

编译系统新技术及 发展

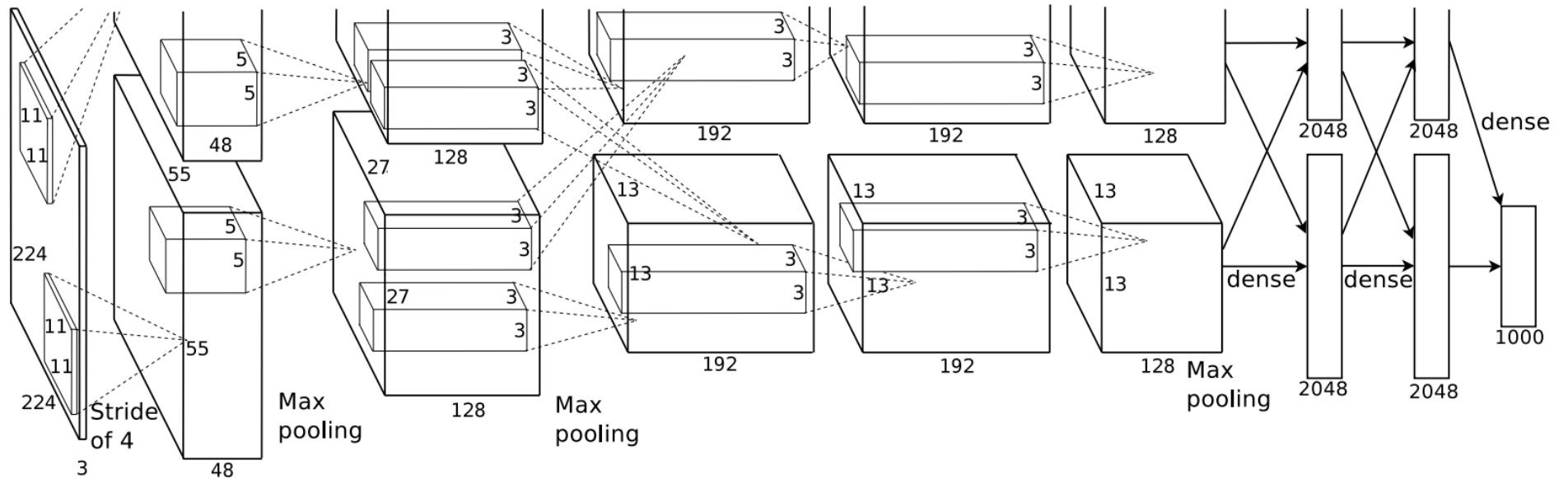
陈文光

清华大学/青海大学

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
- **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



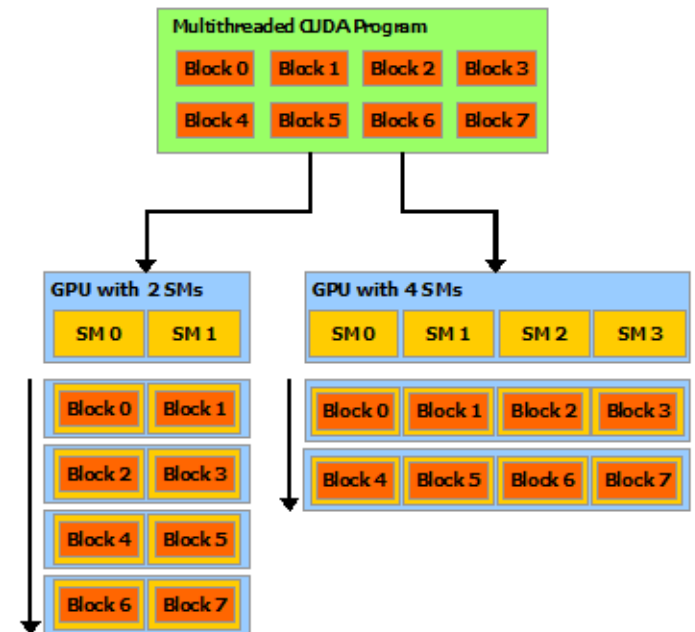
ImageNet: <http://www.cs.toronto.edu/~fritz/absps/imagenet.pdf>

新的底层硬件

- GPU, FPGA, 寒武纪, ...
- CUDA

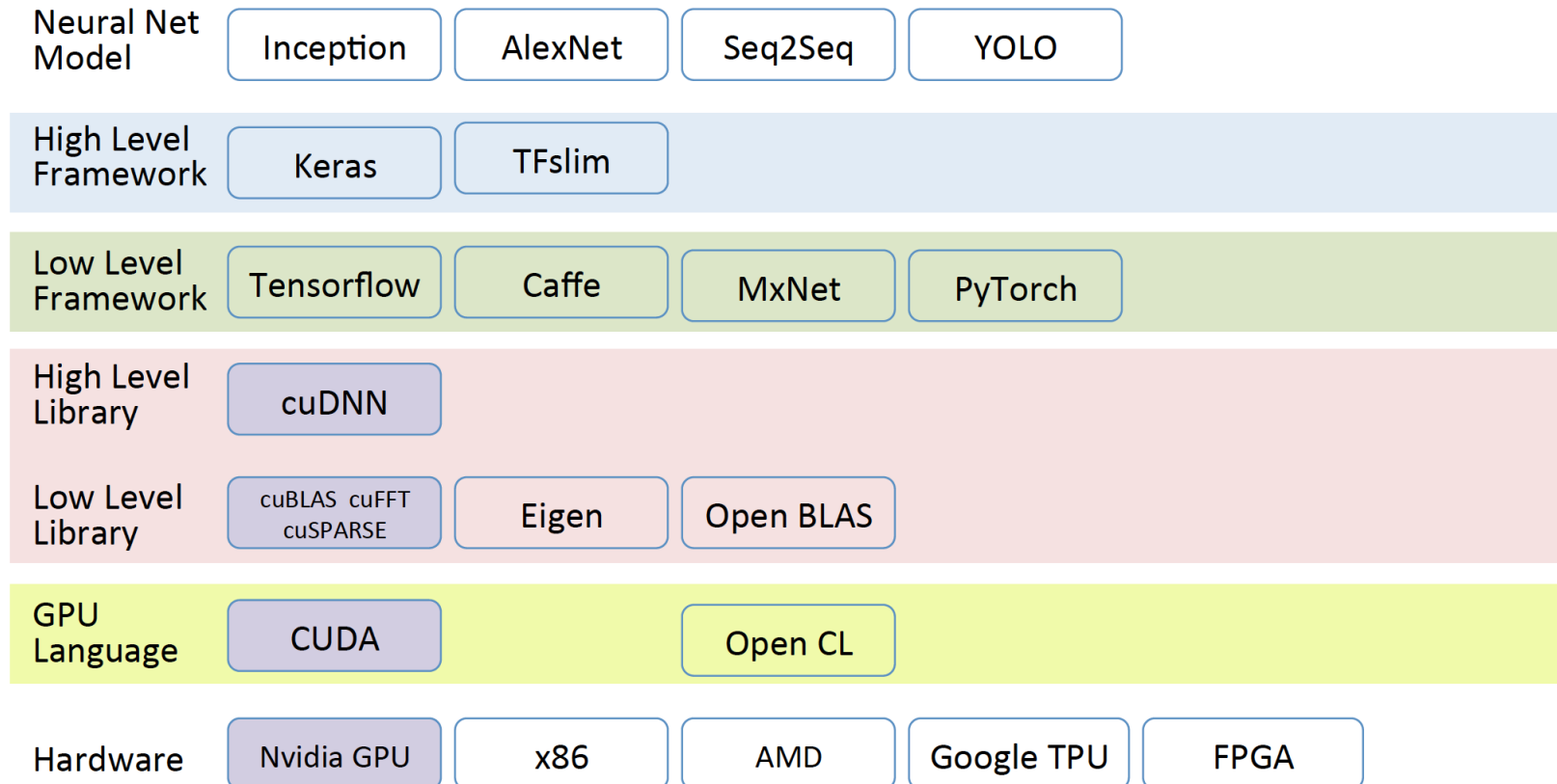
```
__global__  
void add(int n, float *x, float *y)  
{  
    int index = threadIdx.x;  
    int stride = blockDim.x;  
    for (int i = index; i < n; i += stride)  
        y[i] = x[i] + y[i];  
}
```

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



多层抽象 - 编程框架与库

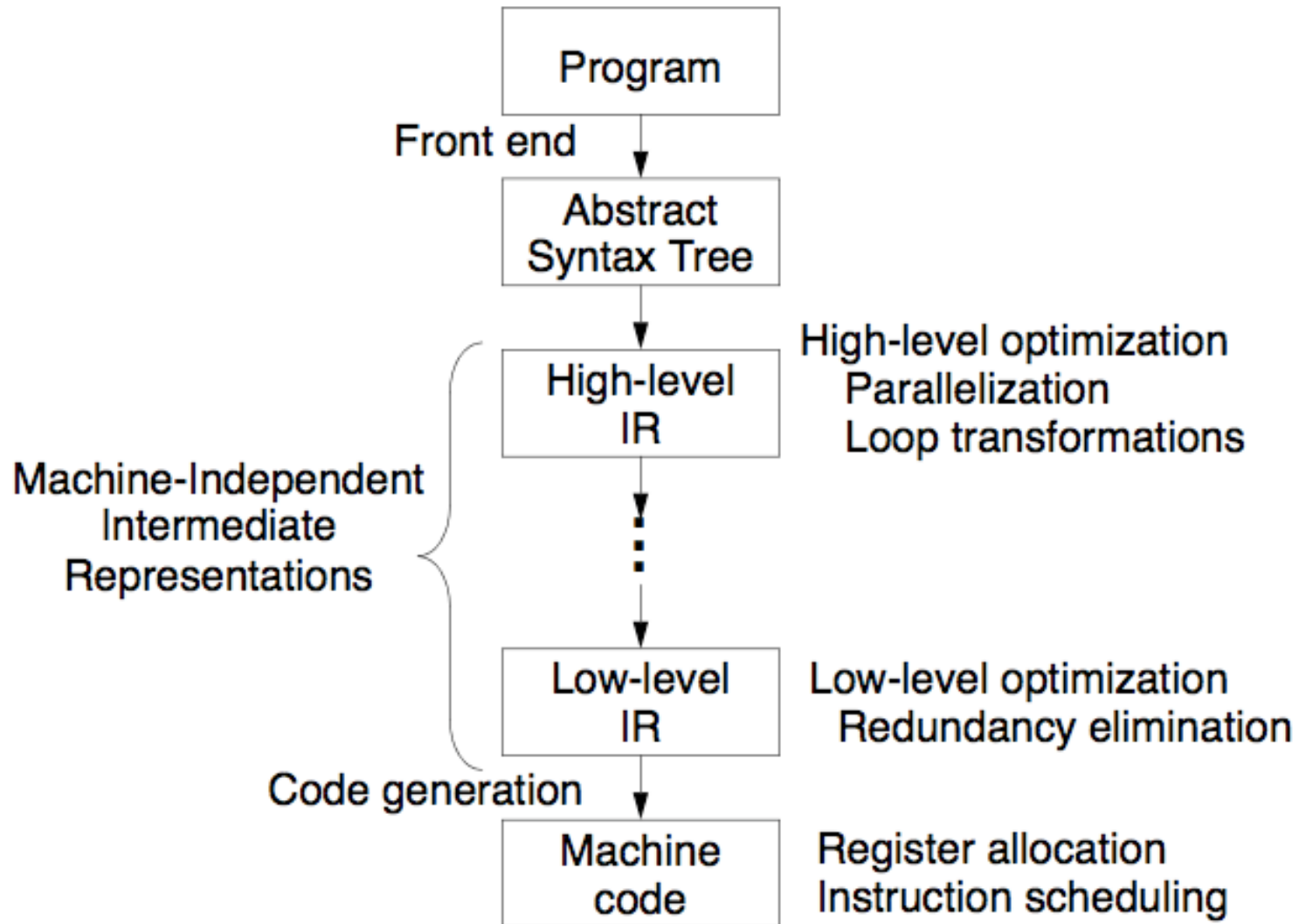
Levels of Abtraction



编译系统的作用

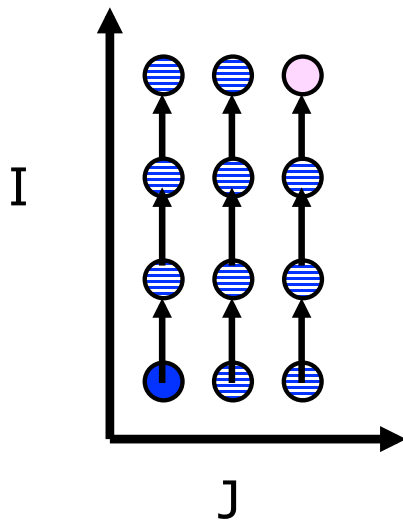
- 翻译
 - 支持高层的编程抽象
 - 支持底层的硬件体系结构
- 优化
 - 更快的执行速度
 - 更小的空间
- 理解程序
 - 安全性(security)
 - 功能正确(safety)

优化编译器的结构



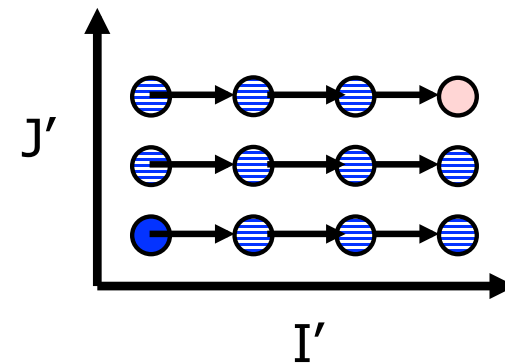
高层循环变换 - 循环交换

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



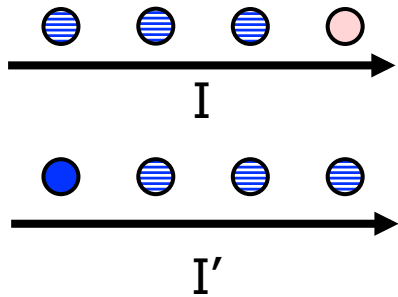
$$\begin{bmatrix} j' \\ i' \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} i \\ j \end{bmatrix}$$

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



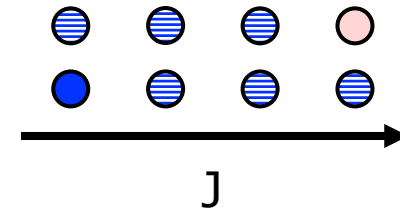
高层循环变换 - 循环合并

```
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)
```



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

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

callq   0x4e
```

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
```

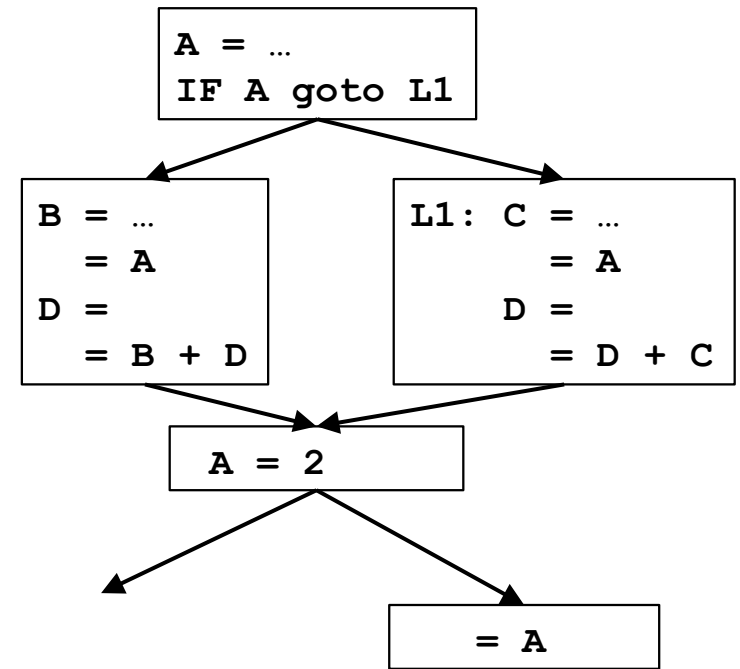
体系结构相关的优化

- 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



Functional units

| | ld | st | alu | fmpy | fadd | br | ... |
|---|----|----|-----|------|------|----|-----|
| 0 | | | | | | | |
| 1 | | | | | | | |
| 2 | | | | | | | |

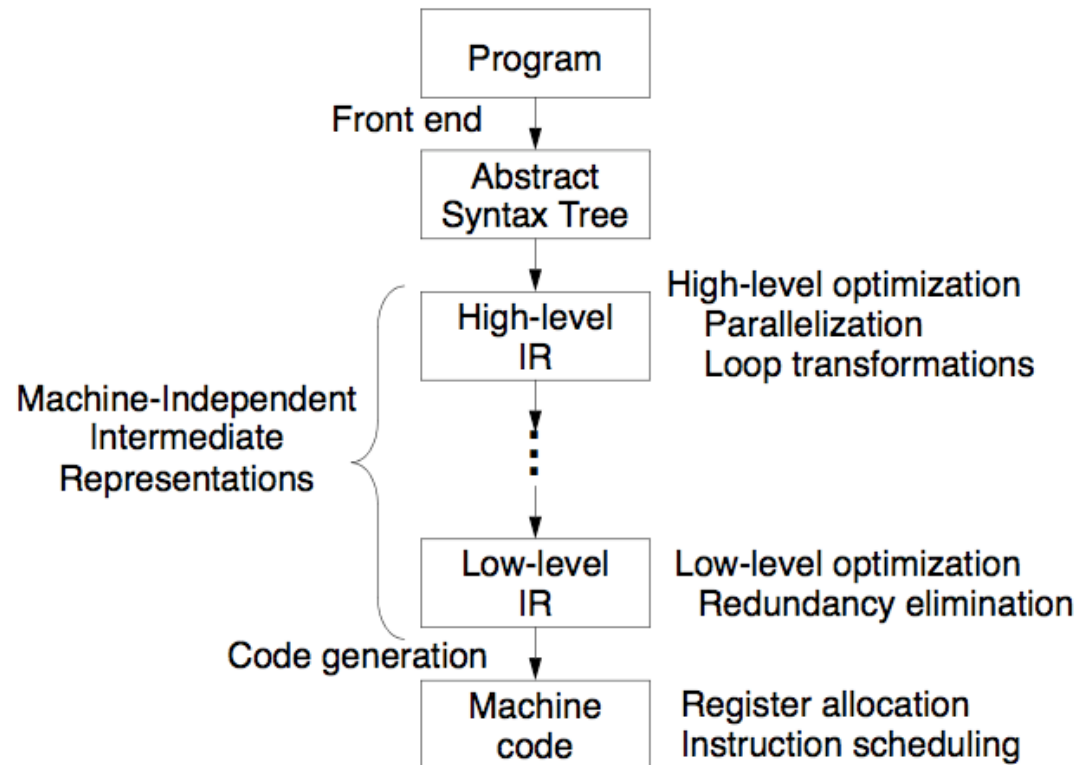
程序优化的现状

- 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

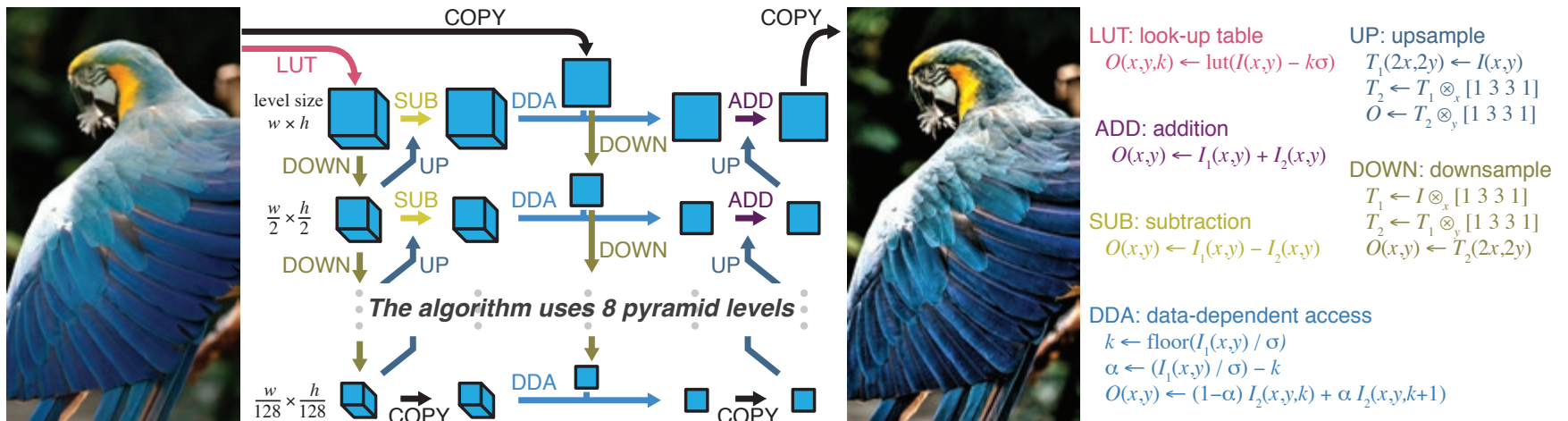
- 消除冗余计算的循环不变量外提可能引起寄存器压力大，从而引发spill，在GPU上开销很大

```
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)

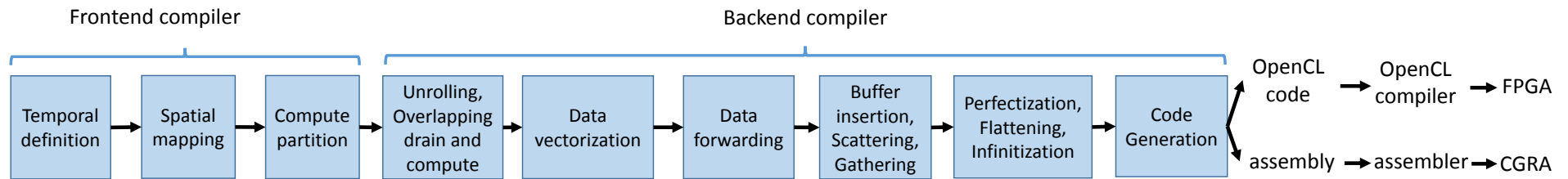
- Halide: 面向图像处理的DSL



```
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 $a+b > 5$ ）
- Reachability
 - 程序抽象成图，程序分析问题建模为图上的可达性问题

例子：指针分析 – 初始建模

```
void main() {  
  x = new C();  
  y = new C();  
  z = new C();  
  m(x,y);  
  n(z,x);  
  q = z.f;  
}
```

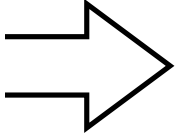
```
void m(C a, C b) {  
  n(a,b);  
}
```

```
void n(C c, C d) {  
  c.f = d;  
}
```

```
vP0('x', 'main@1').  
vP0('y', 'main@2').  
vP0('z', 'main@3').  
assign('a', 'x').  
assign('b', 'y').  
assign('c', 'z').  
assign('d', 'x').  
load('z', 'f', 'q').  
assign('c', 'a').  
assign('d', 'b').  
store('c', 'f', 'd').
```

不断实施规则直到收敛

Rules

 $vP(v,h) :- vP_0(v,h).$
 $vP(v_1,h) :- \text{assign}(v_1,v_2), vP(v_2,h).$
 $hP(h_1,f,h_2) :- \text{store}(v_1,f,v_2), vP(v_1,h_1), vP(v_2,h_2).$
 $vP(v_2,h_2) :- \text{load}(v_1,f,v_2), vP(v_1,h_1), hP(h_1,f,h_2).$

Relations

vP_0

$vP_0('x','main@1').$

$vP_0('y','main@2').$

$vP_0('z','main@3').$

store

$\text{store}('c','f','d').$

load

$\text{load}('z','f','q').$

assign

$\text{assign}('a','x').$

$\text{assign}('b','y').$

$\text{assign}('c','z').$

$\text{assign}('d','x').$

$\text{assign}('c','a').$

$\text{assign}('d','b').$

vP

$vP('x','main@1').$

$vP('y','main@2').$

$vP('z','main@3').$

$vP('a','main@1').$

$vP('d','main@1').$

$vP('b','main@2').$

$vP('c','main@3').$

$vP('c','main@1').$

$vP('d','main@2').$

hP

$hP('main@1','f','main@1').$

$hP('main@1','f','main@2').$

$hP('main@3','f','main@1').$

$hP('main@3','f','main@2').$

程序分析的局限

- **精度和复杂度的矛盾**
 - 区分c1,c2,c3更精确,但过于复杂
 - 不区分则在f处无法将v作为常数

c1:

c2:

c3:

c4:

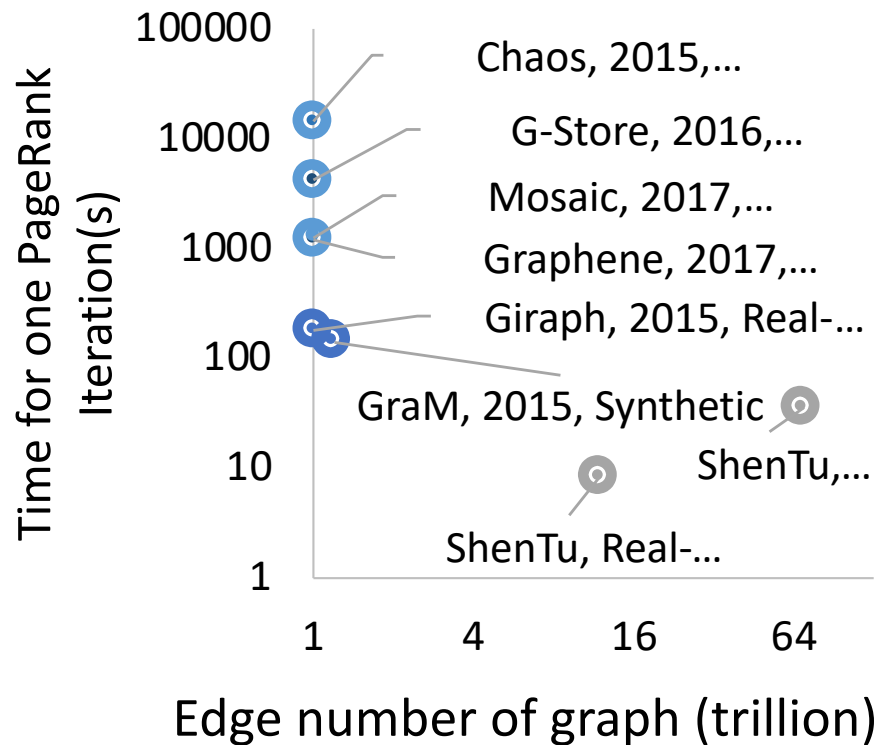
```
for (i = 0; i < n; i++) {  
    t1 = g(0);  
    t2 = g(243);  
    t3 = g(243);  
}
```

```
int g(int v) {  
    return f(v);  
}
```

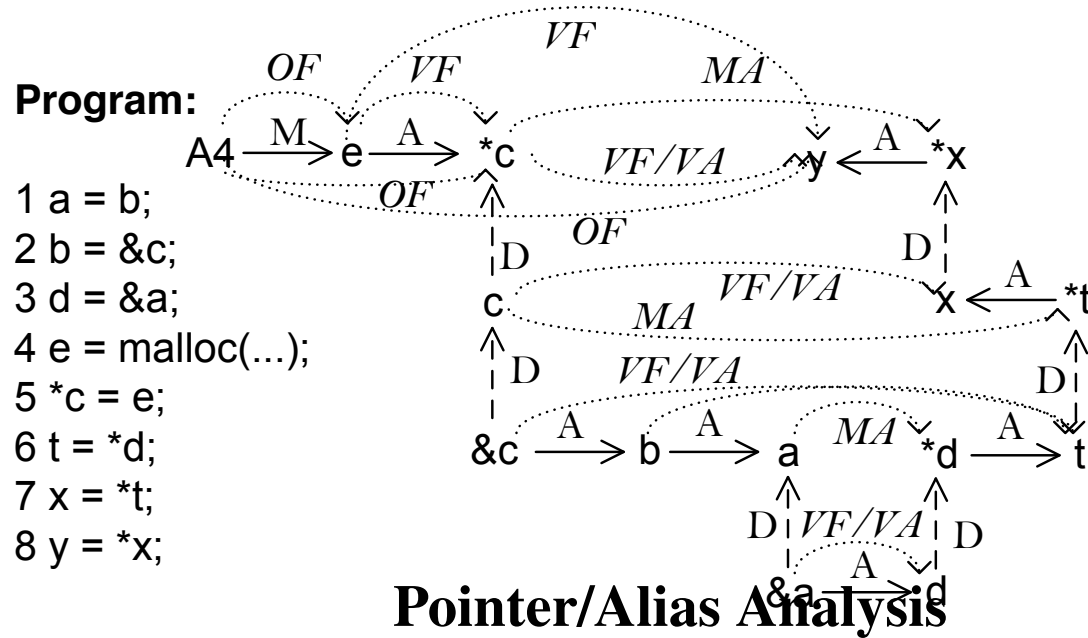
```
int f(int v){  
    return v+1;  
}
```

利用图计算系统的扩展性解决程序分析的规模问题

- 图计算系统可以处理巨大规模的图(千亿结点, 十万亿边)



利用图计算系统进行高精度分析



| Prog | Pointer/Alias Analysis | | | | |
|-------|------------------------|----------------|---------|----|---------|
| | IS=(E,V) | PS=(E,V) | PT | SS | T |
| Linux | (249.5M,52.9M) | (1.1B,52.9M) | 91 secs | 27 | 1.7 hrs |
| PSQL | (25.0M,5.2M) | (862.2M,5.2M) | 10 secs | 16 | 6.0 hrs |
| httpd | (8.2M, 1.7M) | (904.3M, 1.7M) | 3 secs | 13 | 8.4 hrs |

***Graspan: A single-machine, disk-based graph system for interprocedural static analyses of large-scale systems code, ASPLOS 17**

总结

- **编译器是链接程序员和机器的桥梁**
 - 支持更有效的抽象 – 提高开发效率
 - 支持更有效的优化 – 提高运行效率
 - 支持更完善的分析 – 为系统提供安全性
- **编译器教学面临的挑战**
 - 内容少：基本上只介绍了一部分翻译功能
 - 深度不足：如面向对象、函数语言的编译