CSC2/455 Software Analysis and Improvement
Instruction Selection/Instruction Scheduling/Register Allocation

Sreepathi Pai
February 26, 2024

URCS
Outline

Review

Instruction Selection

Instruction Scheduling

Global Register Allocation

Postscript
Outline

Review

Instruction Selection

Instruction Scheduling

Global Register Allocation

Postscript
Optimizations

- Analysis
  - Iterative Dataflow Analysis
  - SSA Form
  - Interprocedural Analysis [next class]
  - Loop Analysis (maybe)
  - Type Checking

- Optimization
  - Dead Code Elimination
  - Partial Redundancy Elimination

- Code Generation
  - Instruction Selection
  - Instruction Scheduling
  - Register Allocation
Outline

Review

Instruction Selection

Instruction Scheduling

Global Register Allocation

Postscript
The Problem

\[ a = b + c \]
\[ c = c + 1 \]

gets converted to (in some pseudo-assembly language):

```
add a, b, c
inc c
```
$x = x + 1 \Rightarrow \text{inc } x$

- A pattern (or “macro”) is specified and when it matches on the 3-address code, instructions are emitted.
- Pattern matching is no longer used to select instructions
  - but still useful for something called *peephole optimization*
Tree Covering

- Pattern matching operates on sequences of instructions
  - Usually one, but could be more
- Tree-based systems operate on *trees* of expressions
  - Limited to basic blocks
See Figure 1 from the Twig Paper
Tree rewriting rules

See Table 1 from the Twig Paper
Informally, a tree pattern can be viewed as a string formed by a pre-order traversal of the tree.

All patterns can then be viewed as strings and an automaton constructed to search for strings and substrings.
See the discussion in Section 5 of the Twig Paper.

- Evaluate costs of subtrees
- Use that to evaluate cost of tree
- Use dynamic programming to pick the tree “covering” with the lowest cost.
The need for instruction scheduling

- Code is in 3-address form
- Instructions have been selected
  - e.g. $a / 4$ becomes $\text{shr } a, 2$ (shift-right) or $\text{sar } a, 2$ (shift-arithmetic right)
- What (linear) order should we output instructions?
A modern CPU pipeline

Source: Intel 64 and IA-32 Architectures Optimization Manual
Pipeline Details Known to Compiler

- Number of functional units and their types
  - Which instructions can be issued together
- Latencies of operations
  - Note some operations can have variable latencies (e.g. memory operations)
- Forwarding latencies
  - Delays when sending values from one functional unit to another
  - Or from one instruction to another
- And other sundry architecture details
  - See the architecture manuals for processor of interest
Two metrics of interest

- **Throughput**: Instructions completed per cycle
  - Higher is better
- **Latency**: Total cycles for execution
  - Lower is better
\[ \begin{align*}
a &= x + y \\
b &= v / w \\
c &= a \times b \\
d &= z \ll 1 \\
e &= c - d
\end{align*} \]
Assume the following delays:

- Add, Shifts, Subtraction: 1 cycle
- Multiplication: 2 cycles
- Division: 3 cycles

<table>
<thead>
<tr>
<th>Cycle</th>
<th>ALU1</th>
<th>ALU2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$a = x + y$</td>
<td>$b = v / w$</td>
</tr>
<tr>
<td>1</td>
<td>$d = z \ll 1$</td>
<td>(busy)</td>
</tr>
<tr>
<td>2</td>
<td>(idle)</td>
<td>(busy)</td>
</tr>
<tr>
<td>3</td>
<td>$c = a \times b$</td>
<td>(idle)</td>
</tr>
<tr>
<td>4</td>
<td>(busy)</td>
<td>(idle)</td>
</tr>
<tr>
<td>5</td>
<td>$e = c - d$</td>
<td>(idle)</td>
</tr>
</tbody>
</table>

What determines the total time of the path?
Critical Path

- The path that takes the longest to execute
  - Equivalently, has zero *slack*
- Delaying operations on critical path will increase total time
List scheduling

Goal: Schedule instructions on every cycle (i.e. build a table like in previous figure)

- Overall structure:
  - Mark operations whose predecessors have completed as ready
  - Pick operations that are ready
  - Schedule them in some order if resources available
  - Proceed to next cycle
  - Repeat until all operations are scheduled

(Figure 12.3 in Cooper and Torczon)
Always prioritize instructions on critical path.

- Can be hard to achieve
- Variable delays
- Multiple functional units
- Other architecture-specific constraints
Other Complications

- Small basic block sizes
  - Loop unrolling
  - Trace scheduling
- Long dependence chains with little parallelism
  - Software Pipelining
Outline

Review

Instruction Selection

Instruction Scheduling

Global Register Allocation

Postscript
The Memory Hierarchy (on CPUs)

- Registers (on chip, few tens to low hundreds)
- L1 cache (tens of KB)
- L2 cache (tens of MB)
- DRAM (tens of GB)
  - off-chip
Numbers every programmer should know

- Register access: 1 cycle
- L1 cache: less than 10 cycles
- L2 cache: less than 100 cycles
- DRAM: hundreds of cycles

Bottomline: Placing frequently used variables in registers can improve performance
The Problem of Global Register Allocation

- Code is in SSA form (with $\phi$-functions removed)
- SSA form uses infinite registers (each temporary is a “register”)
- Must map these logical registers to physical machine registers
Must answer two main questions:

- How many physical registers are needed? ($k$ is the number of physical registers)
  - $< k$, fewer than available – everybody gets a register!
  - $> k$, more than available – results in spill code
  - Spilled registers are stored in memory and reloaded later

- Which physical register is assigned to which variable?
A live range is a set of all definitions and uses of the same variable such that:

- If a use $u$ for variable $i$ is in $LR_i$, then all definitions $d$ that reach $u$ are also in $LR_i$.
- If a definition $d$ for variable $i$ is in $LR_i$, then all uses $u$ reached by $d$ are also in $LR_i$. 
• Live ranges
  • $LR_a = \{a_0\}$
  • $LR_b = \{b_0\}$
  • $LR_c = \{c_0\}$
  • $LR_d = \{d_0, d_1, d_2\}$
Interference Graphs

- Two live ranges “interfere” if they are both live at an operation
  - Implies that they must be in different registers
- Represented by an interference graph
  - Nodes are live ranges
  - An (undirected) edge between nodes \( n \) and \( m \) indicates interference
- Exercise: Draw an interference graph for the example on the previous slide
Given an interference graph, the minimal number of physical registers required is equal to the \textit{chromatic number} of the graph (due to Chaitin).

- Chromatic number is the minimum number of colors required such adjacent nodes have different colors
- NP-complete problem
Three Color Example

LR0
LR1
LR2
LR3
What order should instruction selection, instruction scheduling, and register allocation be performed?
References

- Aho, Ganapathi, and Tjiang, Code Generation Using Tree Matching and Dynamic Programming, ACM TOPLAS *
  - Required reading for x455 students
- Blindell, Survey on Instruction Selection: An Extensive and Modern Literature Review, arXiv
  - Recommended reading for all
- Chapter 12 of Cooper and Torczon (Instruction Scheduling)
- Chapter 13 of Cooper and Torczon (Register Allocation)
- Also recommended: Chapter 8 of the Dragon Book
- Recommended reading for a more detailed description of register allocation in real compilers: VenkataKeerthy et al., R4ReAL: Reinforcement Learning for Register Allocation, CC’23