

# Compiler Technology

Wen-mei Hwu  
ECE

University of Illinois,  
Urbana-Champaign

# Basic Compiler Techniques

- Inline expansion
- Loop unrolling
- Code scheduling and compaction
- Trace/Superblock selection

# Advanced Optimizations

- Dependence removal
- Memory access elimination
- Instruction Layout
- Predication and speculation

# Code Scheduling

- Compile-time code scheduling also known as
  - static scheduling
  - code compaction
- Convert from serial code to parallel code

# Compile-time Scheduling

- Two major forms
  - local: within basic block
  - global: across basic blocks
- Originally for microcode tools
- Further developed for VLIW and superscalar processors

# Compile-time Scheduling

- Goal:
  - pack operations into bundles for simultaneous issue
- Constraints:
  - data dependences
  - control dependences for global
  - machine constraints
    - resource and timing

# List Scheduling Algorithm

- Assign priority to operations
- While there are unassigned operations
  - start a new bundle - *current*
  - identify all ready operations
  - place ready operations into *current* according to priority and machine constraints
- No backtracking

# Dependence Graph

- Operation sequencing constraints
  - register data dependences
  - memory access dependences
  - control dependences
- Constructed with
  - program analysis
  - machine description

# Dependence Graph

- Main functionality
  - assigning priority to operations before scheduling
  - identify ready operations for each bundle (clock cycle)

# Priority Heuristics

- A good heuristic finds good compromise among
  - Earliest ready time
  - Deadline to issue without stretching execution time
  - Resources (registers) freed up

# Earliest Ready Time

- Partition the dependence graph into levels (fig.)
  - start from the top level (operations without dependencies)
  - mark the value for the top level operations to 1
  - go down one level at a time

# Earliest Ready Time (cont.)

- for each operation in the current level
  - calculate constraints from each parent to be the earliest ready time of the parent + latency of the parent
  - earliest time of an operation is the max constraint among all parents

# Deadline

- Find the leaf operations with the largest earliest finish time (earliest ready time + latency), call it the bottleneck (fig.)
- Set the deadline of all leaf operations to the (earliest finish time of the bottleneck - latency of the current operations)

# Deadline (cont.)

- Start with the bottom level go up one level at a time
  - for each operation in the current level calculate constraint from each child to be (deadline of child - latency of current operation)
  - deadline of current operation is min constraint among all children

# Global Scheduling

- Important in integer code
- Insufficient parallelism within basic blocks
- Must address control dependence constraints

# Downward Movement Constraints

- When moving an operation from a BB to one of its dest BB's,
  - all the other dest basic blocks should still be able to use the result of the operation
  - the other source BB's of the dest BB should not be disturbed

# Upward Code Movement Constraints

- When moving an operation from a BB to its source BB's
  - register values required by the other dest BB's must not be destroyed
  - the movement must not cause new exceptions

# Menu Global Compaction

- Live:
  - A register is live at a point if the value stored in it may be referenced in some block including and after that point.
- Menu rules:
  - rules for moving operations across BB's (fig.)

# Menu Compaction Rules

- |    |        |        |                                    |
|----|--------|--------|------------------------------------|
| 1. | B2     | B1&B4  | free top B2                        |
| 2. | B1&B4  | B2     | same copies free at bottom of both |
| 3. | B2     | B3&B5  | Free bottom B2                     |
| 4. | B3&B5  | B2     | same copies free at top of both    |
| 5. | B2     | B3(B5) | 3.+ dest. reg dead in B5 (B3)      |
| 6. | B3(B5) | B2     | 4.+ dest. reg dead in B5 (B3)      |

# Menu Global Compaction

- Compact each basic block separately
- Sort BB's
- Scan BB's in order, move code from current basic block to a previously visited basic block
- Example: Left as homework

# Potential Improvements

- Issues with menu global compaction
  - Compaction decisions in each BB may eliminate opportunities
  - An obscure sequence of non-profitable intermediate moves may have to be applied before a profitable move can be made - a difficult search problem

# Trace Scheduling

- Identify a frequently traversed path in the control flow graph
  - call it a trace
- schedule all bb's in the same trace together as if there were no control dependency
- fix the code to account for control dependences afterwards

# Bookkeeping

- Fix the code so that execution results remain the same after trace scheduling (see Fig.)
- Joins: control transfers into a trace
- Forks: control transfers out of a trace

# Need for Speculation

- Moving branch up is usually not feasible. If branch condition can be evaluated early, the branch would have been scheduled earlier to begin with.
- This motivates speculation.

# Optimizing Bookkeep Code

- In a phase ordering compiler, the bookkeeping code generated during scheduling may not be subject to global optimization.
- This motivates superblock scheduling and region vertical compilation.

# Homework Assignment

- Work out the bookkeeping for trace (BB4, BB2, BB3)
  - Prove that bookkeeping ensures the execution result unchanged
  - Define a cost model to evaluate when trace scheduling reduces execution time considering the extra code introduced by bookkeeping.