spill，在GPU上开销很大

```
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
```



DSL – 领域特定语言(抽象+优化)

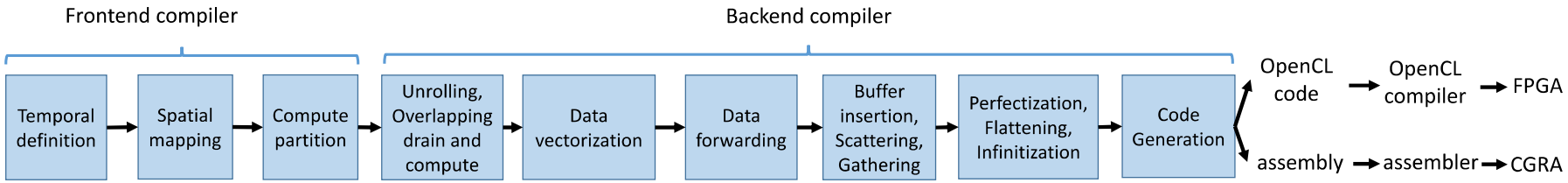
- Halide：面向图像处理的DSL




```
UniformImage in(UInt(8), 2)
Var x, y
Func blurx(x,y) =  in(x-1,y) +    in(x,y) +    in(x+1,y)
Func out(x,y) = blurx(x,y-1) + blurx(x,y) + blurx(x,y+1)
```

更多DSL的例子

- T2S – time to spatial
  - 基于Halide的DSL 和 FPGA编译器



- TACO – 稀疏张量编译器
  - 代数表示的稀疏张量DSL，编译到CPU
    - C = A*B + D
    - 其中A,B,C,D是高维张量，且可能在任意一维是稀疏的

- 面向GPU的稀疏张量编译器

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
- 优化
  - 更快的执行速度
  - 更小的空间
- 理解程序
  - 安全性(security)
  - 功能正确(safety)

关键软件系统

- 飞机和宇宙飞船
- 医疗设备
- 核电站
- 自动驾驶汽车

Image: http://dailypost.ng/2017/07/11/fg-forget-building-proposed-nuclear-power-plant-group/

智能合约

- 如何知道一个智能合约是否是安全的？

```
255   function batchTransfer(address[] _receivers, uint256 _value) public whenNotPaused returns (bool) {
256     uint cnt = _receivers.length;
257     uint256 amount = uint256(cnt) * _value;
258     require(cnt > 0 && cnt <= 20);
259     require(_value > 0 && balances[msg.sender] >= amount);
260
261     balances[msg.sender] = balances[msg.sender].sub(amount);
262     for (uint i = 0; i < cnt; i++) {
263         balances[_receivers[i]] = balances[_receivers[i]].add(_value);
264         Transfer(msg.sender, _receivers[i], _value);
265     }
266     return true;
267   }
268 }
```

Image credit from the [CVE-2018–10299 security alert]

SQL 注入 – 染色分析

SELECT *balance* FROM *AcctData*
WHERE *name* = ‘ *:n’* and password = *‘:p’*

where *n* and *p* are passed by another procedure

n = Charles Dickens’ - - p = who cares

SELECT balance FROM AcctData
WHERE name = ‘ Charles Dickens’ - - and password = ‘who cares’

缓冲区溢出分析

```
foo(char* s)
{
      char buf[32];
      strcpy(buf, s)
}
```

Will this *strcpy* overflow? You will need information from the caller of *foo*

程序分析的方法

- SAT
  - 程序建模成逻辑函数（只有true/false)
- SMT
  - 程序建模成逻辑函数（判断整数表达式的可满足性 如a+b > 5）
- Reachibility
  - 程序抽象成图，程序分析问题建模为图上的可达性问题

编译人才的需求（有用）

- 新的芯片
  - 编译器后端移植和优化
- 新的应用
  - DSL的设计与编译（如halide）
  - 编程系统的设计与实现（如spark）
- 程序分析与验证

目前的编译教学只覆盖了前端和极少的后端
与实用脱节明显，无用所以无趣

清华改革编译教学的初步探索

- 基础编译 + 优化与分析
- 基础编译（32学时）
  - 前端语法语义分析，简单的代码生成
- 优化与分析（32学时）
  - 数据流（体系结构无关优化）
  - 寄存器分配/指令调度（体系结构相关优化）
  - 程序分析

未来计划

- 加强与基础编译课程的协调
- 设计好实验平台
- 更紧凑合理的内容安排
- 加强语言特性和运行时的内容