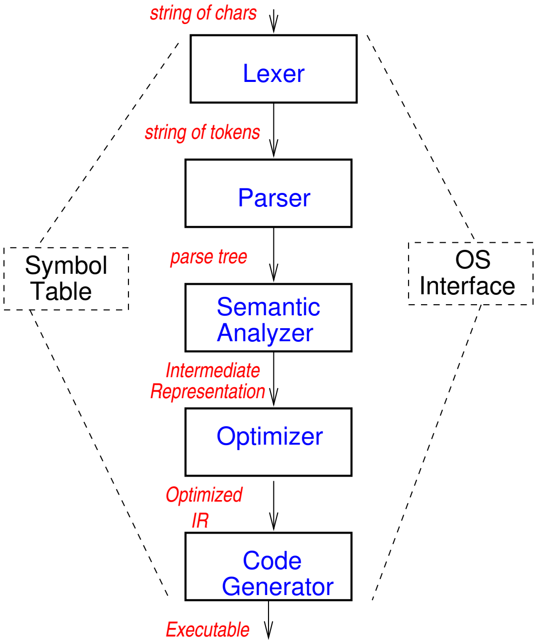
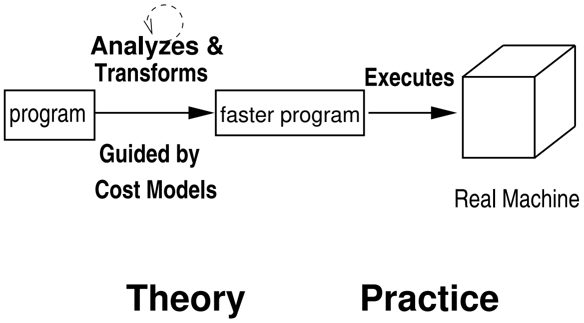
High-Level Compiler Structure



Optimizing Compilers



Why is Optimization needed?

X := Y + 3

temp1 := inttoreal(3)
load(temp2, Y)
temp3 := temp2 + temp1
store(X, temp3)

Z := X + 5

temp4 :=inttoreal(5)
load(temp5,X)
temp6:= temp5 + temp4
store(Z,temp6)

X already in temp3, so needn't reload!

Not because of dumb code!

Different operations (loads, addressing)
exposed at different levels

When is Optimization needed?

High-level
Close to source

Loop structure evident, e.g., CFG

Ex. Loop Fusion, Distribution

Medium-level
Language- and machine independent

Registers, temporaries exposed

Ex. Constant propagation, value numbering

Low-level
Close to machine

May be actual machine instructions

Ex. Register allocation, Scheduling

Link-time

Library routines available

Ex. Interprocedural register allocation

What about optimization order?

Theory	Practice
Transformations interact	Pick "right" order & hope for the best
	Architectural complexity of machines
Take interactions into account	

Theory: Transformations Interact

Constant Propagation *enables*

Dead Code Elimination

Ex. A = 4

if A < 17 then ...

Can eliminate test!

Theory: Transformations Interact

Loop Distribution

```
Do i = 1, 100              Do i = 1, 100
    a(i) = b(i) + c(i)         a(i) = b(i) + c(i)
    d(i) = e(i) * 2        Enddo
Enddo                      Do i = 1, 100
                               d(i) = e(i) * 2
                           Enddo
```

Loop Fusion

7

Theory: Transformations Interact

Instruction scheduling *interferes with* Register allocation

Ex. **A =**	**R1 =**
= A	**= R1**
B =	**R1 =**
= B	**= R1**
Instruction level parallelism	~~Instruction level parallelism~~
~~Reuse of registers~~	Reuse of registers

Integration: [Norris, Pollock], [Bradlee, Eggers, Henry]

8

Practice : Transformation Order

Pick order $T_1 \rightarrow T_2 \rightarrow \ldots\ldots \rightarrow T_k$

Ex.

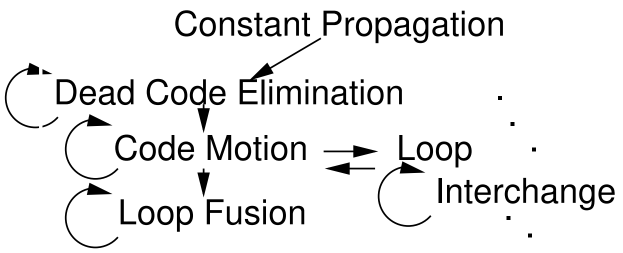
Enabling:

Constant Propagation $\longrightarrow$

Dead Code Elimination

Theory : Transformation Order

Partial Order: **[Whitfield, Soffa]**



General case: Guidance needed!

Guidance in Practice

Separate architectural features
optimized separately
using independent cost functions

Ex.

Parallelism → Locality — **SUIF [Lam et al]**

Locality → Parallelism — **[McKinley et al]**

Guidance in Practice

Single compiler for multiple architectures
needs architecture-based guidance

Ex.

Coarse-grain Parallelism :
Use Loop Fusion

Vector processor:
Use Loop Distribution

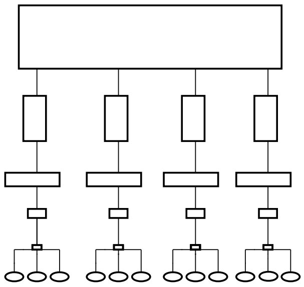
Problem with Practice

Current architectures:

Deep memory hierarchies

Multiple levels of parallelism

Create multiple bottlenecks!



Murphy's Law of Memory Hierarchies

Hiearchical bottlenecks *interact*

Independent solutions not enough!

Back to Theory

Our work: ***(more later...)***

Study how *tiling* transformations *interact*