The Expandable Split Window Paradigm for Exploiting Fine-Grain Parallelism

Manoj Franklin and Gurindar S. Sohi

Presented by Allen Lee

May 7, 2008

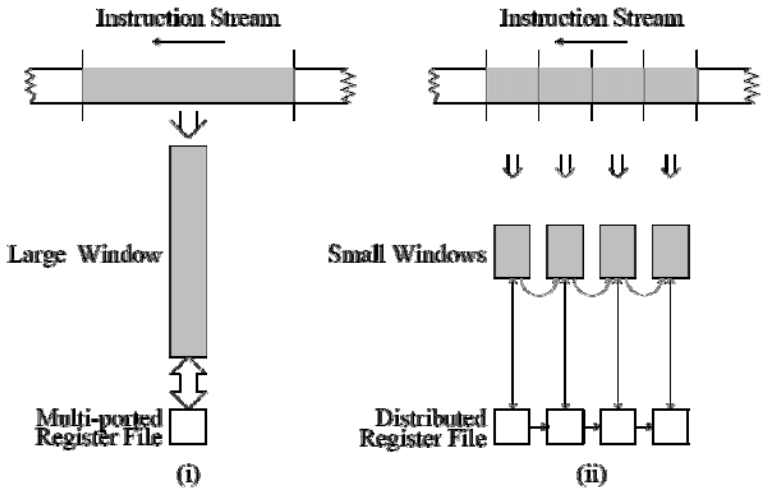
Overview

- There exists a large amount of theoretically exploitable ILP in many sequential programs
- Possible to extract parallelism by considering a large "window" of instructions
  - Large windows may have large communication arcs in the data-flow graph
- Minimize communication costs by using multiple smaller windows, ordered sequentially

Definitions

- **Basic Block**
  - *"A maximal sequence of instructions with no labels (except possibly at the first instruction) and no jumps (except possibly at the last instruction)"* - CS164 Fall 2005 Lecture 21
- **Basic Window**
  - *"A single-entry loop-free call-free block of (dependent) instructions"*

Splitting a Large Window



Example

Pseudocode:

```
for(i = 0; i < 100; i++) {
   x = array[i] + 10;
   if(x < 1000)
     array[i] = x;
   else
     array[i] = 1000;
}
```

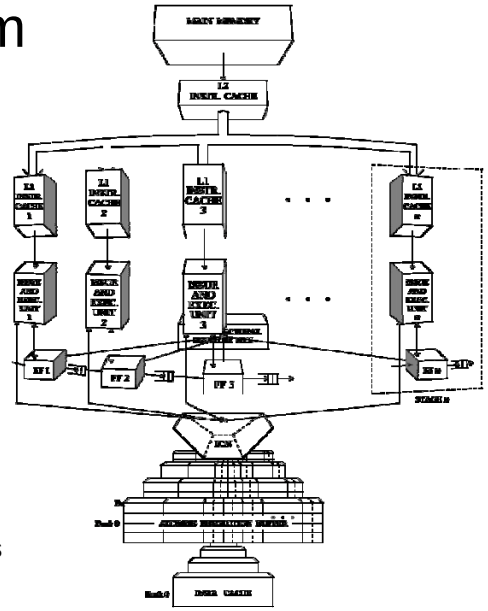
Assembly:

```
A: R1 = R1 + 1
   R2 = [R1, base]
   R3 = R2 + 10
   BLT R3, 1000, B
   R3 = 1000
B: [R1, base] = R3
   BLT R1, 100, A
```

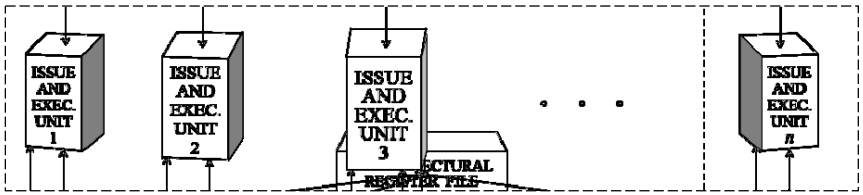
Basic Window 1	Basic Window 2
A_1: $R1_1 = R1_0 + 1$	A_2: $R1_2 = R1_1 + 1$
$R2_1 = [R1_1, base]$	$R2_2 = [R1_2, base]$
$R3_1 = R2_1 + 10$	$R3_2 = R2_2 + 10$
BLT $R3_1$, 1000, B_1	BLT $R3_2$, 1000, B_2
$R3_1 = 1000$	$R3_2 = 1000$
B_1: $[R1_1, base] = R3_1$	B_2: $[R1_2, base] = R3_2$
BLT $R1_1$, 100, A_2	BLT $R1_2$, 100, A_3

Block Diagram

- *n* independent, identical *stages* in a circular queue
- Control unit (not shown) assigns windows to stages
- Windows are assigned in sequential order
- Head window is “committed” when execution completes
- Distributed units for
  - Issue and execution
  - Instruction supply
  - Register file (future file)
- Address Resolution Buffers (ARBs) for speculative stores

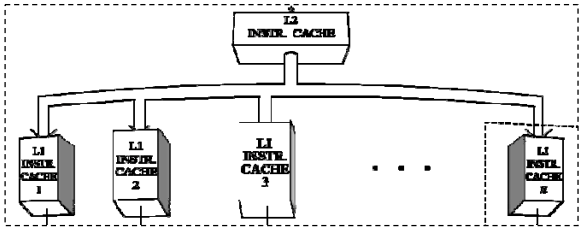


Distributed Issue and Execution



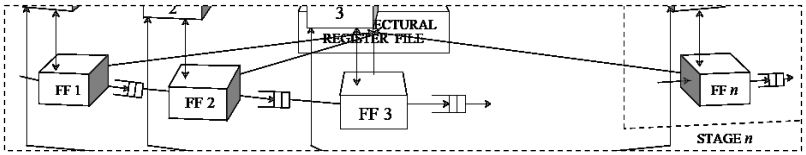
- Take operations from local instruction cache and pumps them into functional units
- Each IE has own set of functional units
- Possible to connect IE units to common Functional Unit Complex

Distributed Instruction Supply



- Two-level instruction cache
  - Each stage has its own L1 cache
  - L1 misses are forwarded to L2 cache
  - L2 misses are forwarded to main memory
- If the transferred window from L2 is a loop, L1 caches in subsequent stages can grab it in parallel (“snarfing”)

Distributed Register File



- Each stage has separate register file (future file)
- Intra-stage dependencies enforced by doing serial execution within IE unit
- Inter-stage dependencies expressed using masks

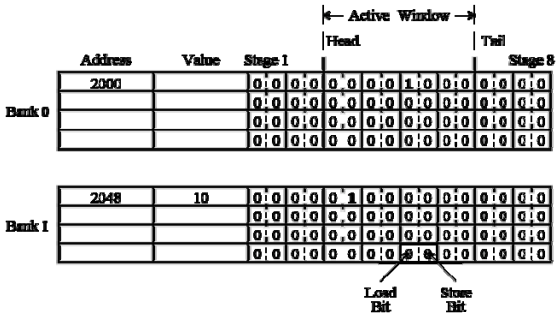
Register Masks

- Concise way of letting a stage know which registers are read and written in a basic window
- **`use masks`**
  - Bit mask that represents registers through which externally-created values flow in a basic block
- **`create masks`**
  - Bit mask that represents registers through which internally-created values flow out of a basic block
- Masks fetched before instructions fetched
- May be statically generated at compile-time by compiler or dynamically at run-time by hardware
- Reduce forwarding traffic between stages

Data Memory

- **Problem:** Cannot allow speculative stores to main memory because no undo mechanism, but speculative loads to same location need to get the new value
- **Solution:** Address Resolution Buffers

Address Resolution Buffer



- Decentralized associative cache
- Two bits per stage: one for load, one for store
- Discard windows if load/store conflict
- Write store value when head window “commits”

Enforcing Control Dependencies

- Basic windows may be fetched using dynamic branch prediction
- Branch mispredictions are handled by discarding subsequent windows
  - The tail pointer in the circular queue is moved back to the stage after the one containing the mispredicted branch

Simulation Environment

- MIPS R2000 – R2010 instruction set
- Up to 2 instructions issued/cycle per IE
- Basic window has up to 32 instructions
- 64KB, direct-mapped data cache
- 4Kword L1 instruction cache
- L2 cache with 100% hit rate
- Basic window = basic block

Results with Unmodified Code

Benchmarks	Mean Basic Block Size	No. of Stages	Branch Prediction Accuracy	Completion Rate
eqntott	4.19	4	90.14%	2.04
espresso	6.47	4	83.13%	2.06
gcc	5.64	4	85.11%	1.81
xlisp	5.04	4	80.21%	1.91
dnasa7	26.60	10	99.13%	2.73
doduc	12.22	10	86.90%	1.92
fpppp	113.42	10	88.86%	3.87
matrix300	21.49	10	99.35%	5.88
spice2g6	6.14	10	86.95%	3.23
tomcatv	45.98	10	99.28%	3.64

- Completion Rate = # of completed instructions per cycle
- Speedup > Completion Rate

Results with Modified Code

Benchmarks	No. of Stages	Prediction Accuracy	Completion Rate
eqntott	4	95.58%	4.23
eqntott	8	96.14%	4.97
espresso	4	92.17%	2.30
dnasa7	10	98.95%	7.17
matrix300	10	99.34%	7.02
tomcatv	10	99.31%	4.49

- Benchmarks with hand-optimized code
  - Rearranged instructions
  - Expanded basic window

Conclusion

- ESW exploits fine-grain parallelism by overlapping multiple windows
- The design is easily expandable by adding more stages
  - But limits to snarfing…