The Anatomy of the Register File in a Multiscalar Processor

Scott E. Breach T.N. Vijaykumar Gurindar S. Sohi

Computer Sciences Department

University of Wisconsin – Madison

Talk Outline

- Example
- Problem
- Multiscalar Architecture
- Register File
- Evaluation

Code

n ↔ r1
i ↔ r2
max ↔ r3
a[i] ↔ r4

```
max = 0;
for (i = 0; i < n; i++) {
    if (a[i] > max) {
        max = a[i];
    }
}
```

```
        move    r1, n
        move    r2, 0
        move    r3, 0
        bge     r2, r1, done
loop:
        add     r2, r2, 4
        lw      r4, –4(r2)
        ble     r4, r3, test
        move    r3, r4
test:
        blt     r2, r1, loop
done:
```

loop prologue

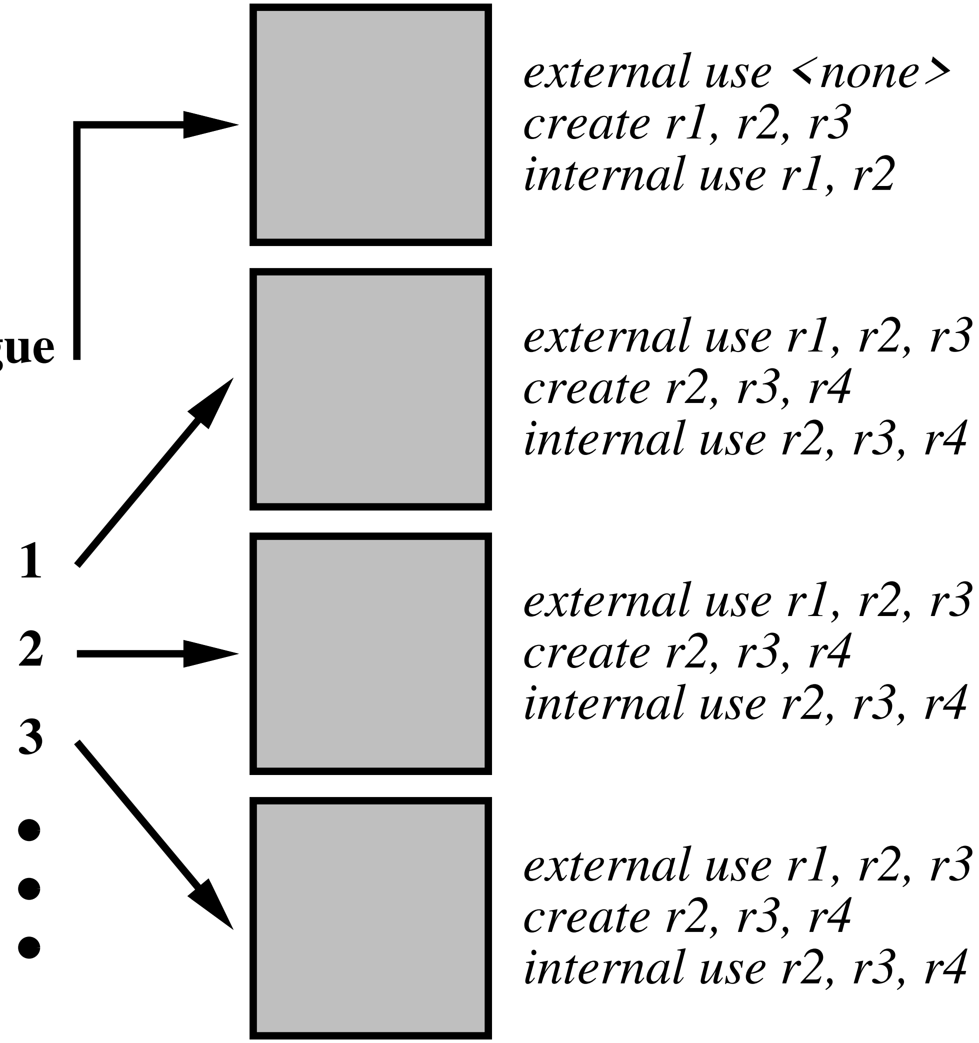
loop body

Serial Execution

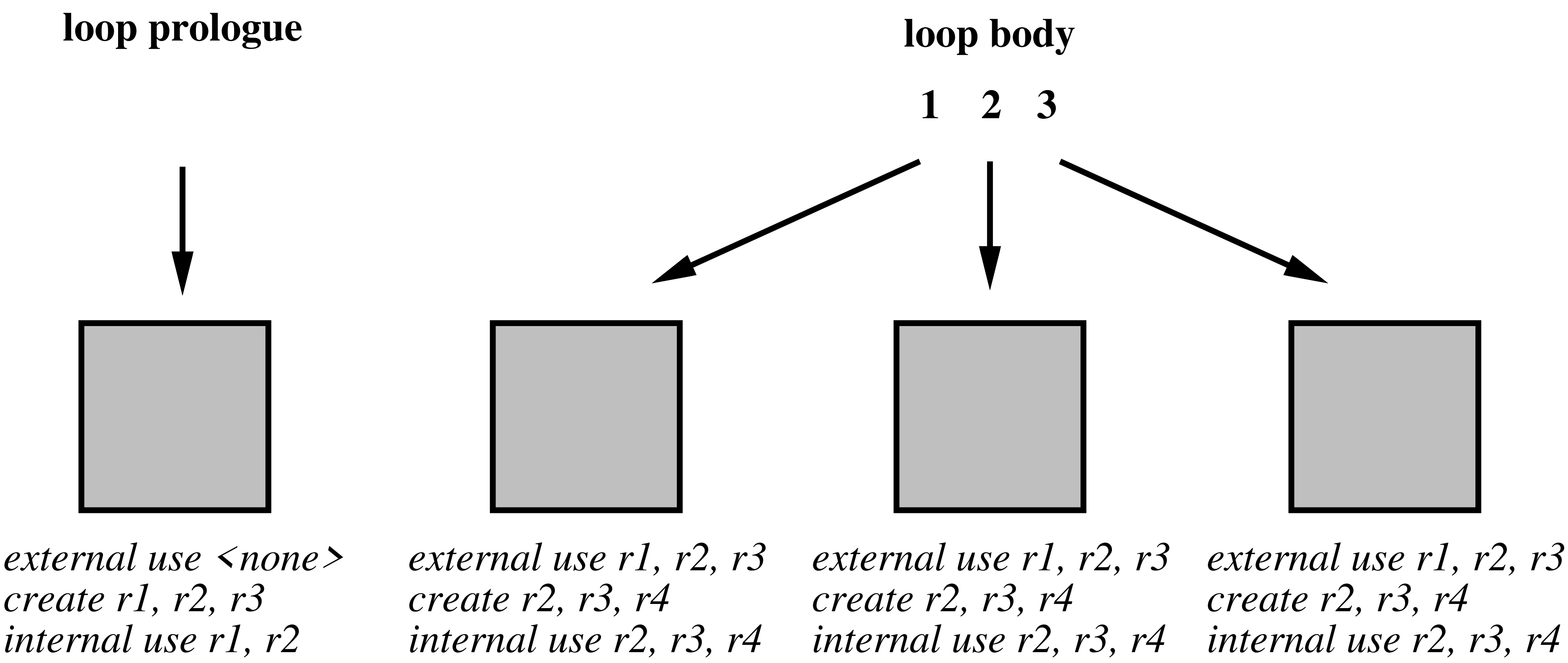
```
        move   r1, n
        move   r2, 0
        move   r3, 0
        bge    r2, r1, done
loop:
        add    r2, r2, 4
        lw     r4, –4(r2)
        ble    r4, r3, test
        move   r3, r4
test:
        blt    r2, r1, loop
done:
```

loop prologue

loop body



Parallel Execution



Loop Invariant : r1

loop prologue

move	**r1, n**	
move	**r2, 0**	
move	**r3, 0**	
bge	**r2, r1, done**	

external use <none>
create r1
internal use r1

loop body 1

loop:		
	add	**r2, r2, 4**
	lw	**r4, –4(r2)**
	ble	**r4, r3, test**
	move	**r3, r4**
test:		
	blt	**r2, r1, loop**
done:		

external use r1
create <none>
internal use <none>

loop body 2

loop:		
	add	**r2, r2, 4**
	lw	**r4, –4(r2)**
	ble	**r4, r3, test**
	~~**move**~~	~~**r3, r4**~~
test:		
	blt	**r2, r1, loop**
done:		

external use r1
create <none>
internal use <none>

loop body 3

loop:		
	add	**r2, r2, 4**
	lw	**r4, –4(r2)**
	ble	**r4, r3, test**
	~~**move**~~	~~**r3, r4**~~
test:		
	blt	**r2, r1, loop**
done:		

external use r1
create <none>
internal use <none>

Non-Loop Carried : r4

loop prologue

```
move    r1, n
move    r2, 0
move    r3, 0
bge     r2, r1, done
```

external use <none>
create <none>
internal use <none>

loop body 1

```
loop:
        add     r2, r2, 4
        lw      r4, -4(r2)
        ble     r4, r3, test
        move    r3, r4
test:
        blt     r2, r1, loop
done:
```

external use <none>
create r4
internal use r4

loop body 2

```
loop:
        add     r2, r2, 4
        lw      r4, -4(r2)
        ble     r4, r3, test
        move    r3, r4
test:
        blt     r2, r1, loop
done:
```

external use <none>
create r4
internal use r4

loop body 3

```
loop:
        add     r2, r2, 4
        lw      r4, -4(r2)
        ble     r4, r3, test
        move    r3, r4
test:
        blt     r2, r1, loop
done:
```

external use <none>
create r4
internal use r4

Loop Carried Induction : r2

loop prologue

move r1, n
move r2, 0
move r3, 0
bge r2, r1, done

external use <none>
create r2
internal use r2

loop body 1

loop:
add r2, r2, 4
lw r4, –4(r2)
ble r4, r3, test
move r3, r4
test:
blt r2, r1, loop
done:

external use r2
create r2
internal use r2

loop body 2

loop:
add r2, r2, 4
lw r4, –4(r2)
ble r4, r3, test
~~move r3, r4~~
test:
blt r2, r1, loop
done:

external use r2
create r2
internal use r2

loop body 3

loop:
add r2, r2, 4
lw r4, –4(r2)
ble r4, r3, test
~~move r3, r4~~
test:
blt r2, r1, loop
done:

external use r2
create r2
internal use r2

Loop Carried Non-Induction : r3

loop prologue

move	**r1, n**
move	**r2, 0**
move	**r3, 0**
bge	**r2, r1, done**

external use <none>
create r3
internal use <none>

loop body 1

loop:		
	add	**r2, r2, 4**
	lw	**r4, –4(r2)**
	ble	**r4, r3, test**
	move	**r3, r4**
test:		
	blt	**r2, r1, loop**
done:		

external use r3
create r3
internal use <none>

loop body 2

loop:		
	add	**r2, r2, 4**
	lw	**r4, –4(r2)**
	ble	**r4, r3, test**
	~~**move**~~	~~**r3, r4**~~
test:		
	blt	**r2, r1, loop**
done:		

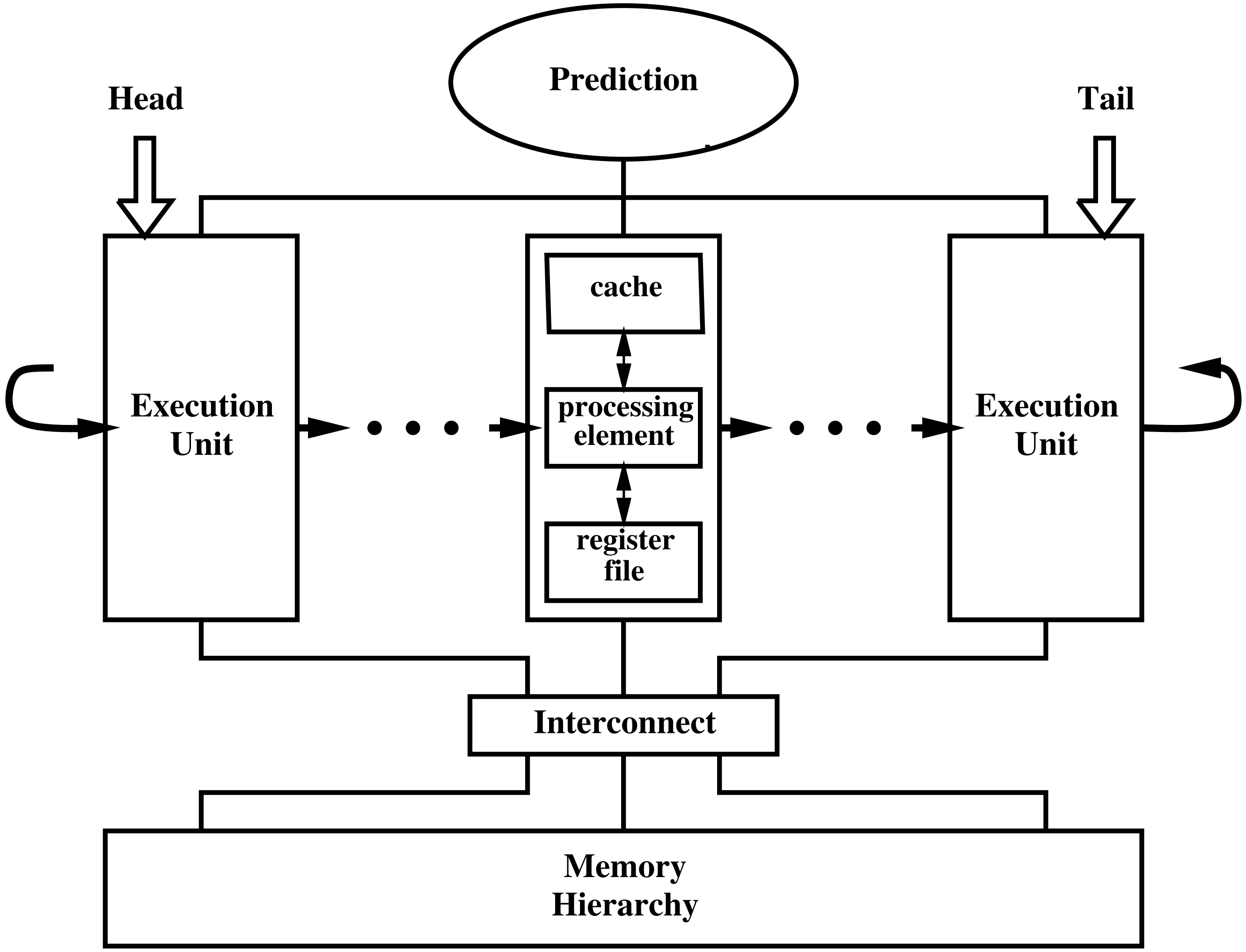
external use r3
create <none>
internal use <none>

loop body 3

loop:		
	add	**r2, r2, 4**
	lw	**r4, –4(r2)**
	ble	**r4, r3, test**
	~~**move**~~	~~**r3, r4**~~
test:		
	blt	**r2, r1, loop**
done:		

external use r3
create <none>
internal use <none>

Multiscalar Organization



Multiscalar Execution Model

- Partition Control Flow Graph Into Tasks
- Execute Task Per Execution Unit
- Enforce Dependences Between Tasks
- Transparent Recovery From Speculation

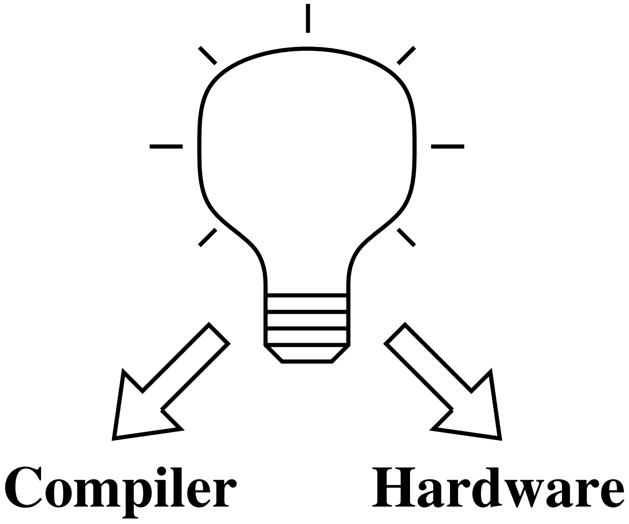
Parallel Execution Performance With Serial Execution Semantics

Basic Idea

Logically...
Single Common Register File

Physically...
Multiple Distributed Register Files, Queues, and Control Logic

Solution



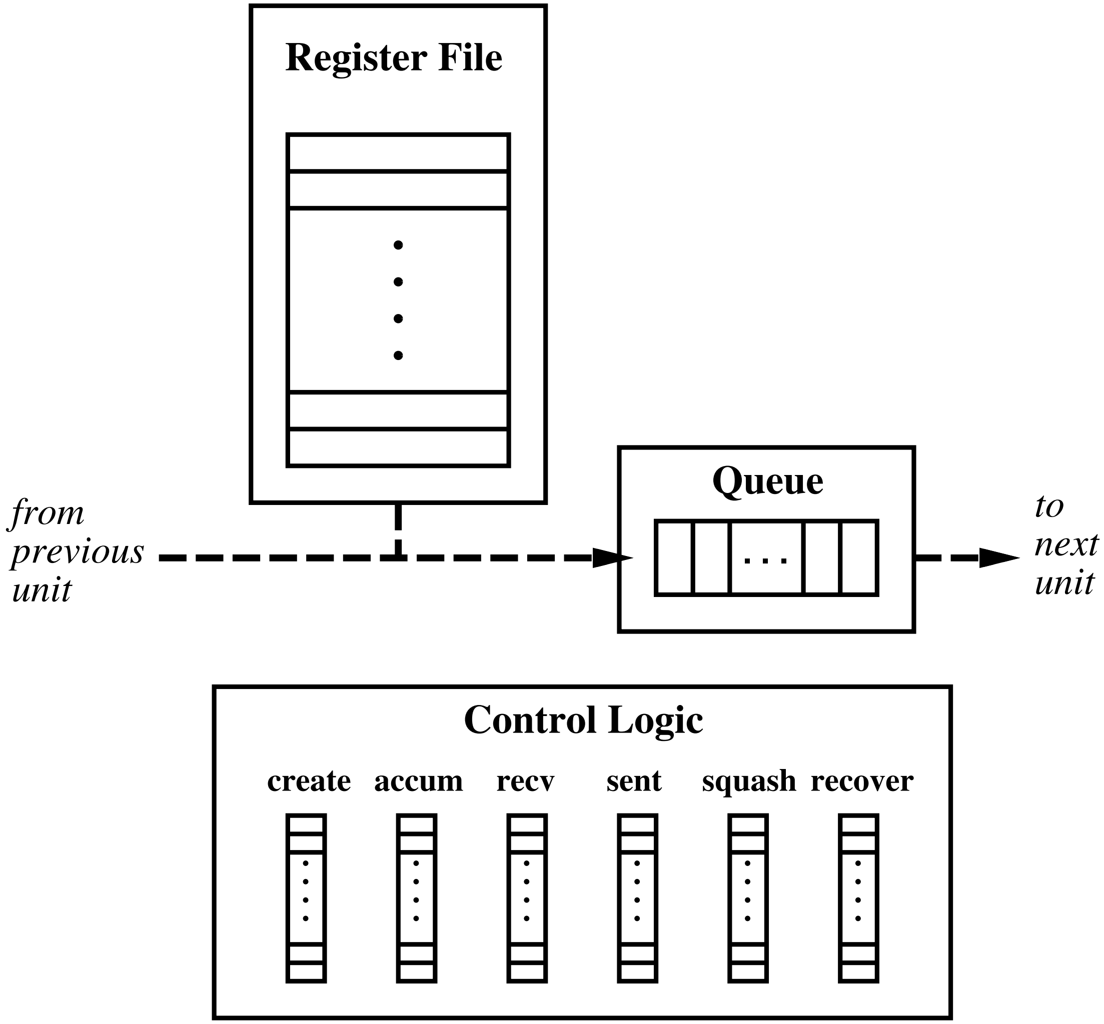
Compiler Streamlines Hardware

Key Issues

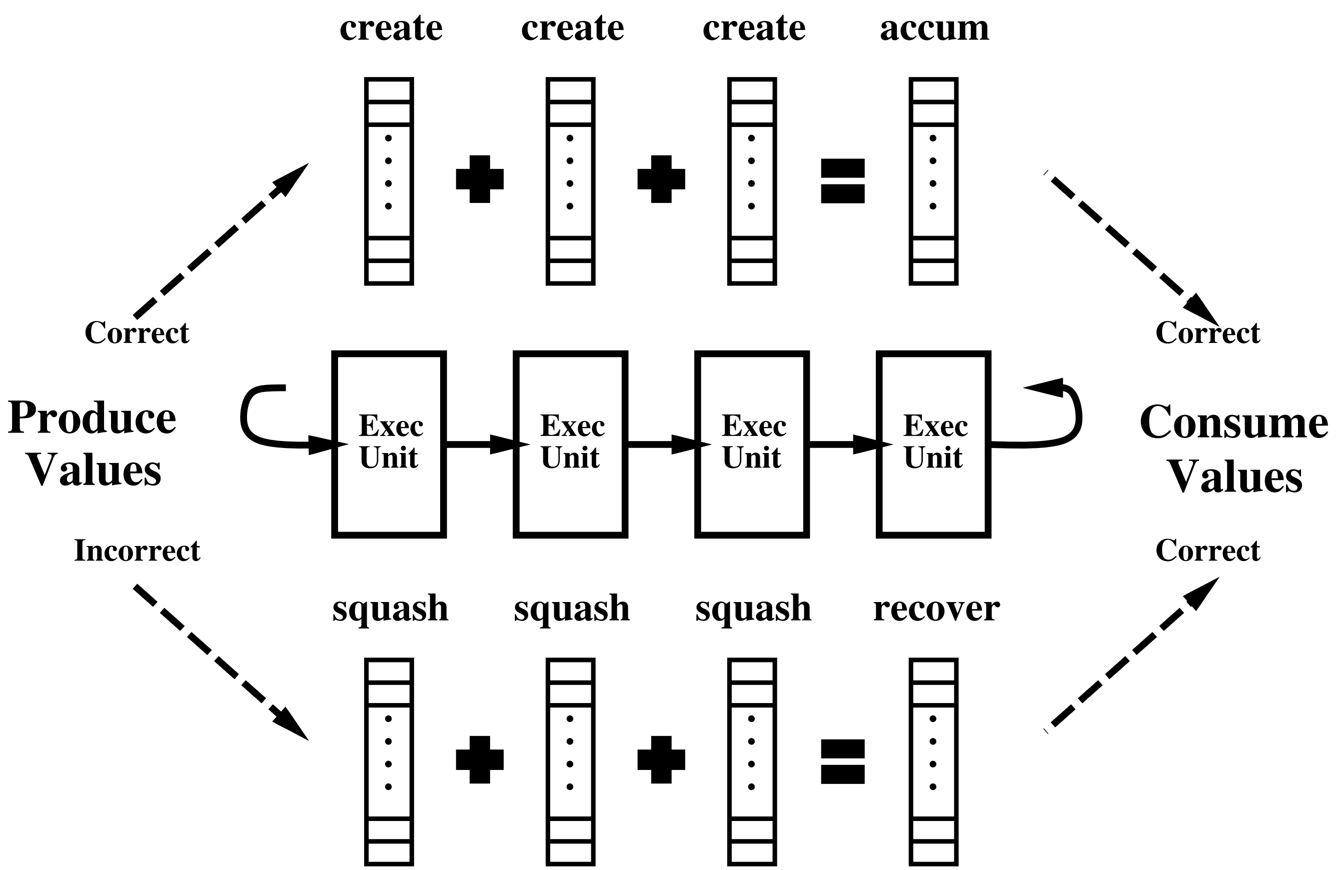
- Storage
- Communication
- Synchronization

$\boxed{\text{Complication of Speculation}}$

Hardware Components



Enforce Register Dependence



Compiler Support

No Control Flow

create

r1	1
r2	1
r3	1
r4	0

```
        move    r1, n           fwd
        move    r2, 0           fwd
        move    r3, 0           fwd
        bge     r2, r1, done
loop:
```

loop prologue

Control Flow

create

r1	0
r2	1
r3	1
r4	1

```
loop:
        add     r2, r2, 4       fwd
        lw      r4, –4(r2)      fwd
        ble     r4, r3, test
        move    r3, r4          fwd
test:
        release r3
        blt     r2, r1, loop
done:
```

loop body

Simple Analysis of Control Flow Graph

Evaluation

benchmark	inst count (millions)		scalar	multiscalar					
				4 unit			8 unit		
	scalar	multi	ipc	ipc	spdup	% bw	ipc	spdup	% bw
compress	71	82	0.58	1.24	1.84	22.4	1.37	2.04	22.4
eqntott	1077	1328	0.72	1.95	2.19	22.7	2.82	3.16	18.5
espresso	526	655	0.78	1.36	1.40	15.2	1.67	1.72	12.8
sc	404	470	0.67	1.03	1.33	9.8	1.26	1.65	7.9
cmp	1	1	0.87	2.88	2.98	6.3	4.96	5.15	11.1
wc	1	1	0.75	2.31	2.64	29.7	4.17	4.77	26.9
tomcatv	2097	2118	0.66	1.73	2.59	11.8	2.57	3.85	9.0