

# Compiler Optimisation

## 5 – Instruction Selection

Hugh Leather

IF 1.18a

hleather@inf.ed.ac.uk

Institute for Computing Systems Architecture

School of Informatics

University of Edinburgh

2018

# Introduction

This lecture:

- Naive translation and ILOC
- Cost based instruction selection
- Bottom up tiling on low level AST
- Alternative approach based on peephole optimisation
- Super-optimisation
- Multimedia code generation

# Code generation

- Aim to generate the most efficient assembly code
- Decouple problem into three phases:
  - Instruction selection
  - Instruction scheduling
  - Register allocation
- In general phases NP-complete and strongly interact
- In practise good solutions can be found
- Instruction scheduling : would like to automate wherever possible – re-targetable ISA specific translation rules plus generic optimiser

# ILOC

## Instruction set review

### Typical ILOC instructions (EaC Appendix A)

load	$r_1$	$\Rightarrow r_2$	$r_2 = \text{Mem}[ r_1 ]$
loadI	$c_1$	$\Rightarrow r_1$	$r_1 = c_1$
loadAI	$r_1, c_1$	$\Rightarrow r_2$	$r_2 = \text{Mem}[ r_1 + c_1 ]$
loadA0	$r_1, r_2$	$\Rightarrow r_3$	$r_3 = \text{Mem}[ r_1 + r_2 ]$
store	$r_1$	$\Rightarrow r_2$	$\text{Mem}[ r_2 ] = r_1$
storeAI	$r_1$	$\Rightarrow r_2, c_1$	$\text{Mem}[ r_2 + c_1 ] = r_1$
storeA0	$r_1$	$\Rightarrow r_2, r_3$	$\text{Mem}[ r_2 + r_3 ] = r_1$
i2i	$r_1$	$\Rightarrow r_2$	$r_2 = r_1$
add	$r_1, r_2$	$\Rightarrow r_3$	$r_3 = r_1 + r_2$
addI	$r_1, c_1$	$\Rightarrow r_2$	$r_2 = r_1 + c_1$
Similar for arithmetic, logical, and shifts			
jump		$r_1$	$\text{PC} = r_1$
jumpI		$l_1$	$\text{PC} = l_1$
cbr	$r_1$	$\Rightarrow l_1, l_2$	$\text{PC} = r_1 ? l_1 : l_2$

# ILOC

- Many ways to do the same thing
- If operators assigned to distinct functional units - big impact

## Different ways to move register, $r_i \Rightarrow r_j$

i2i	$r_i$	$\Rightarrow r_j$
addI	$r_i, 0$	$\Rightarrow r_j$
subI	$r_i, 0$	$\Rightarrow r_j$
multI	$r_i, 1$	$\Rightarrow r_j$
divI	$r_i, 1$	$\Rightarrow r_j$
lshiftI	$r_i, 0$	$\Rightarrow r_j$
rshiftI	$r_i, 0$	$\Rightarrow r_j$
and	$r_i, r_i$	$\Rightarrow r_j$
orI	$r_i, 0$	$\Rightarrow r_j$
xorI	$r_i, 0$	$\Rightarrow r_j$

# ILOC

## Naïve selection

- Simple walk through of first lecture generates inefficient code
- Takes a naive view of location of data and does not exploit different addressing modes available

### Different code to compute $g * h$

Assume  $g$  and  $h$  in global spaces  $G$  and  $H$ , both at offset 4

loadI	@G	$\Rightarrow r_5$		
loadI	4	$\Rightarrow r_6$		
loadAO	$r_5, r_6$	$\Rightarrow r_7$	loadI	4 $\Rightarrow r_5$
loadI	@H	$\Rightarrow r_8$	loadAI	$r_5, @G \Rightarrow r_6$
loadI	4	$\Rightarrow r_9$	loadAI	$r_5, @H \Rightarrow r_7$
loadAO	$r_8, r_9$	$\Rightarrow r_{10}$	mult	$r_6, r_7 \Rightarrow r_8$
mult	$r_7, r_{10}$	$\Rightarrow r_{11}$		

# Instruction selection via tree pattern matching

- IR is in low level AST form exposing storage type of operands
- Tile AST with operation trees generating  $\langle ast, op \rangle$  i.e.  $op$  could implement abstract syntax tree  $ast$
- Recursively tile tree and bottom-up select the cheapest tiling - locally optimal.
- Overlaps of trees must match
  - destination of one tree is the source of another
  - must agree on storage location and type - register or memory, int or float, etc
- Operations are connected to AST subtrees by a set of *ambiguous* rewrite rules
- Rules have costs - ambiguity allows cost based choice

# Instruction selection via tree pattern matching

## Rewrite rules

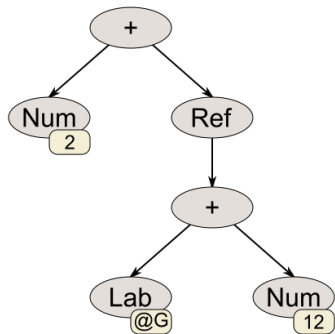
### Subset of rules

Id	Production	Code Template
1:	$Reg \rightarrow Lab$	loadI $ bl  \Rightarrow r_{new}$
2:	$Reg \rightarrow Num$	loadI $n_1 \Rightarrow r_{new}$
3:	$Reg \rightarrow Ref(Reg)$	load $r_1 \Rightarrow r_{new}$
4:	$Reg \rightarrow Ref(+ (Reg_1, Reg_2))$	loadA0 $r_1, r_2 \Rightarrow r_{new}$
5:	$Reg \rightarrow Ref(+ (Reg, Num))$	loadAI $r_1, n_1 \Rightarrow r_{new}$
6:	$Reg \rightarrow + (Reg_1, Reg_2)$	add $r_1, r_2 \Rightarrow r_{new}$
7:	$Reg \rightarrow + (Reg, Num)$	addI $r_1, n_1 \Rightarrow r_{new}$
8:	$Reg \rightarrow + (Num, Reg)$	addI $r_1, n_1 \Rightarrow r_{new}$



# Instruction selection via tree pattern matching

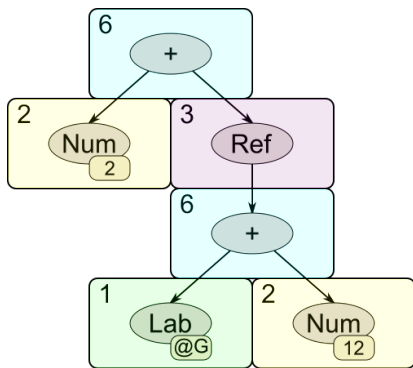
## Rewrite rules



Begin tiling the AST bottom up

# Instruction selection via tree pattern matching

## Rewrite rules



## Code produced

```
loadI @G    => r1
loadI 12    => r2
  add  r1, r2 => r3
  load r3    => r4
loadI 2     => r5
  add  r4, r5 => r6
```

---

### Bad tiling: productions used

1: $Reg \rightarrow Lab$	$loadI\ lbl \Rightarrow r_{new}$
2: $Reg \rightarrow Num$	$loadI\ n_1 \Rightarrow r_{new}$
3: $Reg \rightarrow Ref(Reg)$	$load\ r_1 \Rightarrow r_{new}$
6: $Reg \rightarrow +(Reg_1, Reg_2)$	$add\ r_1, r_2 \Rightarrow r_{new}$

---

# Instruction selection via tree pattern matching

## Rewrite rules

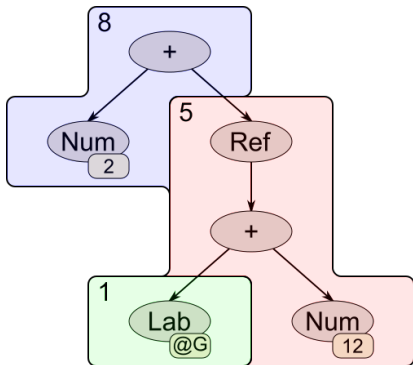
- Many different sequences available
- Selecting lowest cost bottom-up gives

### Code produced

```
loadI  @G    ⇒ r1
loadAI r1, 12 ⇒ r2
addI   r2, 2  ⇒ r3
```

Good tiling: productions used

1: $Reg \rightarrow Lab$	$loadI\  b  \Rightarrow r_{new}$
5: $Reg \rightarrow Ref(+ (Reg, Num))$	$loadAI\ r_1, n_1 \Rightarrow r_{new}$
8: $Reg \rightarrow + (Num, Reg)$	$addI\ r_1, n_1 \Rightarrow r_{new}$



# Instruction selection via tree pattern matching

## Cost based selection

- Examples assume all operations are equal cost
- Certain ops may be more expensive - divs
- Cost of bottom matching can be reduced using table lookups

# Peephole selection

- Other approaches available - peephole optimisation
  - Expand code into operations below machine level
  - Simplify by rules over sliding window
  - Match against machine instructions

## Peephole instruction selection

Selection for:  $b - 2 * c$

$r_{10}$	$\leftarrow$	2
$r_{11}$	$\leftarrow$	@G
$r_{12}$	$\leftarrow$	12
$r_{13}$	$\leftarrow$	$r_{11} + r_{12}$
$r_{14}$	$\leftarrow$	$M(r_{13})$
$r_{15}$	$\leftarrow$	$r_{10} \times r_{14}$
$r_{16}$	$\leftarrow$	-16
$r_{17}$	$\leftarrow$	$r_{arp} + r_{16}$
$r_{18}$	$\leftarrow$	$M(r_{17})$
$r_{19}$	$\leftarrow$	$M(r_{18})$
$r_{20}$	$\leftarrow$	$r_{19} - r_{15}$
$r_{21}$	$\leftarrow$	4
$r_{22}$	$\leftarrow$	$r_{arp} + r_{21}$
$M(r_{22})$	$\leftarrow$	$r_{20}$

Elaborate into very low-level code

## Peephole instruction selection

Selection for:  $b - 2 * c$

$r_{10}$	$\leftarrow$	2
$r_{11}$	$\leftarrow$	@G
$r_{12}$	$\leftarrow$	12
$r_{13}$	$\leftarrow$	$r_{11} + r_{12}$
$r_{14}$	$\leftarrow$	$M(r_{13})$
$r_{15}$	$\leftarrow$	$r_{10} \times r_{14}$
$r_{16}$	$\leftarrow$	-16
$r_{17}$	$\leftarrow$	$r_{arp} + r_{16}$
$r_{18}$	$\leftarrow$	$M(r_{17})$
$r_{19}$	$\leftarrow$	$M(r_{18})$
$r_{20}$	$\leftarrow$	$r_{19} - r_{15}$
$r_{21}$	$\leftarrow$	4
$r_{22}$	$\leftarrow$	$r_{arp} + r_{21}$
$M(r_{22})$	$\leftarrow$	$r_{20}$

First window, no simplification available; advance window

## Peephole instruction selection

Selection for:  $b - 2 * c$

$r_{10}$	$\leftarrow$	2
$r_{11}$	$\leftarrow$	@G
$r_{12}$	$\leftarrow$	12
$r_{13}$	$\leftarrow$	$r_{11} + r_{12}$
$r_{14}$	$\leftarrow$	$M(r_{13})$
$r_{15}$	$\leftarrow$	$r_{10} \times r_{14}$
$r_{16}$	$\leftarrow$	-16
$r_{17}$	$\leftarrow$	$r_{arp} + r_{16}$
$r_{18}$	$\leftarrow$	$M(r_{17})$
$r_{19}$	$\leftarrow$	$M(r_{18})$
$r_{20}$	$\leftarrow$	$r_{19} - r_{15}$
$r_{21}$	$\leftarrow$	4
$r_{22}$	$\leftarrow$	$r_{arp} + r_{21}$
$M(r_{22})$	$\leftarrow$	$r_{20}$

Substitute  $r_{12}$  into  $r_{13}$ ;  $r_{12}$  dead so remove



## Peephole instruction selection

Selection for:  $b - 2 * c$

$r_{10}$	$\leftarrow$	2
$r_{11}$	$\leftarrow$	@G
$r_{13}$	$\leftarrow$	$r_{11} + 12$
$r_{14}$	$\leftarrow$	$M(r_{13})$
$r_{15}$	$\leftarrow$	$r_{10} \times r_{14}$
$r_{16}$	$\leftarrow$	-16
$r_{17}$	$\leftarrow$	$r_{arp} + r_{16}$
$r_{18}$	$\leftarrow$	$M(r_{17})$
$r_{19}$	$\leftarrow$	$M(r_{18})$
$r_{20}$	$\leftarrow$	$r_{19} - r_{15}$
$r_{21}$	$\leftarrow$	4
$r_{22}$	$\leftarrow$	$r_{arp} + r_{21}$
$M(r_{22})$	$\leftarrow$	$r_{20}$

Substitute  $r_{13}$  into  $r_{14}$ ;  $r_{13}$  dead so remove

## Peephole instruction selection

Selection for:  $b - 2 * c$

$r_{10}$	$\leftarrow$	2
$r_{11}$	$\leftarrow$	@G
$r_{14}$	$\leftarrow$	$M(r_{11} + 12)$
$r_{15}$	$\leftarrow$	$r_{10} \times r_{14}$
$r_{16}$	$\leftarrow$	-16
$r_{17}$	$\leftarrow$	$r_{arp} + r_{16}$
$r_{18}$	$\leftarrow$	$M(r_{17})$
$r_{19}$	$\leftarrow$	$M(r_{18})$
$r_{20}$	$\leftarrow$	$r_{19} - r_{15}$
$r_{21}$	$\leftarrow$	4
$r_{22}$	$\leftarrow$	$r_{arp} + r_{21}$
$M(r_{22})$	$\leftarrow$	$r_{20}$

No simplification available; advance window

## Peephole instruction selection

Selection for:  $b - 2 * c$

$r_{10}$	$\leftarrow$	2
$r_{11}$	$\leftarrow$	@G
$r_{14}$	$\leftarrow$	$M(r_{11} + 12)$
$r_{15}$	$\leftarrow$	$r_{10} \times r_{14}$
$r_{16}$	$\leftarrow$	-16
$r_{17}$	$\leftarrow$	$r_{arp} + r_{16}$
$r_{18}$	$\leftarrow$	$M(r_{17})$
$r_{19}$	$\leftarrow$	$M(r_{18})$
$r_{20}$	$\leftarrow$	$r_{19} - r_{15}$
$r_{21}$	$\leftarrow$	4
$r_{22}$	$\leftarrow$	$r_{arp} + r_{21}$
$M(r_{22})$	$\leftarrow$	$r_{20}$

No simplification available; advance window

## Peephole instruction selection

Selection for:  $b - 2 * c$

$r_{10}$	$\leftarrow$	2
$r_{11}$	$\leftarrow$	@G
$r_{14}$	$\leftarrow$	$M(r_{11} + 12)$
$r_{15}$	$\leftarrow$	$r_{10} \times r_{14}$
$r_{16}$	$\leftarrow$	-16
$r_{17}$	$\leftarrow$	$r_{arp} + r_{16}$
$r_{18}$	$\leftarrow$	$M(r_{17})$
$r_{19}$	$\leftarrow$	$M(r_{18})$
$r_{20}$	$\leftarrow$	$r_{19} - r_{15}$
$r_{21}$	$\leftarrow$	4
$r_{22}$	$\leftarrow$	$r_{arp} + r_{21}$
$M(r_{22})$	$\leftarrow$	$r_{20}$

Substitute  $r_{16}$  into  $r_{17}$ ;  $r_{16}$  dead so remove

## Peephole instruction selection

Selection for:  $b - 2 * c$

$r_{10}$	$\leftarrow$	2
$r_{11}$	$\leftarrow$	@G
$r_{14}$	$\leftarrow$	$M(r_{11} + 12)$
$r_{15}$	$\leftarrow$	$r_{10} \times r_{14}$
$r_{17}$	$\leftarrow$	$r_{arp} - 16$
$r_{18}$	$\leftarrow$	$M(r_{17})$
$r_{19}$	$\leftarrow$	$M(r_{18})$
$r_{20}$	$\leftarrow$	$r_{19} - r_{15}$
$r_{21}$	$\leftarrow$	4
$r_{22}$	$\leftarrow$	$r_{arp} + r_{21}$
$M(r_{22})$	$\leftarrow$	$r_{20}$

Substitute  $r_{17}$  into  $r_{18}$ ;  $r_{17}$  dead so remove

## Peephole instruction selection

Selection for:  $b - 2 * c$

$r_{10}$	$\leftarrow$	2
$r_{11}$	$\leftarrow$	@G
$r_{14}$	$\leftarrow$	$M(r_{11} + 12)$
$r_{15}$	$\leftarrow$	$r_{10} \times r_{14}$
$r_{18}$	$\leftarrow$	$M(r_{arp} - 16)$
$r_{19}$	$\leftarrow$	$M(r_{18})$
$r_{20}$	$\leftarrow$	$r_{19} - r_{15}$
$r_{21}$	$\leftarrow$	4
$r_{22}$	$\leftarrow$	$r_{arp} + r_{21}$
$M(r_{22})$	$\leftarrow$	$r_{20}$

No simplification available; advance window

## Peephole instruction selection

Selection for:  $b - 2 * c$

$r_{10}$	$\leftarrow$	2
$r_{11}$	$\leftarrow$	@G
$r_{14}$	$\leftarrow$	$M(r_{11} + 12)$
$r_{15}$	$\leftarrow$	$r_{10} \times r_{14}$
$r_{18}$	$\leftarrow$	$M(r_{arp} - 16)$
$r_{19}$	$\leftarrow$	$M(r_{18})$
$r_{20}$	$\leftarrow$	$r_{19} - r_{15}$
$r_{21}$	$\leftarrow$	4
$r_{22}$	$\leftarrow$	$r_{arp} + r_{21}$
$M(r_{22})$	$\leftarrow$	$r_{20}$

No simplification available; advance window

## Peephole instruction selection

Selection for:  $b - 2 * c$

$r_{10}$	$\leftarrow$	2
$r_{11}$	$\leftarrow$	@G
$r_{14}$	$\leftarrow$	$M(r_{11} + 12)$
$r_{15}$	$\leftarrow$	$r_{10} \times r_{14}$
$r_{18}$	$\leftarrow$	$M(r_{arp} - 16)$
$r_{19}$	$\leftarrow$	$M(r_{18})$
$r_{20}$	$\leftarrow$	$r_{19} - r_{15}$
$r_{21}$	$\leftarrow$	4
$r_{22}$	$\leftarrow$	$r_{arp} + r_{21}$
$M(r_{22})$	$\leftarrow$	$r_{20}$

No simplification available; advance window



## Peephole instruction selection

Selection for:  $b - 2 * c$

$r_{10}$	$\leftarrow$	2
$r_{11}$	$\leftarrow$	@G
$r_{14}$	$\leftarrow$	$M(r_{11} + 12)$
$r_{15}$	$\leftarrow$	$r_{10} \times r_{14}$
$r_{18}$	$\leftarrow$	$M(r_{arp} - 16)$
$r_{19}$	$\leftarrow$	$M(r_{18})$
$r_{20}$	$\leftarrow$	$r_{19} - r_{15}$
$r_{21}$	$\leftarrow$	4
$r_{22}$	$\leftarrow$	$r_{arp} + r_{21}$
$M(r_{22})$	$\leftarrow$	$r_{20}$

Substitute  $r_{21}$  into  $r_{22}$ ;  $r_{21}$  dead so remove

## Peephole instruction selection

Selection for:  $b - 2 * c$

$r_{10}$	$\leftarrow$	2
$r_{11}$	$\leftarrow$	@G
$r_{14}$	$\leftarrow$	$M(r_{11} + 12)$
$r_{15}$	$\leftarrow$	$r_{10} \times r_{14}$
$r_{18}$	$\leftarrow$	$M(r_{arp} - 16)$
$r_{19}$	$\leftarrow$	$M(r_{18})$
$r_{20}$	$\leftarrow$	$r_{19} - r_{15}$
$r_{22}$	$\leftarrow$	$r_{arp} + 4$
$M(r_{22})$	$\leftarrow$	$r_{20}$

Substitute  $r_{22}$  into  $M(r_{22})$ ;  $r_{22}$  dead so remove

## Peephole instruction selection

Selection for:  $b - 2 * c$

$r_{10}$	$\leftarrow$	2
$r_{11}$	$\leftarrow$	@G
$r_{14}$	$\leftarrow$	$M(r_{11} + 12)$
$r_{15}$	$\leftarrow$	$r_{10} \times r_{14}$
$r_{18}$	$\leftarrow$	$M(r_{arp} - 16)$
$r_{19}$	$\leftarrow$	$M(r_{18})$
$r_{20}$	$\leftarrow$	$r_{19} - r_{15}$
$M(r_{arp} + 4)$	$\leftarrow$	$r_{20}$

No more code to bring into window

## Peephole instruction selection

Selection for:  $b - 2 * c$

$r_{10}$	$\leftarrow$	2
$r_{11}$	$\leftarrow$	@G
$r_{14}$	$\leftarrow$	$M(r_{11} + 12)$
$r_{15}$	$\leftarrow$	$r_{10} \times r_{14}$
$r_{18}$	$\leftarrow$	$M(r_{arp} - 16)$
$r_{19}$	$\leftarrow$	$M(r_{18})$
$r_{20}$	$\leftarrow$	$r_{19} - r_{15}$
$M(r_{arp} + 4)$	$\leftarrow$	$r_{20}$

Simplified code is 8 instructions versus 14

## Peephole instruction selection

Selection for:  $b - 2 * c$

loadl	2	$\Rightarrow$	$r_{10}$
loadl	@G	$\Rightarrow$	$r_{11}$
loadAl	$r_{11} + 12$	$\Rightarrow$	$r_{14}$
mult	$r_{10}, r_{14}$	$\Rightarrow$	$r_{15}$
loadAl	$r_{arp}, -16$	$\Rightarrow$	$r_{18}$
load	$r_{18}$	$\Rightarrow$	$r_{19}$
sub	$r_{19}, r_{15}$	$\Rightarrow$	$r_{20}$
storeAl	$r_{20}$	$\Rightarrow$	$r_{arp}, 4$

Match against machine instructions

# Peephole selection

- Works well with linear IR and gives in practise similar performance
- Sensitive to window size - difficult to argue for optimality
- Needs knowledge of when values are dead
- Has difficulty handling general control-flow

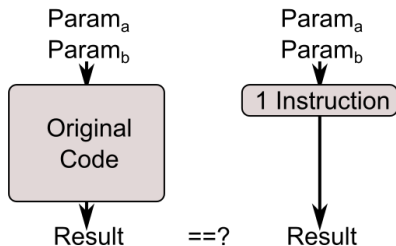
# Super-optimisation

- Super-optimisers *search* for the best instruction sequence
- Generally very slow - minutes, hours, or weeks!
- Only suitable for very small, hot kernels

# Super-optimisation

## Massalin's super-optimiser

- Start with length  $k = 1$
- Generate **all** instruction sequences of length  $k$
- Run test cases to compare behaviour to original code

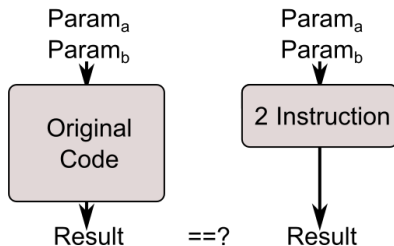




# Super-optimisation

## Massalin's super-optimiser

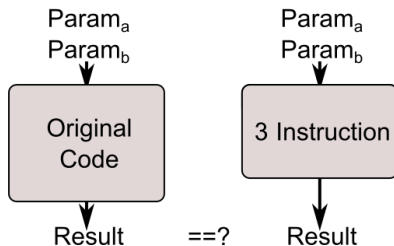
- Start with length  $k = 1$
- Generate **all** instruction sequences of length  $k$
- Run test cases to compare behaviour to original code
- If success, return sequence else increase length



# Super-optimisation

## Massalin's super-optimiser

- Start with length  $k = 1$
- Generate **all** instruction sequences of length  $k$
- Run test cases to compare behaviour to original code
- If success, return sequence else increase length
- **Test cases not correctness guarantee**

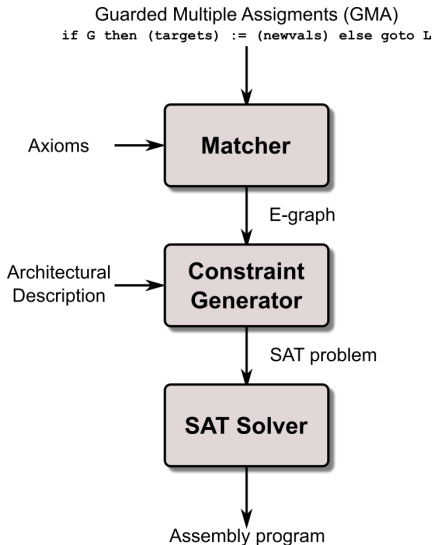


## Denali: A goal directed super-optimiser

- Super-optimiser. Attempt to find optimum code - not just improve.
- “Denali: A goal directed super-optimizer” PLDI 2002 by Joshi, Nelson and Randall. Expect you to read, understand and know this
- Based on theorem proving over all equivalent programs. Basic idea: use a set of axioms which define equivalent instructions
- Generate a data structure representing all possible equivalent programs. Then use a theorem prover to find the shortest sequence
- “There does not exist a program  $k$  cycles or less”. Searches all equivalence to disprove this. Theorem provers designed to be efficient at this type of search

# Denali: A goal directed super-optimiser

## Structure



# Denali: A goal directed super-optimiser

## Axioms

Axioms are a mixture of generic and machine specific for Alpha

- $4 = 2^2$  – generic
- $(\forall k, n :: k * 2^n = k \ll n)$  – machine specific
- $(\forall k, n :: k * 4 + n = \mathbf{s4addl}(k, n))$

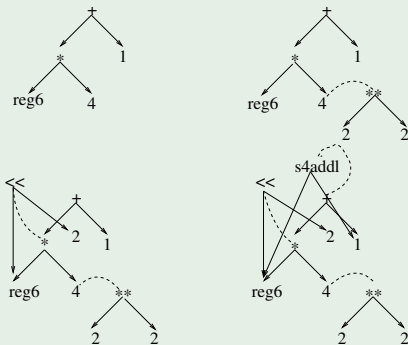
# Denali: A goal directed super-optimiser

E-graph

Equivalences represented in an E-graph.

$O(n)$  graph can represent  $O(2^n)$  distinct ways of computing term

Match expression  $1 + \text{reg6} * 4$



Dashed lines denote equivalences (matches)

# Denali: A goal directed super-optimiser

## Unknowns

Once equivalent programs represented, now need to see if there is a solution in  $K$  cycles.

Unknowns:

- $L(i, T)$  Term  $T$  started at time  $i$
- $A(i, T)$  Term  $T$  finished at time  $i$
- $B(i, Q)$  Equivalence class  $Q$  finished by time  $i$

Need constraints to solve.

Let  $\lambda(T) = \text{latency of term } T$

# Denali: A goal directed super-optimiser

## Constraints

- $\bigwedge_{i,T}(L(i, T) \Leftrightarrow A(i + \lambda(T) - 1, T))$  – arrives  $\lambda$  cycles after being launched
- $\bigwedge_{i,T} \bigwedge_{Q \in \text{args}(T)} (L(i, T) \Rightarrow B(i - 1, Q))$  – operation cannot be launched till args ready
- $\bigwedge_{Q \in G} B(K - 1, Q)$  – all terms in the goal must be finished within  $K$  cycles

Now test with a SAT solver setting  $K$  to a suitable number.

Generates excellent code

Finds best code fast. Approximate memory latency, limited implementation



# Multimedia code

- Re-targetable code generation key issue in embedded processors
- Heterogeneous instruction sets. Restrictions on function units.
- Exploiting powerful multimedia instructions
- Standard Code generation seems completely blind to parallelism. Shorter code may severely restrict ILP
- Denali gets around this but expensive
- Multimedia instructions are often SIMD like. Need parallelisation techniques. Middle section of lectures.

# Summary

- Naive translation and ILOC
- Cost based instruction selection
- Bottom up tiling on low level AST
- Alternative approach based on peephole optimisation
- Super-optimisation
- Multimedia code generation

# PPar CDT Advert

## EPSRC Centre for Doctoral Training in Pervasive Parallelism

- 4-year programme:  
MSc by Research + PhD
- Research-focused:  
Work on your thesis topic  
from the start
- Collaboration between:
  - ▶ University of Edinburgh's  
School of Informatics
    - \* Ranked top in the UK by  
2014 REF
  - ▶ Edinburgh Parallel Computing  
Centre
    - \* UK's largest supercomputing  
centre
- Research topics in software,  
hardware, theory and  
application of:
  - ▶ Parallelism
  - ▶ Concurrency
  - ▶ Distribution
- Full funding available
- Industrial engagement  
programme includes  
internships at leading  
companies

The biggest revolution  
in the technological  
landscape for fifty years

Now accepting applications!  
Find out more and apply at:

[pervasiveparallelism.inf.ed.ac.uk](http://pervasiveparallelism.inf.ed.ac.uk)



THE UNIVERSITY OF EDINBURGH  
**informatics**

**EPSRC**

Engineering and Physical Sciences  
Research Council