CSE548 – Lecture 5 Highlights VLIW

Thierry Moreau

VLIW background

- Microprogramming
  - Implementing complex operations for control units in CPUs (think of steps in the execution of an instruction)
  - Vertical programming vs. horizontal programming
- VLIW
  - Descends from horizontal microprogramming
  - Long instructions : specifies multiple operations that can be performed simultaneously
- Principles:
  - Compiler plays key role in detecting ILP
  - Architecture should provide features that assist the compiler in determining the most common ways of exploiting ILP
  - Architecture should provide mechanisms to communicate the parallelism detected by the compiler to the underlying hardware

VLIW benefits

- Why were people interested in VLIW up to the mi-90s?
  - Hardware of in-order statically scheduled processors is much simple and O3
    - Smaller hardware footprint, faster clock cycles
  - Compiler-based exploitation of ILP should be favorable
    - Compiler looks at the whole program rather than the comparatively small instruction window
  - Not as restrictive as SIMD vector machines
- The challenge:
  - Single long instruction op need to be performed *without hazards* within the same cycle
  - Requires static scheduling… Not good because of branch prediction and various latencies from mem-ops

VLIW and the ELI-512

- Trace scheduling:
  - Compaction of long streams of code
  - Control speculation: pick a stream with highest probability of execution
  - Post-processor inserts new code at the stream exits and entrances for recovery
- Jump mechanism:
  - n independent tests results in n+1 possible jumping location
  - Delayed-branch mechanism
- Memory accessing:
  - Bank prediction, precedence of local searches, pre-looping
- Problems:
  - Adding enough test instructions without making the machine too big
  - Put enough memory references in each instruction without making the machine too slow

Grid Processor Architectures

- Array of ALUs, with limited control, connected by a thin operand network
- Block atomic execution model
  - Map large instruction blocks to nodes
  - Eliminates the need for a centralized instruction issue window, or register renaming table, fewer register file R&W
  - Converts the conventional broadcast bypass network into a routed point to point network
- Compiler does the work of statically scheduling
  - BUT instructions are issued dynamically with the execution order determined by the availability of input operands
- Hyperblock control:
  - Predication: execute-all approach; special instruction (cmove)
  - Early exits: branch from the middle of a block
  - Block commit: all stores and output reg values have been committed
- Results: high IPC (1-9)
- Problems: memory access, frame management,