Why Multiscalar?

Guri Sohi

Wisconsin Multiscalar Project
University of Wisconsin — Madison
URL: http://www.cs.wisc.edu/~mscalar

Outline

- The Problem
- Processing basics and wish lists
- Options for high performance
- The multiscalar model
- Performance results
- Concluding remarks

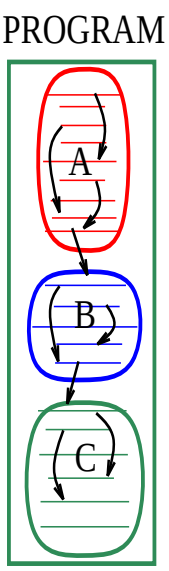
The Problem

Software: create static image of dynamic computation

Hardware: recreate dynamic computation from static representation and carry out computation

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

The Big Question

- How do we sequence, schedule, execute, and communicate in a more powerful manner?
- Powerful =
  - a large variety of applications
  - time efficient
  - space efficient
  - power efficient?
  - etc.

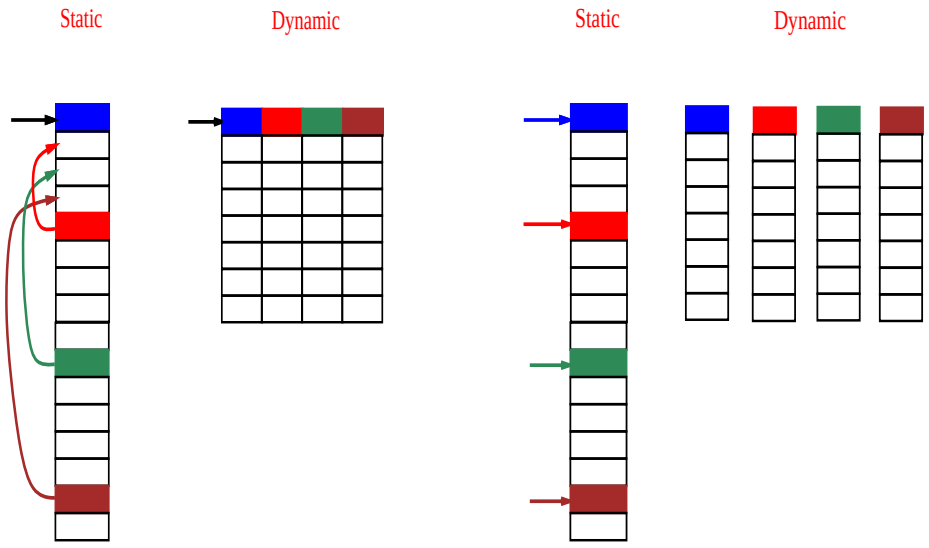
Target: 10 IPC

- Sequence through static program and establish a large instruction window (100s of instructions)
- Maintain a large window
- Sequence through program and initiate at least 10 operations into this window per cycle
- Schedule for execution at least 10 operations per cycle
- Provide lots of storage for inter-operation communications

Options

- Sequencing: single (wide) vs. multiple
- Scheduling: central vs. distributed
- Operation execution: not much choice; provide requisite bandwidth
- Operand communication: central vs. distributed storage

Sequencing/Scheduling Options



Single Sequencer

Multiple Sequencers

Engineering Considerations

- Now throw in engineering into the big picture
- What is desirable from the engineering viewpoint?
  - Hardware wish list
  - Software wish list

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

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 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

Software Wish List

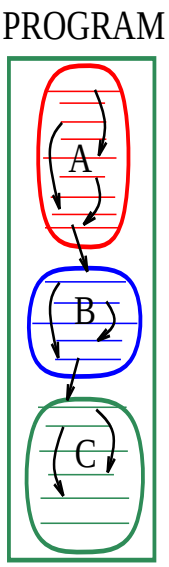
- Simplify engineering
  - Don't force "rigid model"
  - Don't ask for guarantees
  - Don't expect software to track hardware
  - Others .....

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

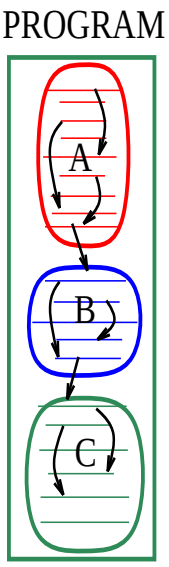
Sequencing

- Unraveling the operations to be executed dynamically
- Use 2-level sequencing
  - sequence high level in task-sized steps
  - sequence within task
  - vectors?
- Use control flow speculation to increase sequencing power
  - overcome “stalls”



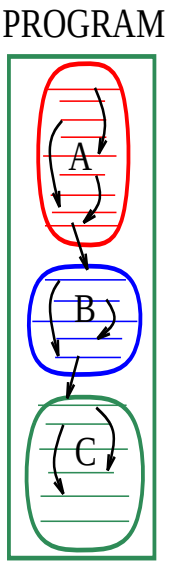
Scheduling

- 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
- Use *memoization* to avoid re-doing work
  - 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

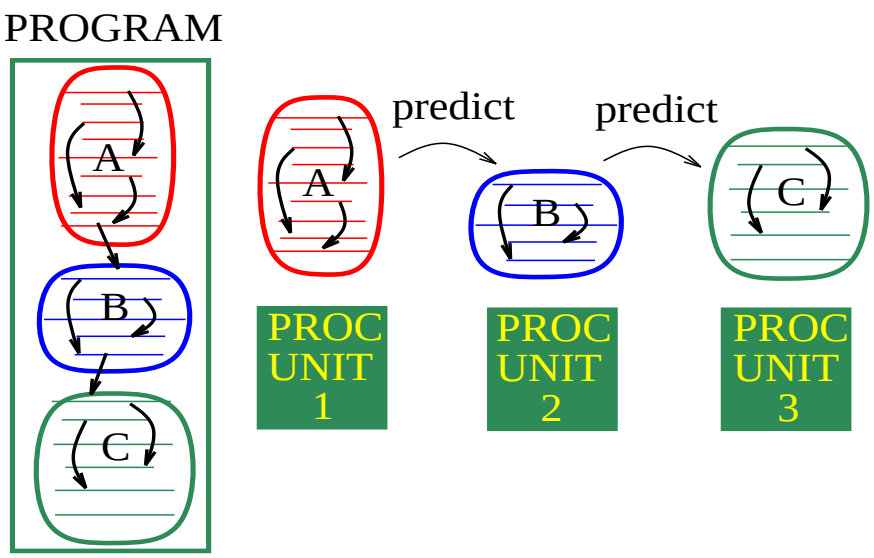
- Break sequencing process into two steps
  - Sequence through static representation in *task-sized* steps
  - Sequence through each task in conventional manner
- Split large instruction window into ordered tasks
- Assign a task to a simple execution engine; exploit ILP by overlapping execution of multiple 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?

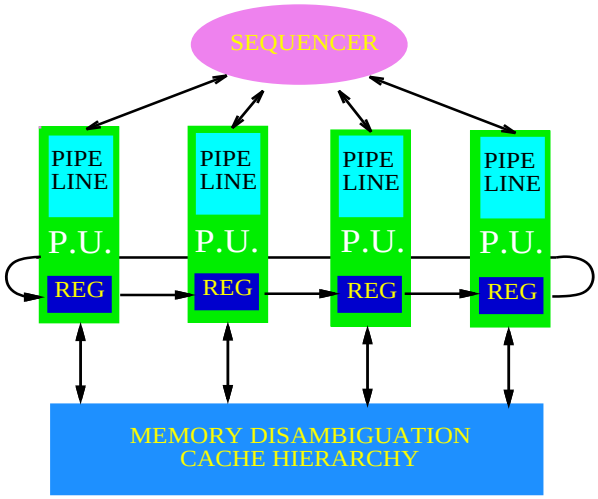
- A portion of the static representation resulting in a contiguous portion of the dynamic instruction stream

  - part of a basic block
  - basic block
  - multiple basic blocks
  - loop iteration
  - entire loop
  - procedure call, etc.

Multiscalar Big Picture: Basics



Multiscalar Big Picture: Hardware



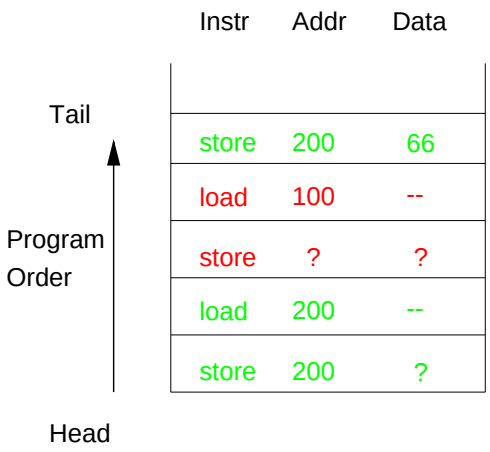
Register Values

- 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

Traditional Memory Interface: Load Store Queue

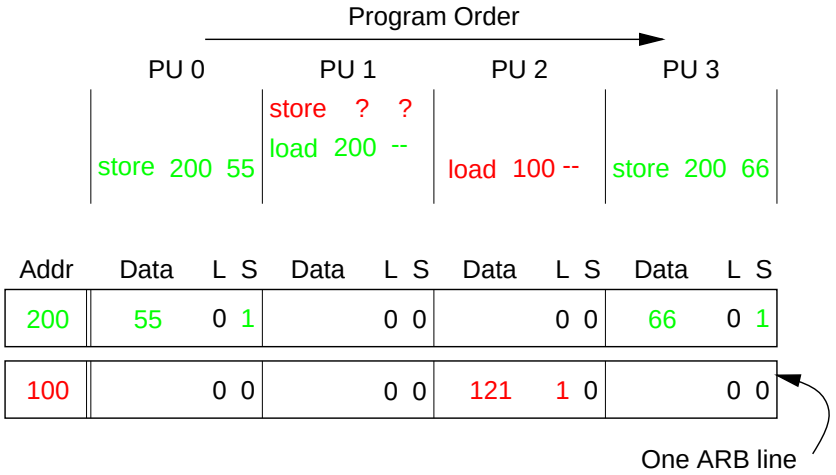


Arranged by
PROGRAM ORDER

Searched using
ADDRESS

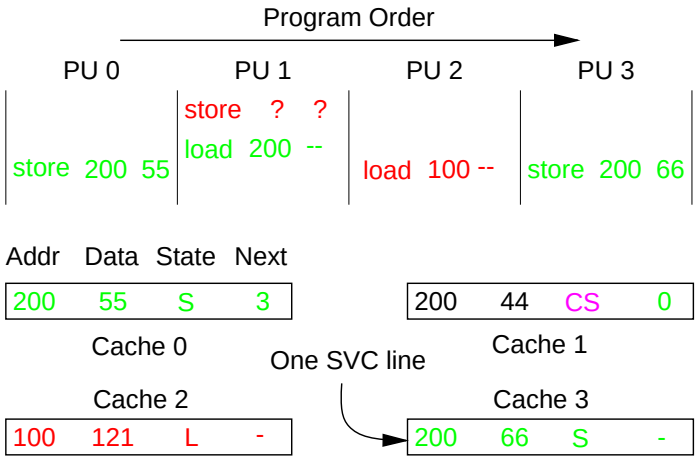
- Memory Dependence Speculation
- Multiple Versions

Memory System I: Address Resolution Buffer



- Arranged by ADDRESS; Searched using PROGRAM ORDER
- Each line buffers multiple versions
- Committed versions are written back immediately

Memory System II: Speculative Versioning Cache



- Maintains a linked list of versions; PU #s used as pointers
- Each line buffers only one version
- Committed versions written back only when necessary

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

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

    /* do a linear search fo rthe 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

```
Targ Spec      Branch, Branch
Targ1          OUTER
Targ2          OUTERFALLOUT
Create mask    $4,$8,$17,$20,$23
```

		Forward Bits	Stop Bits
OUTER:			
	addu $20, $20, 16	F	
	ld $23, SYMVAL−16($20)	F	
	move $17, $21		
	beq $17, $0, SKIPINNER		
INNER:			
	ld $8, LELE($17)		
	bne $8, $23, SKIPCALL		
	move $4, $17		
	jal process		
	j INNERFALLOUT		
SKIPCALL:			
	ld $17, NEXTLIST($17)		
	bne $17, $0, INNER		
INNERFALLOUT:			
	release $8, $17		
	bne $17, $0, SKIPINNER		
	move $4, $23	F	
	jal addlist		
SKIPINNER:			
	release $4		
	bne $20, $16, OUTER		Stop Always
OUTERFALLOUT:			

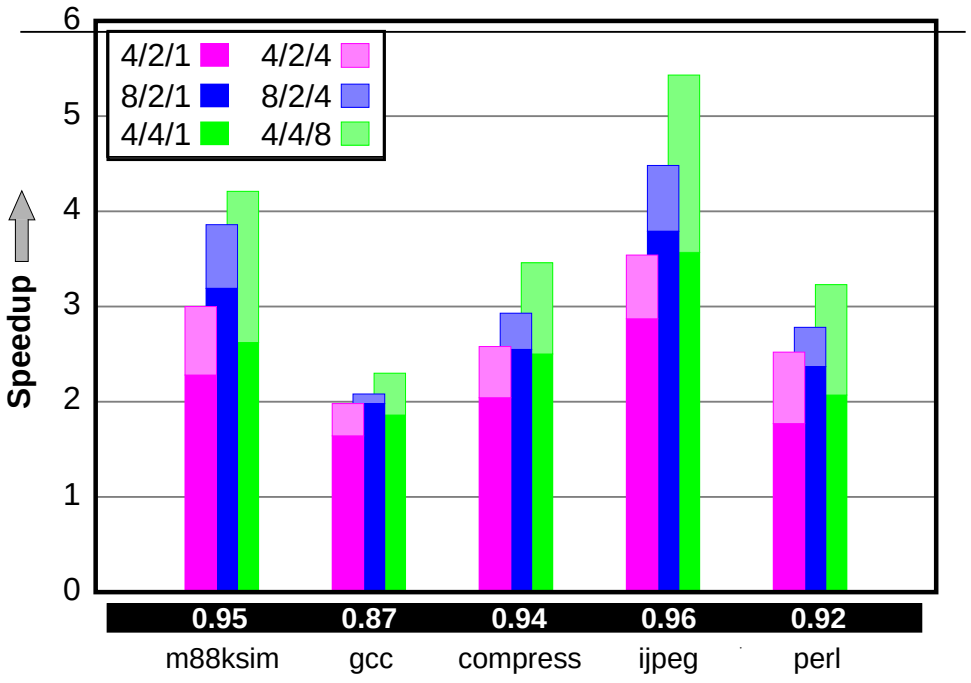
Going from one generation to another could leave binary untouched!

Binary Compatibility Options

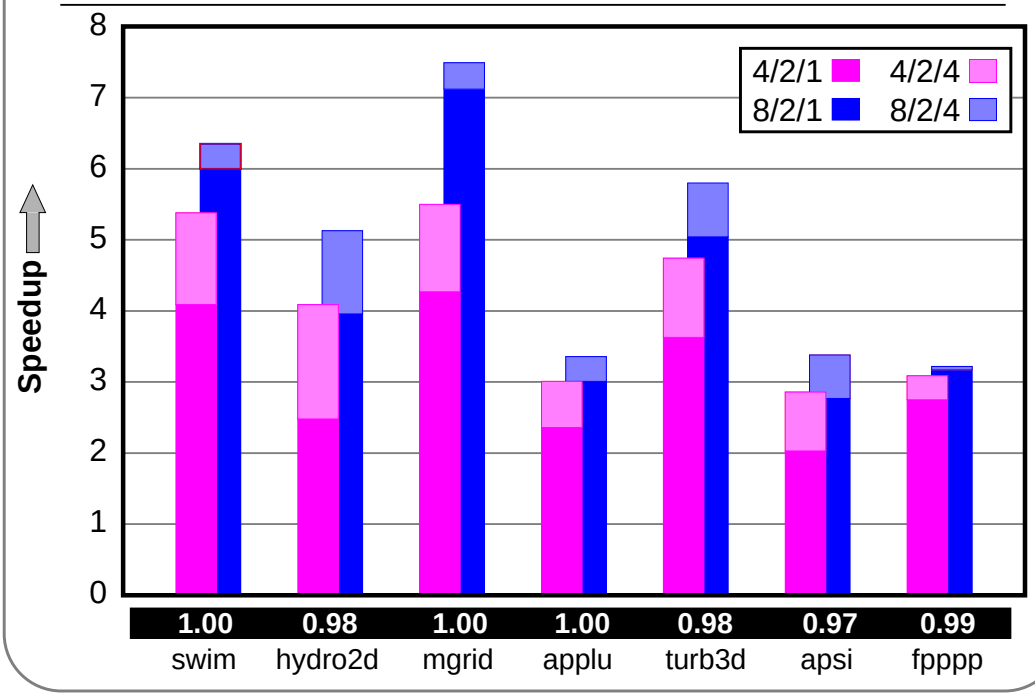
- Multiscalar-specific information (task successors, create masks, forward bits, stop bits) is available in a binary
- Recover information at run time
  - "Low" performance but run ordinary binaries
- Binary to binary translation
  - Better performance by including some optimizations
- Compiler
  - Best performance, but needs recompilation

Regardless, binary from one multiscalar generation to another can remain the same

Performance: SPECint95



Performance: SPECfp95



Comparison with Multiprocessors

Attributes	Multiprocessor	Multiscalar
Speculative task initiation	No/Difficult	Yes
Multiple flows of control	Yes	Yes
Task determination	Static	Static (possibly dynamic)
Software guarantee of inter-task control independence	Required?	Not required
Software knowledge of inter-task data dependences	Required?	Not required
Inter-task sync.	Explicit	Implicit/Explicit
Inter-task communication	Through memory Through messages	Through registers and memory
Register space	Distinct for PEs	Common for PEs
Memory space	Common Distinct	Common for PEs

Why Not Multiscalar?

- Programs can be (statically) parallelized easily
- Hardware replication not desirable
  - interconnect not an issue (copper?)
  - centralized designs easier to design/validate
  - centralized designs easier to test

Concluding Remarks

- Future microarchitectures will be decentralized (operation execution, operand communication, scheduling, sequencing)
- Multiscalar model enables distributed execution of a sequential (or parallel) program
- Beginning of a new generation of microarchitectures
  - much works remains to be done

Related Projects

- Stanford HYDRA
- CMU STAMPede
- Minnesota Superthreaded
- Waikato WARP engine
- Washington SMT
- MIT M-machine
- MIT RAW
- Michigan HPS
- CMU Superflow
- Illinois IMPACT

Acknowledgments

Thanks to:

Faculty: Jim Smith, Chuck Kime

Former/Current Students/visitors: Manoj Franklin, Scott Breach, T. N. Vijaykumar, Dionisios Pnevmatikatos, Andreas Moshovos, Todd Austin, Eric Rotenberg, Quinn Jacobson, Jeremy Williamson, Paul Thayer, Selim Bilgin, Matt Kupperman, Subramanya Sastry, Amir Roth, Sridhar Gopal, Matt Mergener, Craig Zilles, Atsushi Okamura, Anand Kamannavar, Padmaja Nandula.

for helping to conceive, develop, realize, and refine the multiscalar vision

Thanks to: DARPA, NSF, Intel for funding the efforts