Last Lectures (1)

- Instruction scheduling
  - List scheduling
  - Classification
    - Regime, search strategy, region shapes
- Region formation
  - Mutual most likely strategy

Last Lectures (2)

- Region Types
  - Linear
    - Traces / Superblocks
  - Non linear
    - Hyperblocks / Treecpn / Trace 2
    - SEME vs MEME
- Details on Superblocks
  - Moderate compile time penalty
  - Major performance improvements
  - Sometimes severe code size increase

Last Lectures (3)

- Region enlargement
  - Loop unrolling
  - Loop peeling
  - Tail duplication
  - If-conversion (Hyperblocks)
- Dependence elimination
  - Renaming
  - Induction variable / accumulator expansion
  - Operation combining and migration
In Today's Lecture

- Cyclic Scheduling
  - Software Pipelining
  - Modulo Scheduling
- Predicated Execution
  - If-conversion

Cyclic Scheduling

- Programs spend most of the time in loops
- Apply specialized scheduling techniques
  - Enlarge the loop body by unrolling
  - Overlap the execution of several loop iterations
  - Execute (parts of) different iterations in parallel
  - When applicable leads to good improvements
    - Not all loops can be handled

Software Pipelining

- Family of cyclic schedulers
  - Divide the loop into stages
  - Execute stages of different iterations in parallel
  - Effectively pipelines the loop
    (similar to pipelining for computer architectures)
  - Dominating approach: Modulo Scheduling
- Optimize for throughput
  - The latency of a single iteration is irrelevant
Modulo Scheduling (1)

- Important SW Pipelining Technique
  - Explores the space of possible loop kernels
  - Initiation interval (II)
    - Constant interval between the start of successive kernel iterations
  - Lower bound of II (MinII)
    - Consider available hardware resources (ResII)
    - Data dependencies and recurrences (ReclII)
  - Upper bound of II (MaxII)
    - Length of a linear schedule

Modulo Scheduling (2)

- Search for an II starting from MinII
  - If a valid schedule could be found stop
  - Otherwise decide whether to
    - Backtrack - revert some scheduling decision
    - Increase the II
    - Abort if II is larger than MaxII
  - High computational complexity
    - Backtracking and searching the II
Scheduling Heuristics

- Modified version of list scheduling
  - Employ the modulo reservation table (MRT)
    - Similar to regular reservation tables
    - Ensure that a resource is not used at the same cycle, or at following cycles that modulo the II collide with it
  - Allows backtracking to revert scheduling decisions
    - Limited by a backtracking budget
    - May cause significant overhead

Prolog / Epilog

- Partial copies of the loop kernel
  - May cause some code size increase
  - Strongly depending on the number of stages
- Several versions required in the presence of multiple loop exits and uncertain trip counts
- May harm runtime on small trip counts
  - Loop kernel is never reached (steady state)

Example: Modulo Scheduling

```c
int i, x;
short a[];
for (i=0; i < 100; ++i) 
  a[i+1] = a[i]*x + 7;
```

- Original C Code

Variable Names

- MS increases the register pressure
  - Preserve multiple copies of the same variable for different iterations
    - Life ranges exceeding the II
  - Spilling may disrupt the compact schedule
- Solutions
  - Modulo variable expansion (loop unrolling)
  - Copy operations
    - Hardware support (rotating register files)
Example: Variable Names (1)

- Minor change in the schedule
- The value of \( ldh \) exceeds the II
  - \( mpyl \) still requires the value of the last iteration

Example: Variable Names (2)

- Solution
  - Shown here: A extra copy operation
  - Alternative: Modulo Variable Expansion

Modulo Variable Expansion

- Unrolling increases the II
  - It is always possible to find a suitable unrolling factor to prevent copy operations
    - \( \text{Assume } y \text{ is the longest lifetime of a variable} \)
    - \( \text{The unrolling factor is given by } k = \lfloor \frac{1}{T} \rfloor \)
  - Larger kernel, prolog, and epilog
  - Complicates handling of loop exits

Limitations of Modulo Scheduling

- Internal control flow
  - No internal control flow permitted
    - Applying if-conversion helps
  - Early loop exits are complex to handle
- Nested Loops
  - Recursively invoke the modulo scheduler starting with the innermost loop
- Low trip counts
  - Possibly never reach the steady state
  - Two versions unrolling & pipelined loop (code bloat)
Software Pipelining on a MIPSPro

- SPECfp92 on a MIPSPro (R8000, 75Mhz)
- SPECmarks with pipelining enabled/disabled

Predicated Execution

- Predicated operations
  - Conditionally nullify the result of the operation
  - Partial vs. full predication
    - All operations can be predicated
    - Conditional move or select operations

- Enable elimination of branches
  - Increase available ILP

Software Pipelining on a VLIW

- TriMedia TM1000
  - 5 issue VLIW
  - 100 Mhz
- Benchmarks
  - DSPStone
  - MiBench

If-Conversion

- Restructure the CFG
  - Augment blocks with predicates
  - Merge blocks
  - Eliminate branches
- Additional benefits
  - Increase scope for schedulers (Hyperblocks)
  - Enable software pipelining (internal control flow)
  - Other optimizations may benefit as well
Example: If-Conversion

- Simple approaches
  - Select a candidate basic block following a heuristic
    - Often following simple patterns (e.g., if-then-else, etc.)
  - Calculate a predicate
  - Predicate all instructions of the block
- More Sophisticated
  - Use the Program Dependence Graph (PDG)
  - Try to place predicate definitions optimally
  - Try to derive a minimal set of predicates

Program Dependence Graph

If-conversion using RK'
If-conversion on Itanium (1)

- SPECINT2000, running on a near-production silicon

* SPECINT2000, running on a near-production silicon

If-conversion on Itanium (2)

- SPECINT2000, running on a near-production silicon

Limitation of If-Conversion

- Relies on hardware support
  - Non predicable instructions cause problems
- Aggressive application
  - Causes slow-down, increase of code size, and power consumption
  - Estimating profitability is hard
  - Reverse-if-conversion
    - Undo if-conversion if unprofitable
Outlook

- Second Assignments
  - Presentation of your results
    - Minimal execution time of the mpeg2decoder
    - Best scheduler implementation
  - The winner gets a small price