Speculation and Future-Generation Computer Architecture

Guri Sohi

University of Wisconsin — Madison
URL: http://www.cs.wisc.edu/~sohi
Outline

• Computer architecture and speculation
  • control, dependence, value speculation

• Multiscalar: next generation microarchitecture
Processing: Big Picture

- **Sequence** through static representation of program to create dynamic stream of operations
- **Execute** operations in the dynamic stream
  - **Determine** dependence relationships
  - **Schedule** operations for execution
  - **Execute** operations
  - **Communicate** values
Role of Computer Architect

- Use available technology to perform processing tasks
- Match processing tasks to hardware blocks constructed from available technology
- Do so in a manner that is easy to design/verify
Constraints for the Architect

• Program ordering constraints, a.k.a, dependences
  • Control
  • Artificial (i.e., name or storage)
  • Ambiguous
  • True

• Increased latencies manifest as performance-degrading stalls through program dependences

• Need for architect to develop techniques to overcome dependences
Speculation and Computer Architecture

Speculation: “.. to assume a business risk in hope of gain”

-- Webster

• Speculation in computer architecture is used to try to overcome constraining conditions
Speculation and Computer Architecture

- Speculate outcome of event rather than waiting for outcome to be known
  - mis-speculation if wrong
  - mis-speculation can have penalty
- Develop techniques to speculate better
Memory Hierarchies

- Technology trend: small and fast, or large and slow
- Can I get fast with appearance of large?

- **Speculate**: going to use referenced data item again (temporal locality)
- **Speculate**: going to reference data items in proximity to referenced item (spatial locality)
Superscalar (Generation 3)

- Technology allows large execution engines to be built
- Feeding execution engines is a problem
  - branch instructions constrain program sequencing and instruction fetching
  - waiting for branch outcome wasteful
- Speculate the outcome of a branch (control speculation)
- Use mechanisms to improve accuracy of speculation: branch predictors
Software or Hardware?

- Speculation can be done in both hardware and software
- Software speculation requires instruction set (ISA) support
  - problematic if ISA change not practical
  + requires little hardware support
- Hardware speculation does not require ISA support
  + better if business reasons preclude changing ISA
  - requires more hardware
Hardware Speculation Mechanisms

- Need to give appearance of sequential execution

- Need to recover from mis-speculations
  - history buffers: keep checkpoint, and back up to checkpoint
  - reorder buffers: do not update (architectural) state until speculation outcome is known

- Speculation improvement mechanisms
  - branch predictors
Observation

- Mechanisms rely on ability to give appearance of sequential execution
- Appearance of sequential execution implies appearance that no ordering constraints were violated
- Basic (recovery) mechanisms for supporting control speculation can be used to support arbitrary forms of speculative execution!
Out-of-Order Processing

Processing Phase

Program Form

static program

dynamic instruction stream

execution window

Execution Wavefront

completed instructions

instruction fetch & branch prediction

dependence checking & dispatch

instruction issue instruction execution

instruction reorder & commit

Guri Sohi Speculation and Future-Generation Computer Architecture Slide 14
Generic Superscalar Processor

- Pre-decode
- Instruction cache
- Instruction buffer
- Register rename & dispatch
- Integer/address instruction buffers
- Functional units
- Floating pt. instruction buffers
- Floating pt. register file
- Integer register file
- Re-order buffer
- Functional units and data cache
- Memory interface
Technology Trends

- Wires used to pass values
- Wires getting relatively slower
  - Short wires for fast clock
  - Short wires implies localized communication
Alpha 21264

Reservation Stations

Reg File0

Exec

Reg File1

Exec

1 CP delay

Exec

Exec
Decentralized Scheduling

• Scheduler in one cluster needs dependence information for instructions in other cluster
  • problematic for memory operations
    • knowledge of memory operations needed
    • address calculations needed

• Can we work without knowledge of memory operations/addresses, i.e., ambiguous dependences?

• Use data dependence speculation
Scheduling Memory Operations

- Data dependence speculation is the default
  - predict no dependences

- Improving accuracy of data dependence prediction
  - akin to branch prediction for control dependences

- Track history of dependence mis-speculations
  - small number of static dependence pairs
  - exhibit temporal locality

- Use history for future data dependence speculation/synchronization decisions
Overcoming True Dependences

• Break true dependence using value speculation
  • predict the outcome of an operation (e.g., 32 bits)

• Mechanisms to support other forms of speculation (e.g., control speculation) can be used to recover

• Subject of significant current research
Multiscalar Processors

- Paradigm for Generation 4 processors
- Proposed in early 1990s for 2000s processors
- Employs “thread-level speculation”
- First commercial example of concept: Space Time Computing (STC) in Sun MAJC
Processing Hardware: Big Picture

- Start with a static representation of a program
- **Sequence** through the program to generate the dynamic stream of operations
  - Use single PC to walk through static representation?
- **Execute** operations in dynamic stream
  - **Schedule** operations for execution
  - **Execute** operations
  - **Communicate** values
Basic Issues

• Sequencing
• Scheduling
• Operation execution
• Operand communication

• How do we sequence, schedule, execute, and communicate in a more powerful manner?
Trends

- Lots of transistors available
- Wires getting slower
- Design complexity getting unwieldy
- Verification complexity getting unwieldy
- Software complexity getting unwieldy
Hardware Wish List

- Simplify engineering (design, verification, testing)
  - Use of simple, regular hardware structures
    - clock speeds comparable to single-issue processors
  - “Locality” of interconnect
  - Easy growth path from one generation to next
    - reuse existing processing cores
    - No centralized bottlenecks

- Still build high-performance processor
  - speed up execution of single program
The “Hardware-Influenced” Solution

- Take current generation processor
- Replicate some parts, share others
- Have next generation processor
- Different units can sequence, schedule, etc. in parallel

BUT, the software problem .......
The Software Problem

• Can’t always break up single program into “independent” chunks (i.e., multiple sequencers) statically
  • control dependences
  • data dependences (especially ambiguous ones)
  • also load balance

• Can’t map program onto rigid hardware model
Hardware/Software Cooperation

• Take “mostly sequential” static program
• Use speculation to overcome dependence limitations
  • When in doubt, speculate
• Break up program into “potentially independent” chunks dynamically
Sequencing

• Unraveling the operations to be executed dynamically

• Use 2-level sequencing
  • sequence high level in task-sized steps
  • sequence within task \((\text{task} == \text{thread})\)
  • vectors?

• Use control flow speculation to increase sequencing power
  • overcome “stalls”
Scheduling

- Use *multiple schedulers* to improve scheduling power
- Use *data dependence speculation* to overcome scheduling limitations
  - ambiguous dependences
- Use *value speculation* to overcome scheduling limitations
  - true dependences
Operand Communication

- Values bound to registers and memory
- Values created speculatively
- Storage
  - where should values be buffered?
- Synchronization
  - operation uses value of latest producer
- Communication
  - forwarding created value to (future) consumers
- Create and exploit localities to reduce/simplify interconnect!
**Multiscalar Paradigm**

- Break sequencing process into two steps
  - Sequence through static representation in *task-sized* steps
  - Sequence through each task in conventional manner
- Split large instruction window into ordered tasks
- Assign a task to a simple execution engine; exploit parallelism by *overlapping execution of multiple tasks*
- Use separate PCs to sequence through separate tasks
- Maintain the *appearance* of a single-PC sequencing through the static representation
- Use control and data dependence speculation
What is a Task/Thread?

• A portion of the static representation resulting in a contiguous portion of the dynamic instruction stream

  - part of a basic block
  - basic block
  - multiple basic blocks
  - loop iteration
  - entire loop
  - procedure call, etc.
Multiscalar Big Picture: Basics

PROGRAM

PROC UNIT 1

predict

predict

PROC UNIT 2

PROC UNIT 3

A

B

C

Guri Sohi
Speculation and Future-Generation Computer Architecture
Slide 34
Multiscalar: Logical Hardware

SEQUENCER

PIPELINE P.U. REG
PIPELINE P.U. REG
PIPELINE P.U. REG
PIPELINE P.U. REG

MEMORY DISAMBIGUATION CACHE HIERARCHY
Register Values

- Each core works out of its “local” register file
- Multiple register files act like separate “renamed” files
- Each register file contains register state at a particular time in the (speculative) execution of a program
Memory Values

- Storage
- Synchronization
- Communication
- Versions
Scheduling Memory Operations

• Data dependence speculation is the default
  • predict no dependences

• Improving accuracy of data dependence prediction
  - akin to branch prediction for control dependences

• Track history of dependence mis-speculations
  • small number of static dependence pairs
  • exhibit temporal locality

• Use history for future data dependence speculation/synchronization decisions
Example: Problem

• Process stream of tokens
• Create entry in list for new token
• Use information in list to process token
Example: C Code

```c
for (indx = 0; indx < BUFSIZE; indx++) {
    /* get the symbol for which to search */
    symbol = SYMVAL(buffer[indx]);

    /* do a linear search for the symbol in the list */
    for (list = listhd; list; list = LNEXT(list)) {
        /* if symbol already present, process entry */
        if (symbol == LELE(list)) {
            process(list);
            break;
        }
    }

    /* if symbol not found, add it to the tail */
    if (! list) {
        addlist(symbol);
    }
}
```
Example

- Each task is a complete list search
- Searches are usually independent and parallel
  - Multiscalar can assume they are always independent
- Branches that separate tasks are predictable
- Branches within a task unlikely to be 100% predictable
  - Superscalar/VLIW unlikely to be able to overlap processing of different tokens
Example: Executable

Going from one generation to another could leave binary untouched!
Concluding Remarks - 1

- Speculation is a very important tool in computer architect’s toolset
  - used to overcome performance-limiting constraints
  - control, dependence, value, other forms, .....  
  - heavy use likely in future computer systems

- Wire delays will cause future microarchitectures to be distributed

- Distributed microarchitectures have many pluses
  - natural support of “threads”
  - easier to design, verify, test?
  - fault-tolerance
• Multiscalar model enables distributed execution of a sequential (or parallel) program
  - makes heavy use of dependence speculation
• Beginning of a new generation of microarchitectures
  - much works remains to be done
• More info at www.cs.wisc.edu/~mscalar