Co-degeneration

To pick tips on the register allocation:

place operands in registers
reduce load/store operations

Instruction scheduling
- calculate when each instruction executes
- enable pipelining to hide latency
- calculate number of registers required to evaluate a subtree without storing values to memory
- label each interior node with that number

Phase 1

Phase 2

Overview of algorithm

combines allocation and scheduling
- uses minimum number of registers
- generates z-address code for expression trees
- combines allocation and scheduling
- uses minimum number of registers
- generates z-address code for expression trees

Seth-Ulmen algorithm

Code generation

Topics

Code generation for trees

IR to code using specification
- automate backend construction
- code generation generators
  - functional
  - object-oriented
  - high-level languages
- required on VLIW architectures
- reduce load/store operations
- place operands in registers
/* case 1 - generate left child first */
if {
endif
endif

procedure Encode(u)

REG = current register number (initialized to 1)

emt(Rex, REG, name)

REG = REG - 1

// GENERATE REG

REG = REG + 1

emt(load, REG, name)

// GENERATE REG

/* case 0 - just load it */
if in leaf mode, then

REG = current register number (initialized to 1)

emt(op, REG, REG, REG + 1, REG)

REG = REG - 1

REG = REG + 1

REG = REG - 1

emt(load, REG, name)

REG = current register number (initialized to 1)

emt(op, REG, REG, REG + 1, REG)

REG = REG - 1

REG = REG + 1

// GENERATE REG

if is leaf then

REG = current register number (initialized to 1)

emt(op, REG, REG, REG + 1, REG)

REG = REG - 1

REG = REG + 1

REG = REG - 1

emt(load, REG, name)

REG = current register number (initialized to 1)

emt(op, REG, REG, REG + 1, REG)

REG = REG - 1

REG = REG + 1

// GENERATE REG

if in interior mode, then

(1, u(n)) REG = REG + 1

emt(load, REG, name)

if in leaf mode, then

REG = current register number (initialized to 1)

emt(op, REG, REG, REG + 1, REG)

REG = REG - 1

REG = REG + 1

REG = REG - 1

emt(load, REG, name)

if in leaf mode, then

// GENERATE REG

endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endi
schedule instructions first \(\Rightarrow\) many registers

allocate registers first \(\Rightarrow\) many stalls

Phase ordering problem

will use too many registers

issue all loads, then execute all operators

Naive approach

Apply instruction scheduling

\(\theta\) increases for modern microprocessors

hardware to stall (interlock)

attempts to access target of load early causes

issue load, result appears \(\theta\) cycles later

Delay-load architecture

Improved code generation for trees

\(\text{Seth-Ullman Example}\)
Delayed load scheduling (DLS)

**Approach**

1. Schedule the operations (la Sethi-Ullman)
2. Schedule the loads
3. Schedule the remaining \( R - 1 \) ops

**Legal ordering**

- children of an operator appear before it
- each load appears before operator that uses it

**The algorithm**

1. Run Sethi-Ullman algorithm
2. Schedule a series of (op, load) pairs
3. Schedule the remaining \( R - 1 \) ops

**The final schedule**

- preserves relative order of operations
- changes relative order of loads to operations
- preserves relative order of operations
- preserves relative order of loads

**The canonical order**

- Given \( R \) registers
- Schedule \( R \) loads
- Schedule the remaining \( R - 1 \) ops

**The algorithm**

1. Run Sethi-Ullman algorithm
2. Put loads into canonical order
3. Create an ordering of the operators using \( \min Reg + 1 \) regs
4. Requires some renaming

- This keeps extra register pressure down
Example

Canonical ordering

Operators

Loads

Example
Limitations

input

like Sertilman

Output

values not kept in registers

limited to a single basic block

handles free, not dags

Output

Limited

Strengths

fast, simple algorithm

clever metric for spilling

no excuse to do worse

non-constant delay causes deeper problems

Output

values not kept in registers

limited to a single basic block

handles free, not dags

Limited

This work raises the bar for non-optimizing compilers.

Like Sertilman