Compiler Directed Early Register Release

Timothy M. Jones, Michael F.P. O'Boyle
University of Edinburgh

Jaume Abella†‡, Antonio González †‡, Oğuz Ergin†
†Intel Labs, ‡Universitat Politècnica de Catalunya

19^{th} September 2005

School of informatics

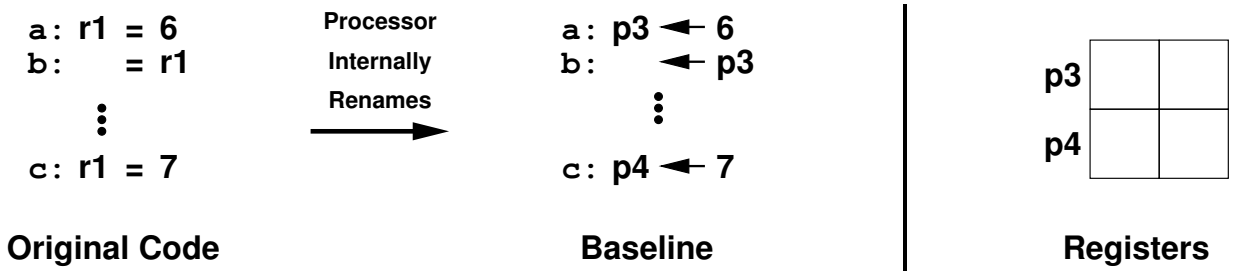
Overview

- The Problem
- Previous Approaches
- Compiler Analysis
- Microarchitecture Changes
- Results
- Conclusions

The Problem

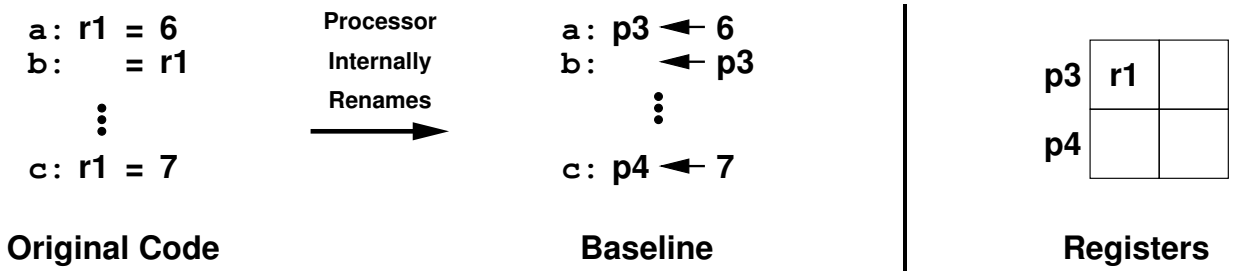
- The register file's performance is critical
- One of the components with the highest power density
  - Expensive cooling systems
  - Power of each generation of processor is increasing non-linearly
- Registers can be idle for many cycles after last consumer issues
  - Butts and Sohi (ISCA 2004) estimate 47% of their allocated cycles

Motivation - Baseline Case



Dispatch	Issue	Execute	Writeback	Commit

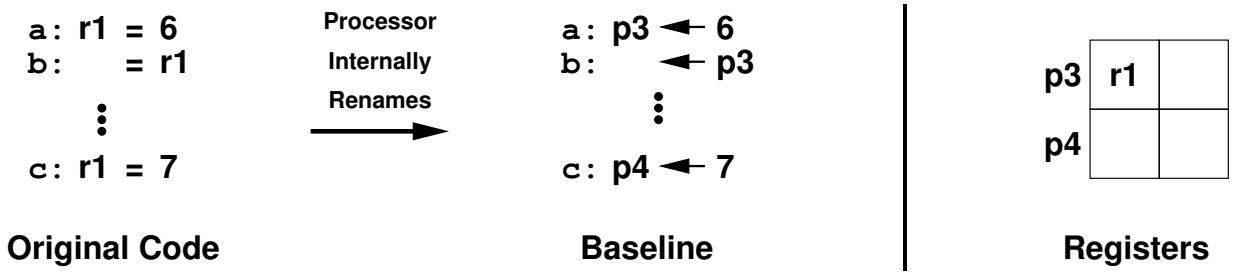
Motivation - Baseline Case



a allocates p3

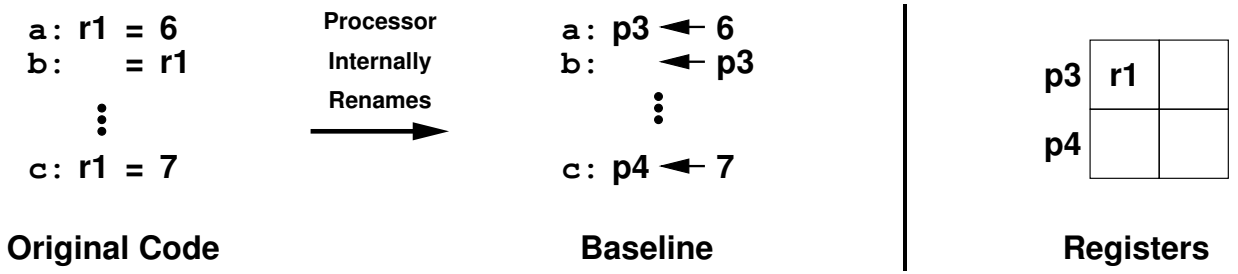
Dispatch	Issue	Execute	Writeback	Commit
a				

Motivation - Baseline Case



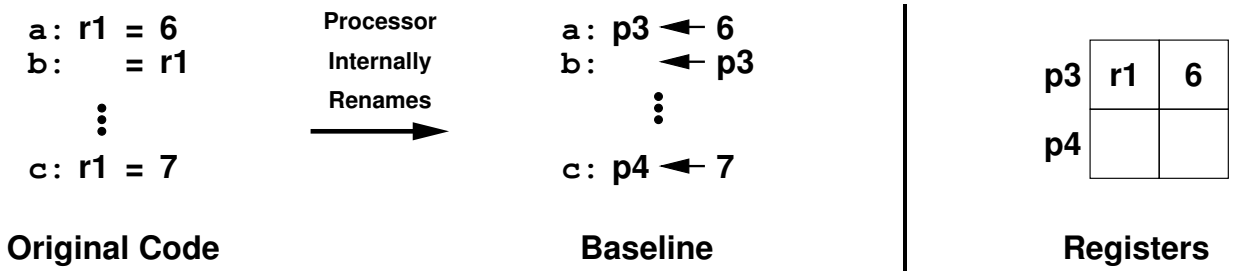
Dispatch	Issue	Execute	Writeback	Commit
	a			

Motivation - Baseline Case



Dispatch	Issue	Execute	Writeback	Commit
		a		

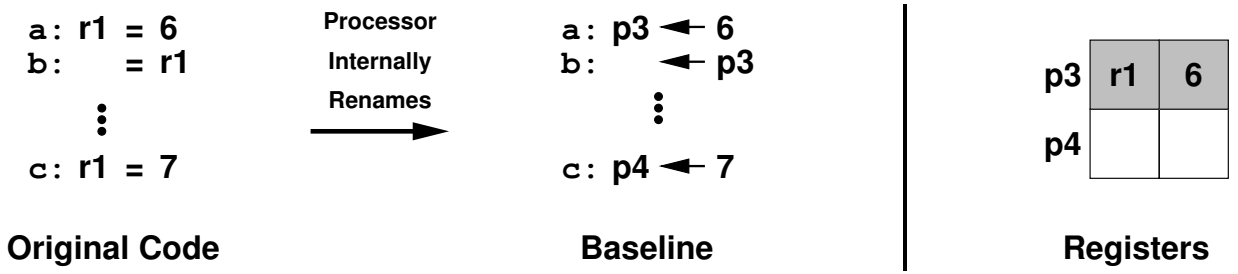
Motivation - Baseline Case



a writes to p3

Dispatch	Issue	Execute	Writeback	Commit
b			a	

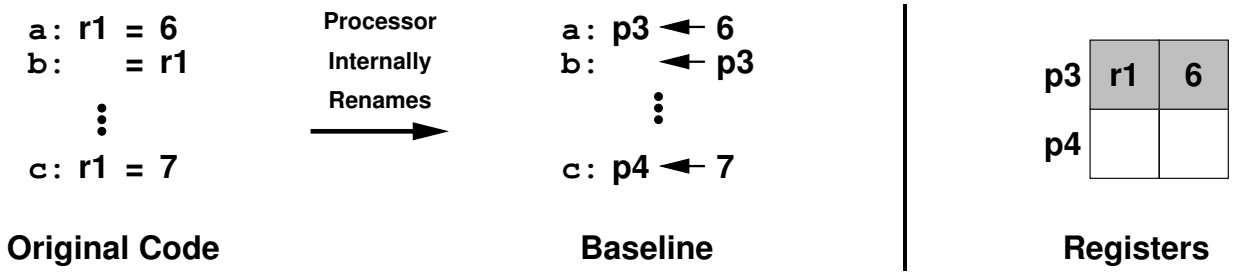
Motivation - Baseline Case



b reads p3, value now no longer needed

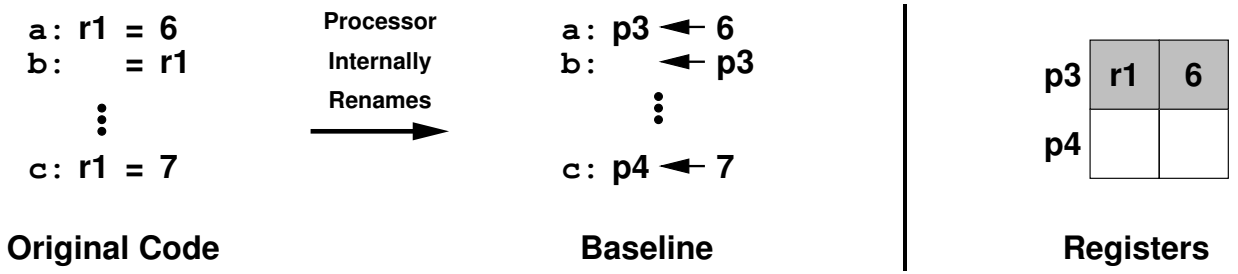
Dispatch	Issue	Execute	Writeback	Commit
	b			a

Motivation - Baseline Case



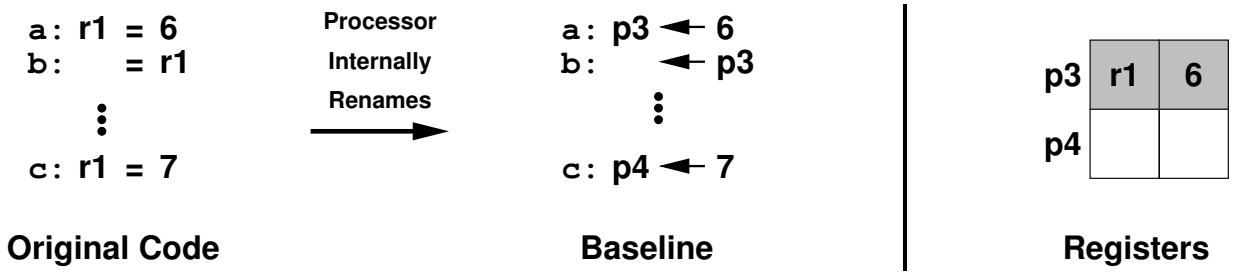
Dispatch	Issue	Execute	Writeback	Commit
		b		

Motivation - Baseline Case



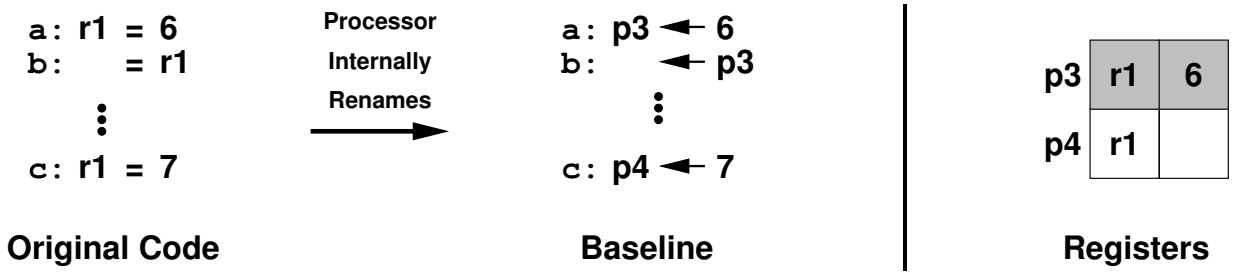
Dispatch	Issue	Execute	Writeback	Commit
			b	

Motivation - Baseline Case



Dispatch	Issue	Execute	Writeback	Commit
				b

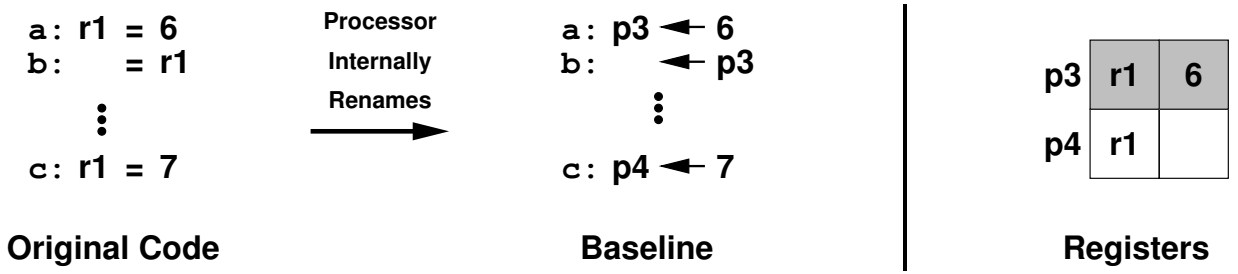
Motivation - Baseline Case



c dispatches many cycles later and allocates p4

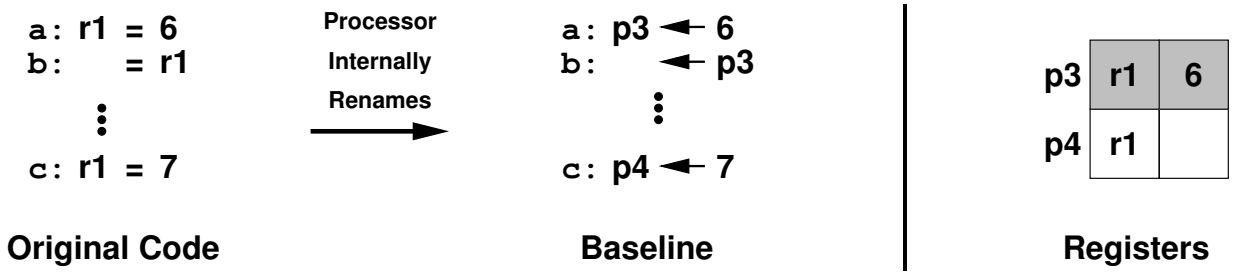
Dispatch	Issue	Execute	Writeback	Commit
c				

Motivation - Baseline Case



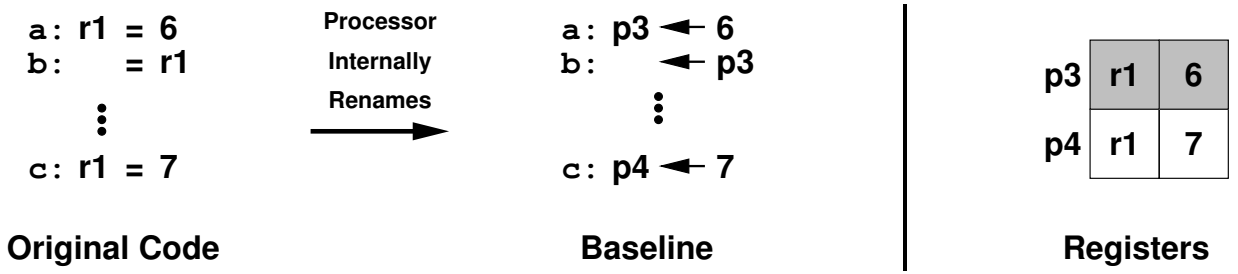
Dispatch	Issue	Execute	Writeback	Commit
	c			

Motivation - Baseline Case



Dispatch	Issue	Execute	Writeback	Commit
		c		

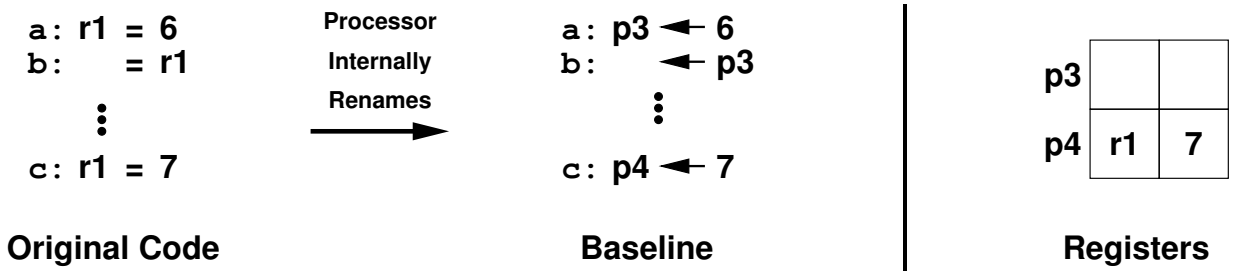
Motivation - Baseline Case



c writes to p4

Dispatch	Issue	Execute	Writeback	Commit
			c	

Motivation - Baseline Case



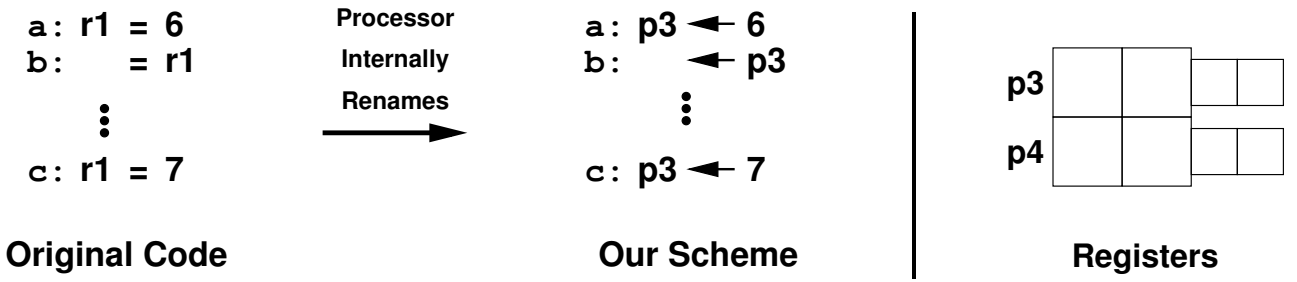
c releases p3, the previous version of r1

Dispatch	Issue	Execute	Writeback	Commit
				c

Motivation - Our Scheme

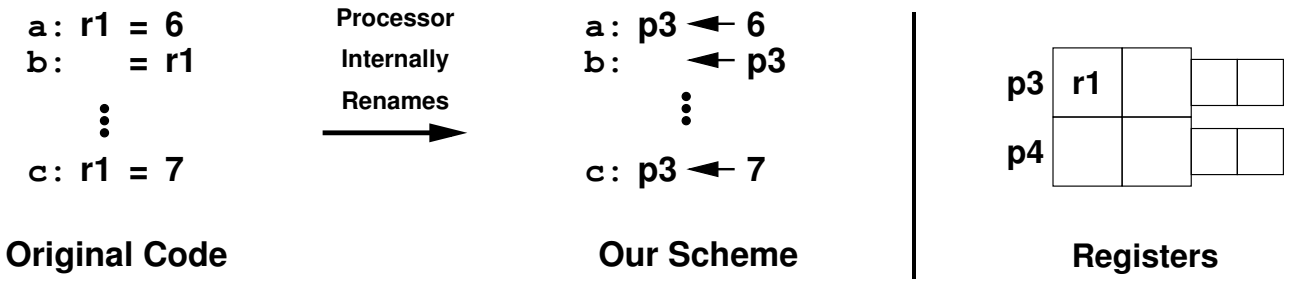
- Release single-use registers much earlier
  - Once the only consumer has issued
- Save a copy of the register values
  - Using checkpointed register file from Ergin *et al.* (2004)
    - Minimal space and energy overhead
  - Precise processor state can always be recovered
- Backup released by the redefiner
  - Instead of the main register

Motivation - Our Scheme



Dispatch	Issue	Execute	Writeback	Commit

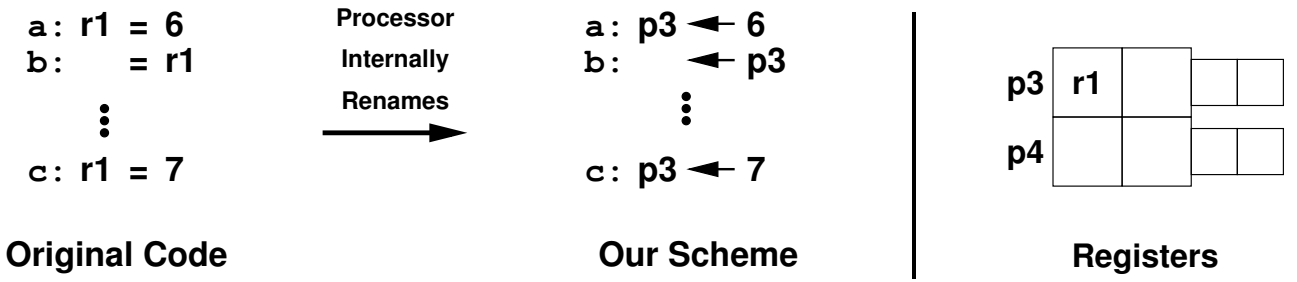
Motivation - Our Scheme



a allocates p3 as before

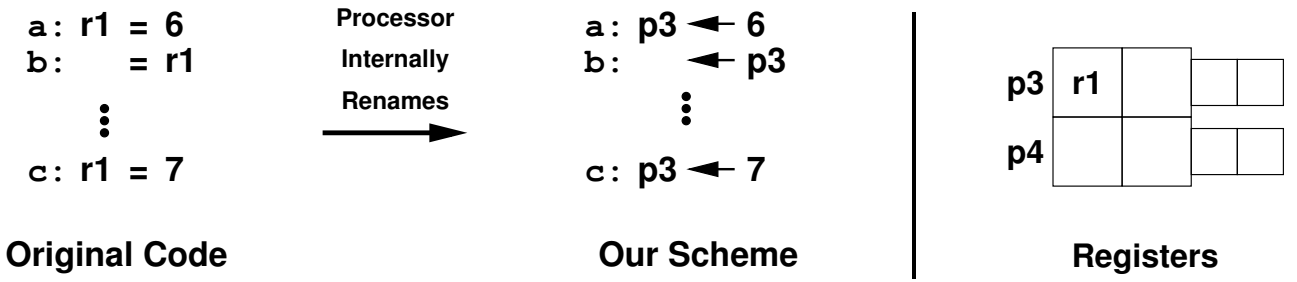
Dispatch	Issue	Execute	Writeback	Commit
a				

Motivation - Our Scheme



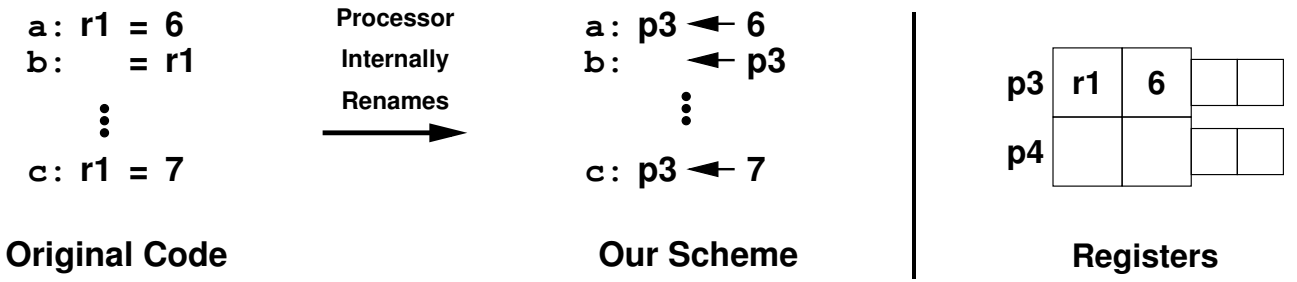
Dispatch	Issue	Execute	Writeback	Commit
	a			

Motivation - Our Scheme



Dispatch	Issue	Execute	Writeback	Commit
		a		

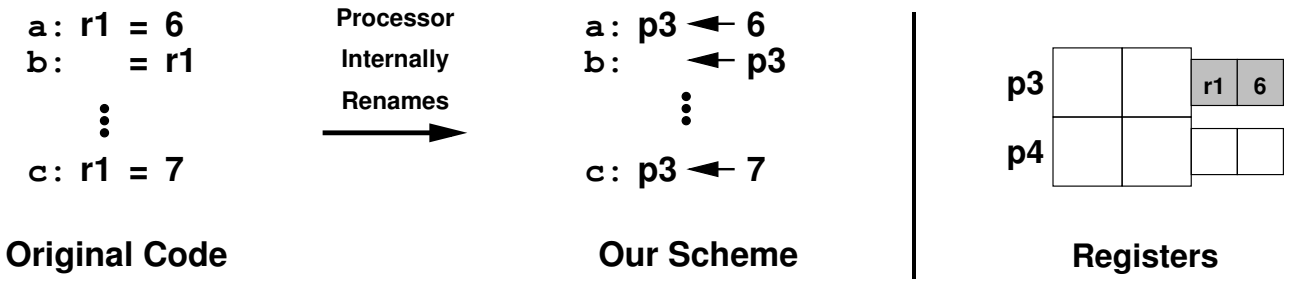
Motivation - Our Scheme



a writes to p3

Dispatch	Issue	Execute	Writeback	Commit
b			a	

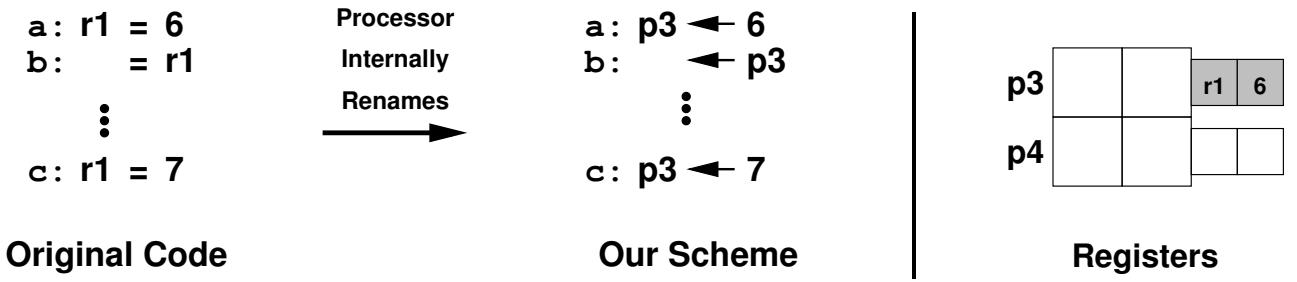
Motivation - Our Scheme



b reads p3, saves a backup copy and frees the register

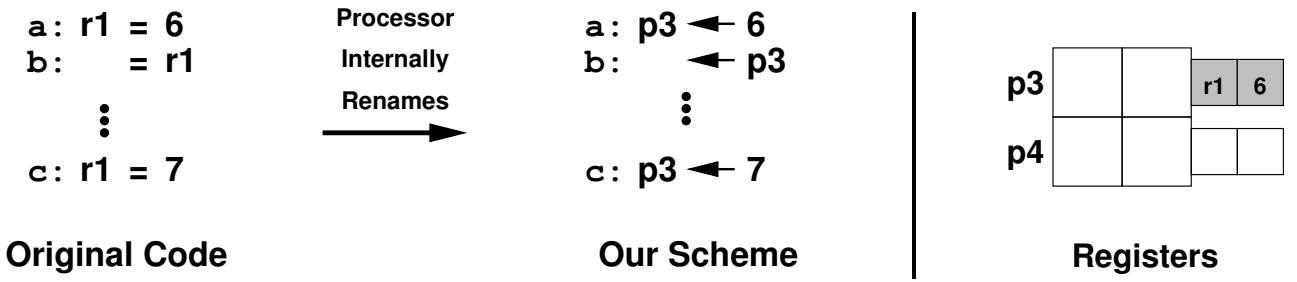
Dispatch	Issue	Execute	Writeback	Commit
	b			a

Motivation - Our Scheme



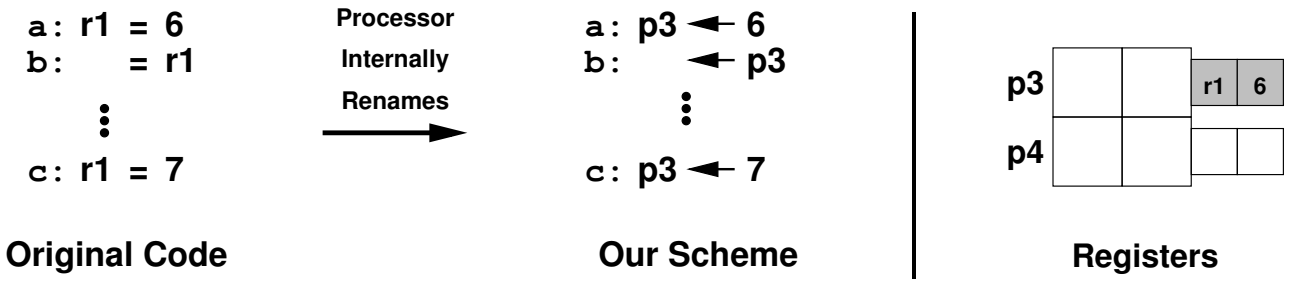
Dispatch	Issue	Execute	Writeback	Commit
		b		

Motivation - Our Scheme



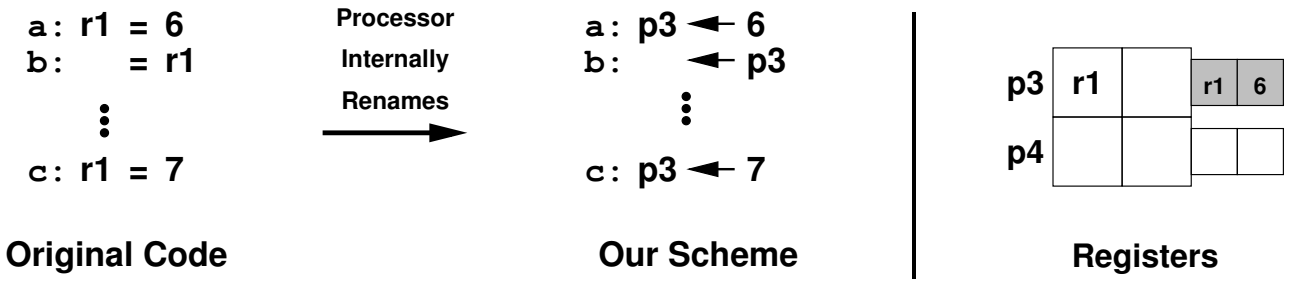
Dispatch	Issue	Execute	Writeback	Commit
			b	

Motivation - Our Scheme



Dispatch	Issue	Execute	Writeback	Commit
				b

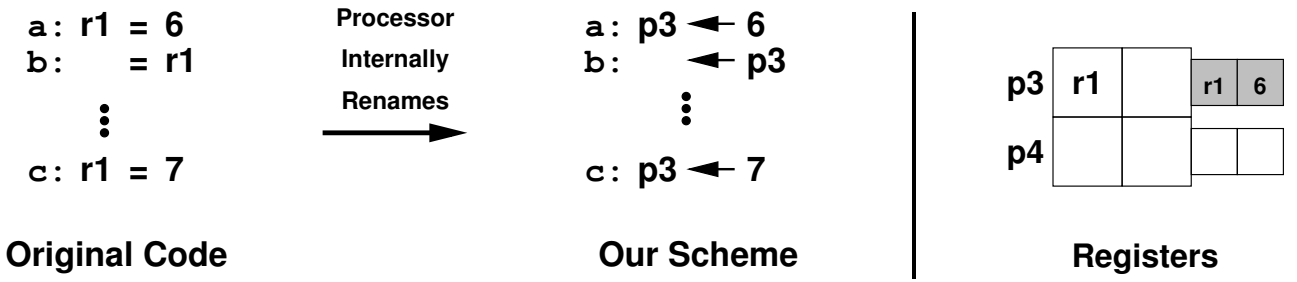
Motivation - Our Scheme



c dispatches many cycles later and can allocate p3

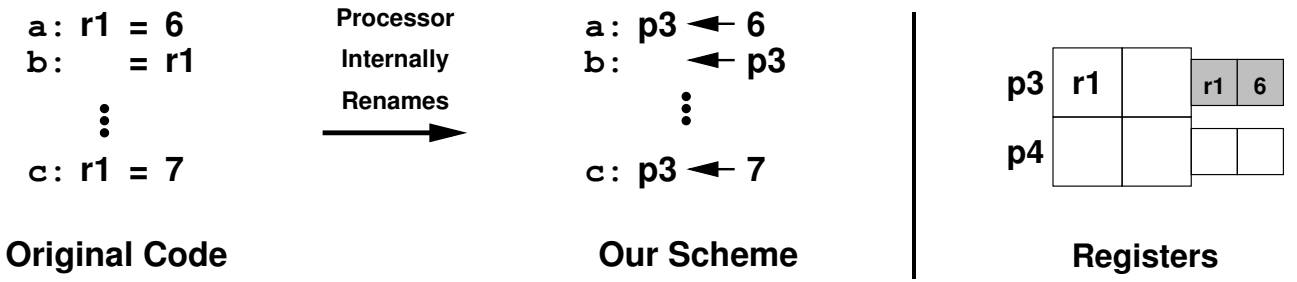
Dispatch	Issue	Execute	Writeback	Commit
c				

Motivation - Our Scheme



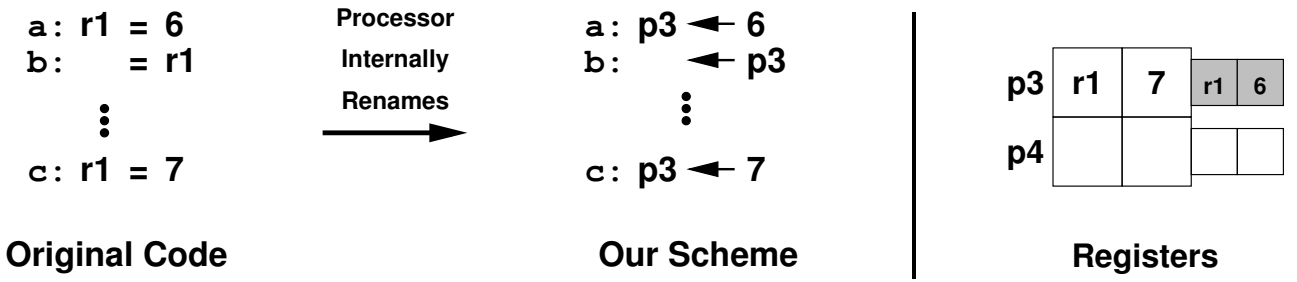
Dispatch	Issue	Execute	Writeback	Commit
	c			

Motivation - Our Scheme



Dispatch	Issue	Execute	Writeback	Commit
		c		

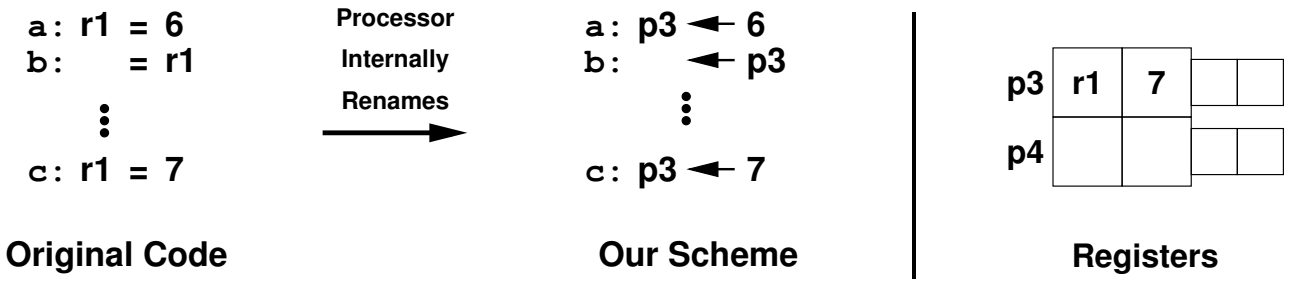
Motivation - Our Scheme



c writes to p3

Dispatch	Issue	Execute	Writeback	Commit
			c	

Motivation - Our Scheme



c releases the backup copy of r1

Dispatch	Issue	Execute	Writeback	Commit
				c

Motivation - Summary

- Our scheme releases single-use registers early
  - When the only consumer issues
- A copy of the register values are saved
  - Using checkpointed register file, Ergin *et al.* (2004)
  - Precise exceptions and interrupts are maintained
- Redefiner releases the backup if the register was released early
  - Instead of the main register
- Our scheme uses fewer registers

Hardware Approaches

- Wait until redefiner enters the pipeline
- Speculative early releasing could help
  - Backup copy needs to be kept
  - Increases the register file complexity
- Not all schemes implement precise exceptions

Hardware Approaches

- Monreal *et al.* (ICPP 2002)
  - Release registers when last user commits
  - And redefiner is non-speculative
  - No precise interrupts or exceptions
- Ergin *et al.* (ICCD 2004)
  - Propose the checkpointed register file
  - Release registers if all users have issued
    - When defining instruction commits
    - Or redefiner dispatches
  - Release registers even if redefiner is speculative

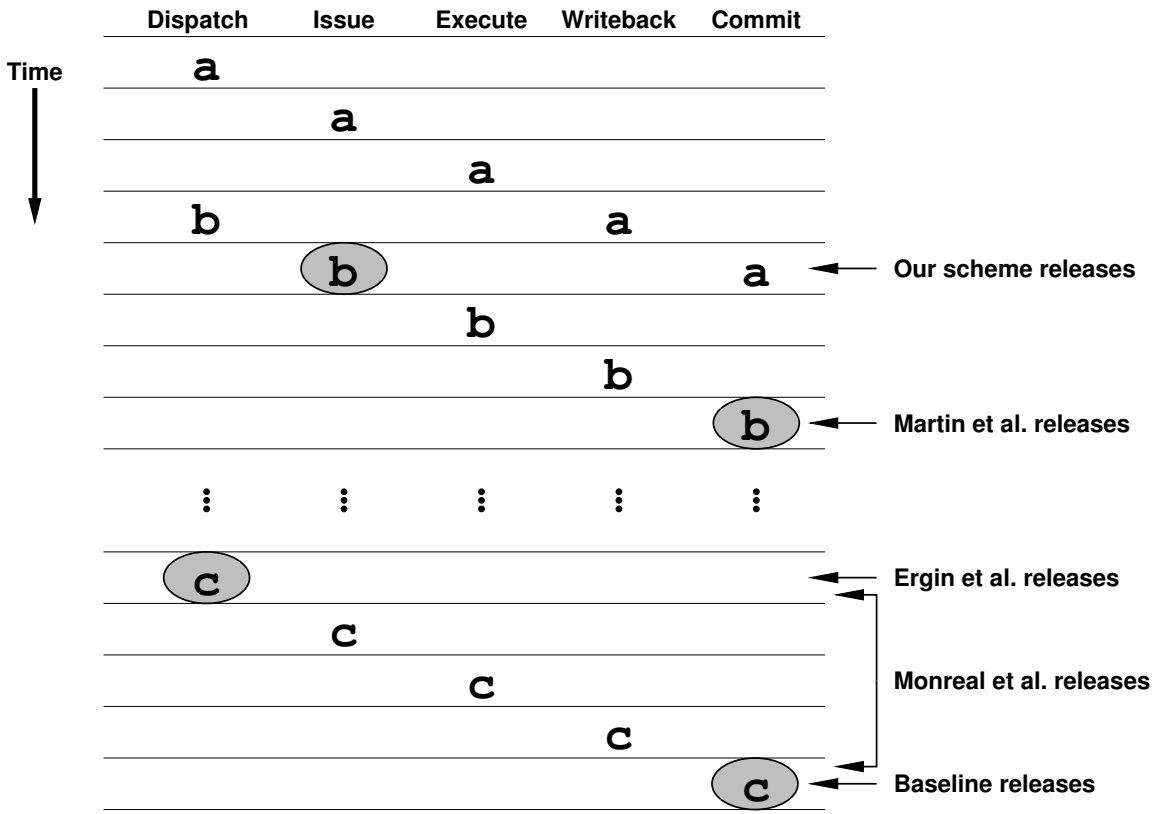
Software Approaches

- Various ways of early releasing
  - Through special instructions
  - Special versions of existing instructions
- Do not implement precise exceptions
- Martin *et al.* (MICRO 1997)
  - Communicate dead value information to processor
  - Explicitly through instructions before procedure calls
  - Implicitly on execution of a call or return
    - We also use this

Our Approach

- Compiler identifies single-use registers
  - Cruz *et al.* (ISCA 2000) estimate 88% of integer registers
- Renames to special register names
- Processor recognises they will be used just once
- Releases register early after consumer issues
  - Much earlier than other schemes
- Stored as backup in case of exception

Comparison of Schemes



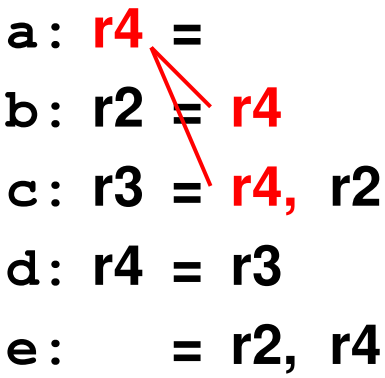
Compiler Analysis

- Based on simple data-flow and liveness analysis
- Rename to virtual registers
  - To distinguish between different versions of each register
- Identify single-use registers
  - Register used just once along every control path from producer
- Create interference graph
- Rename to single-use registers, not register allocation

Example - Original Instructions

```
a: r4 =
b: r2 = r4
c: r3 = r4, r2
d: r4 = r3
e:    = r2, r4
```

Example - Multi-use r4



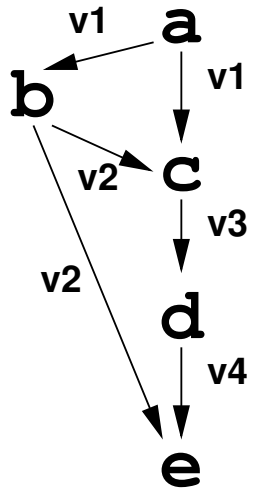
Example - Single-use r4

```
a: r4 =
b: r2 = r4
c: r3 = r4, r2
d: r4 = r3
e:    = r2, r4
```

- Next Step: Separate out single-use and multi-use cases
  - By renaming to virtual registers

Example - Data Dependency Graph

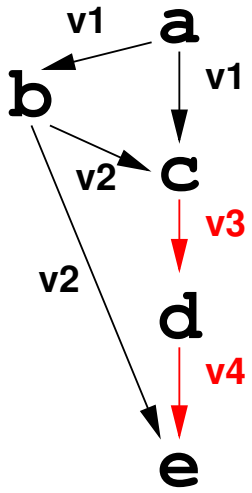
```
a: r4 =                 a: v1 =
b: r2 = r4              b: v2 = v1
c: r3 = r4, r2          c: v3 = v1, v2
d: r4 = r3              d: v4 = v3
e:    = r2, r4          e:    = v2, v4
```



- Next Step: Identify single-use registers

Example - Single-use registers

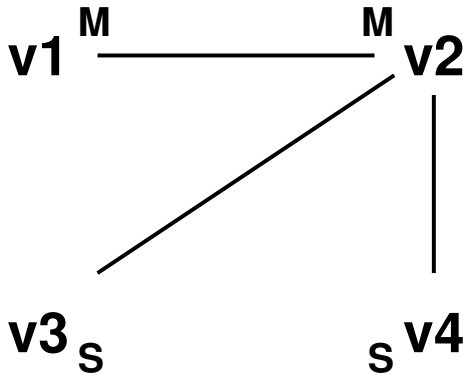
```
a: r4 =                 a: v1 =
b: r2 = r4              b: v2 = v1
c: r3 = r4, r2          c: v3 = v1, v2
d: r4 = r3              d: v4 = v3
e:    = r2, r4          e:    = v2, v4
```

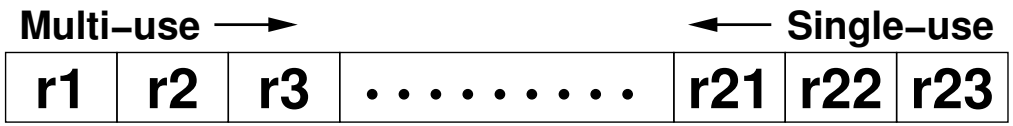


- Next Step: Create interference graph and rename to logical registers

Example - Interference Graph

```
a: r4 =                a: v1 =
b: r2 = r4             b: v2 = v1
c: r3 = r4, r2         c: v3 = v1, v2
d: r4 = r3             d: v4 = v3
e:    = r2, r4         e:    = v2, v4
```





Example - Renamed Instructions

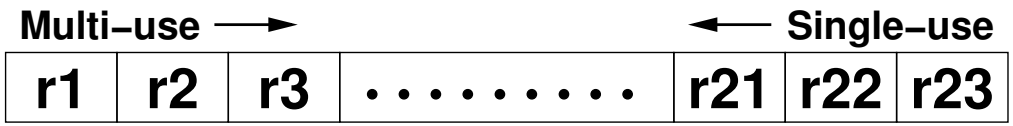
```
a: r4 =
b: r2 = r4
c: r3 = r4, r2
d: r4 = r3
e:    = r2, r4
```

Original Code - 3 Registers

v2	r1
v1	r2
v3	r23
v4	r23

```
a: r2  =
b: r1  = r2
c: r23 = r2, r1
d: r23 = r23
e:     = r1, r23
```

New Code - 3 Registers

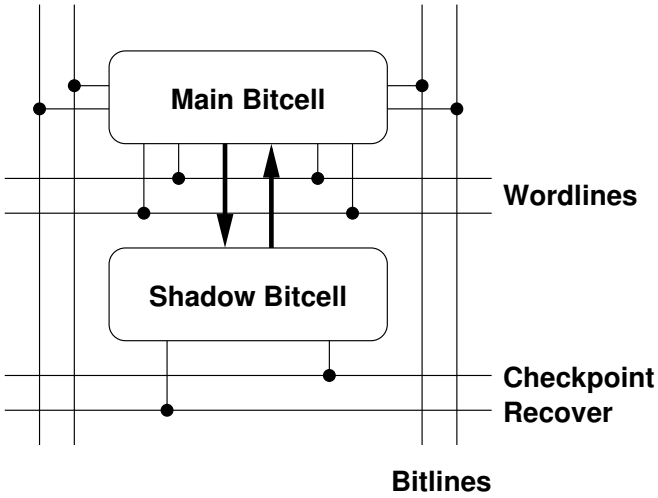


Microarchitecture Changes

- Compiler identifies single-use registers
- Processor can now recognise these too
- Needs to take advantage of them
  - Needs mechanism for releasing early
  - Needs to implement precise interrupts and exceptions
- No ISA impact

Microarchitecture Changes

- Checkpointed register file
  - Ergin *et al.* (ICCD 2004)
  - Area overhead 19.4%
  - Delay overhead only 0.5%
  - Energy overhead only 2%
  - Precise interrupts and exceptions
- Banks of 8 registers
  - Can be independently turned off to save energy when empty



Microarchitecture Changes

- Register file
  - New *checkpointed* bit for each register
- Reorder buffer
  - An extra *early_release* bit per source operand
  - An extra *did_early_release* bit per source operand
- Map tables
  - New *early_release* bit for each register in dispatch table
  - New *checkpointed* bit for each register in retirement table

Early Releasing

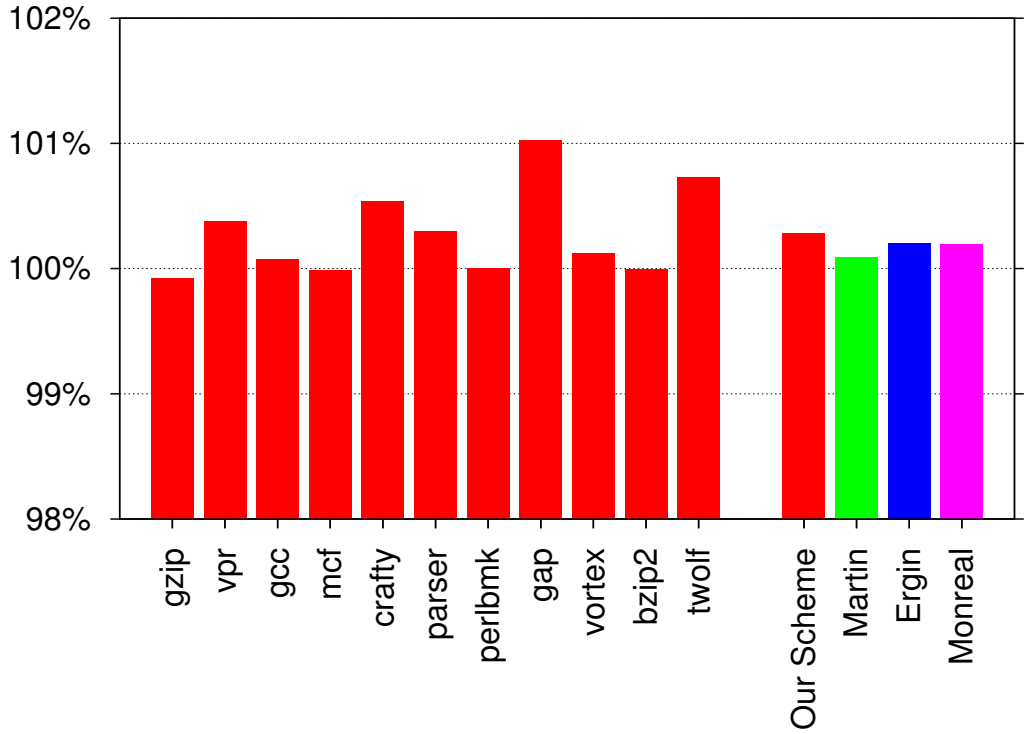
- Dispatch
  - Copy *early_release* bits from map table to reorder buffer
- Issue
  - Read register file *checkpointed* bits in parallel with data
  - If unset and *early_release* bit is set, checkpoint register
  - Set register's *checkpointed* bit and relevant *did_early_release* bit
- Commit
  - Release previous version register or *checkpointed* bit
  - Copy *did_early_release* bits to retirement map table

Experimental Setup

- Compiler
  - MachineSUIF from Harvard
  - SUIF2 from Stanford
- Processor (8-way)
  - 128 entry reorder buffer, 80 entry issue queue
  - Varying register file sizes
  - Wattch and SimpleScalar
- Benchmarks
  - SpecINT 2000 (not 252.eon)

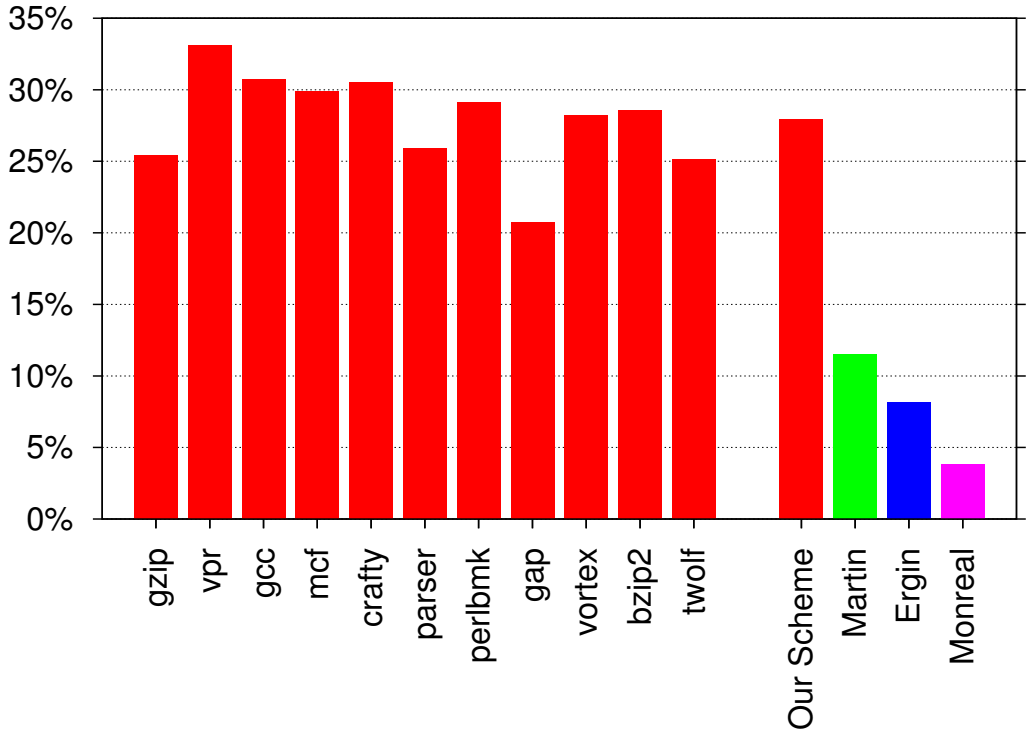
Performance 112 Registers

- Negligible improvement
- No scheme has much of an effect



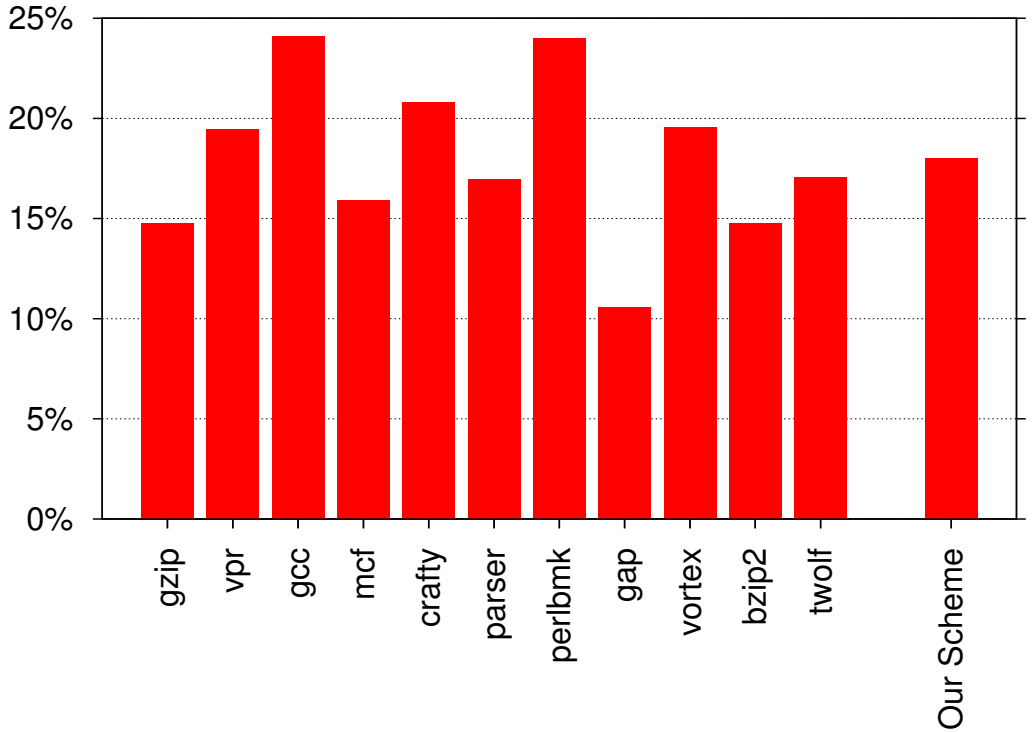
Occupancy Reduction 112 Registers

- Greater reduction in register pressure than other schemes
  - Average 28% reduction
  - Martin is 11%
  - Ergin is 8%
  - Monreal is 4%
- Directly affects energy consumption due to banking scheme used



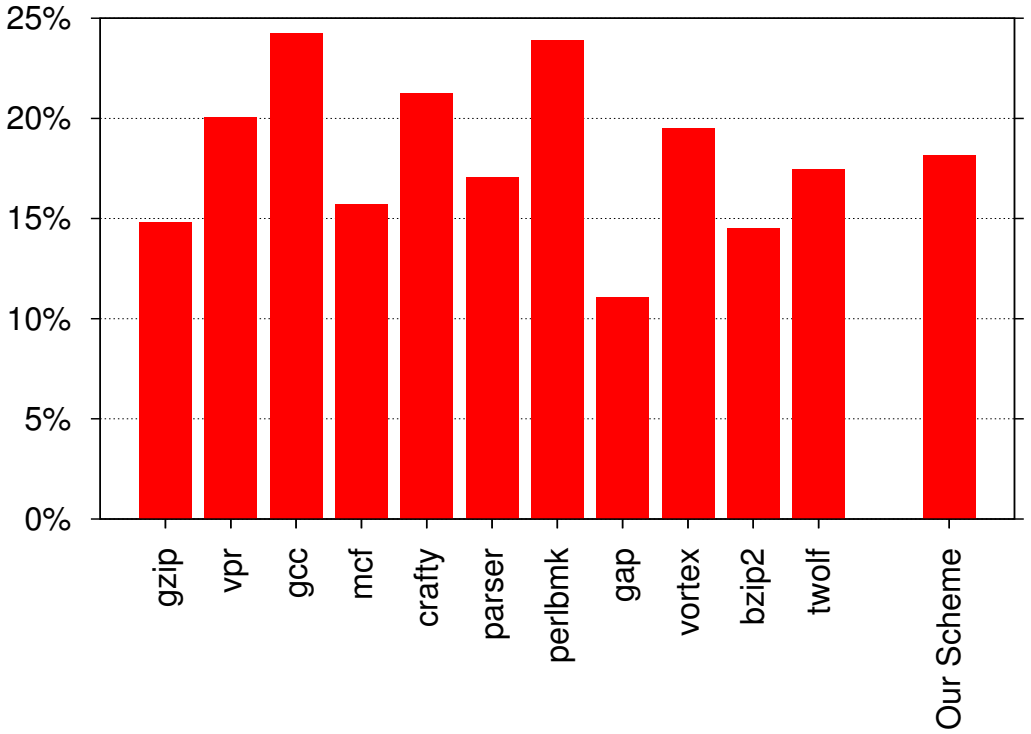
Dynamic Energy Savings 112 Registers

- No values for other schemes
- Energy reduction of 18% on average
  - Savings of 24% for gcc



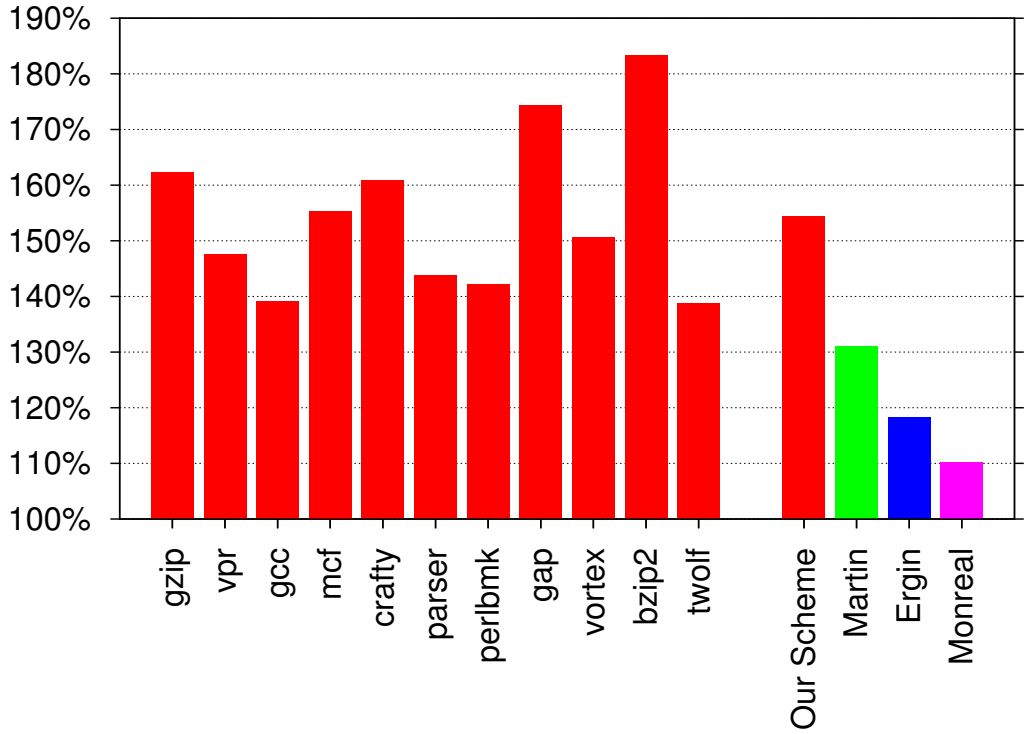
Static Energy Savings 112 Registers

- Average 18% reduction
  - Most savings for gcc with 24%



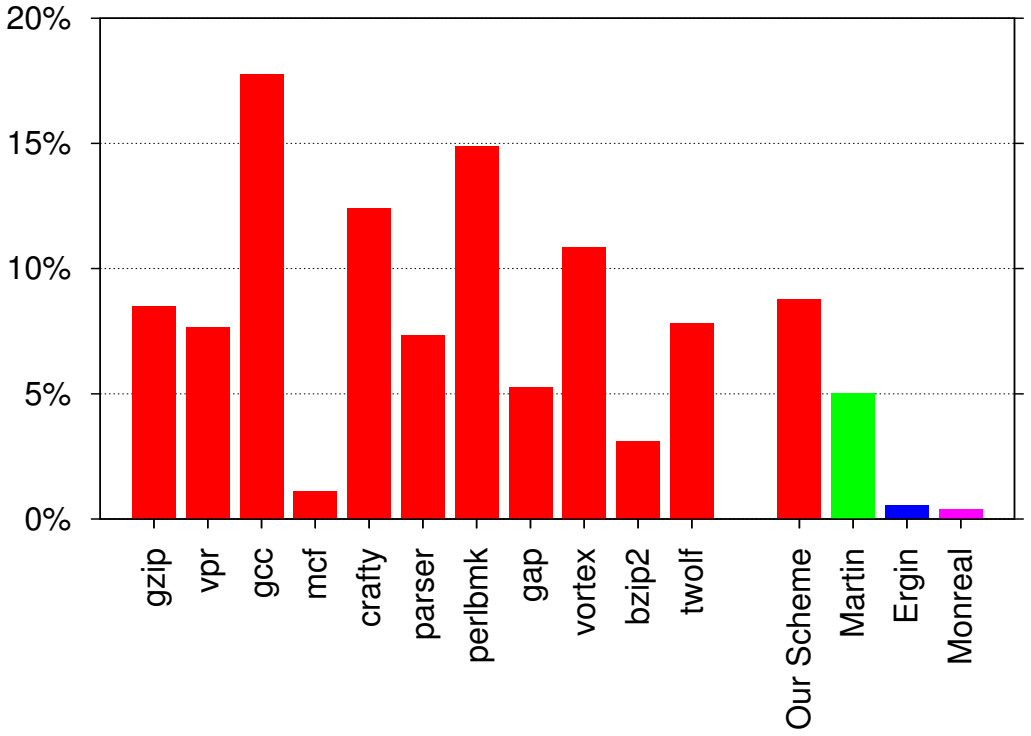
Performance 40 Registers

- Out-performs other schemes
  - Our approach is 54% faster
  - Martin is 31% faster
  - Ergin is 18% faster
  - Monreal is 10% faster
- Best is bzip2, 83% faster



Occupancy 40 Registers

- Smaller reduction in register pressure
  - Average 9% reduction
  - Martin has 5% reduction
  - Ergin and Monreal have less than 1% reduction
- Energy savings
  - Increase of 3% dynamic
  - Savings of 34% static



Conclusions

- Simple compiler analysis to rename single-use registers
- Release registers at issue
  - Much earlier than other schemes
- Large reductions in register pressure with many registers
  - Means large energy savings
- Large performance increases with few registers
- Out-performs all other hardware and software schemes

Future Work

- Look at communicating two-use values
- Consider other times registers can be released early
- Look at other register file optimisations
  - Can the compiler indicate short bit-width values?
  - How can the compiler help with register caching schemes?
- Focus on other parts of the processor