CSE 504: Compiler Design

Instruction Scheduling

Pradipta De
pradipta.de@sunykorea.ac.kr
Current Topic

- Instruction Scheduling: Definitions
- List Scheduling for Basic Blocks
- Regional Scheduling
Instruction Scheduling

• The task of ordering operations in a block or a procedure to make effective use of processor resources

• Why do we need Instruction Scheduling
  – Operations have different latencies
    • 1 cycle: add, sub, shift
    • 2 cycles: mult
    • Integer or floating point division takes more cycles
    • Memory operations can take variable number of cycles
  – Multiple functional units in processors
    • How to prevent resources from idling
  – How to ensure that operands are ready before issuing instructions
    • Use stalls, or reorder instructions
Example

“Start” shows the cycle in which operation begins execution

The overlap among operations hides the latency of memory operations
Each node has two attributes: **operation type**, and **delay**
Defining Instruction Scheduling

• Given a dependence graph $D$, a schedule $S$ maps each node $n \in N$ to a non-negative integer denoting cycle when it should be issued, assuming first operation is issued at cycle 1

• Constraints:
  – $S(n) \geq 1$ for all $n$ (do not issue before execution starts)
  – If $(n_1, n_2) \in E$, then $S(n_1) + \text{delay}(n_1) \leq S(n_2)$
  – For each type $t$, there are no more operations of type $t$ in any cycle than the target machine can issue

• The length of a schedule $S$, denoted $L(S)$, is
  • $L(S) = \max n \in N (S(n) + \text{delay}(n))$

• The goal is to find the shortest possible correct schedule
  – Typically time optimal schedule

• Local Instruction Scheduling is a NP-hard problem
List Scheduling: Heuristic for Instruction Scheduling

• Greedy heuristic approach

• Steps:
  – Rename to avoid anti-dependencies
  – Build a dependence graph
  – Assign priorities to each operation
  – Iteratively select an operation and schedule it
    • Use a READY queue for operations that are ready for scheduling
    • At each cycle choose highest priority operation to schedule from READY queue, and update READY queue
List Scheduling Algorithm

Cycle ← 1
Ready ← leaves of $D$
Active ← $\emptyset$

while (Ready $\cup$ Active $\neq \emptyset$)
  for each op $\in$ Active
    if $S$(op) + delay(op) < Cycle then
      remove op from Active
      for each successor s of op in $D$
        if s is ready
          then add s to Ready

  if Ready $\neq \emptyset$ then
    remove an op from Ready
    $S$(op) ← Cycle
    add op to Active

Cycle ← Cycle + 1
Priority Assignment

• Different heuristics are used for priority assignment and tie breaking during selection

• Priority assignment techniques
  – node’s rank denoted by number of immediate successors
  – Node’s rank is the number of descendants
  – Node’s rank is equal to its delay \( \Rightarrow \) schedule long latency operations early
  – Node’s rank is the number of operands for which this operation is required
Regional Scheduling

• Extended Basic Blocks
  – B1 has multiple predecessors
  – Every other block Bi has exactly one predecessor

• Treat paths in an EBB as if they are single blocks \(\Rightarrow\) can apply list scheduling on the larger context

• Shared prefixes and premature exit complicates scheduling
  – B1 is a shared prefix
  – \{B2, B5\} is a premature exit
Complications in EBB

Move f to B1

Undo f
Superblock Cloning

- Join points limit opportunities for scheduling $\rightarrow$ compiler can clone blocks to create longer join-free paths
Summary

- Basic Block scheduling uses list scheduling heuristics

- To extend the scope of basic blocks, extended basic blocks are used
  - Compiler uses techniques to increase the length of EBBs