

6.035

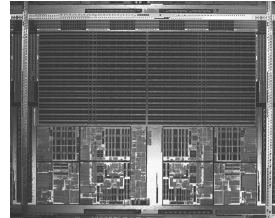
Fall 2006

Project 5: Optimizer

Michael Gordon



AMD Opteron 270



- ~233 million transistors
- 90nm
- 2.0 GHz
- Dual Core
- 2 Processors per board
- 64 KB L1 Ins Cache
- 64 KB L2 data cache
- 2 MB on-chip L2 cache
- 12 Integer pipeline stages

Michael Gordon

2

6.035 ©MIT Fall 2006

Opteron's Key Features

- Multiple issue (3 instructions per cycle)
- Register renaming
- Out-of-order execution
- Branch prediction
- Speculative execution
- Load/Store buffering
- Dual core

Michael Gordon

3

6.035 ©MIT Fall 2006

What is a Superscalar?

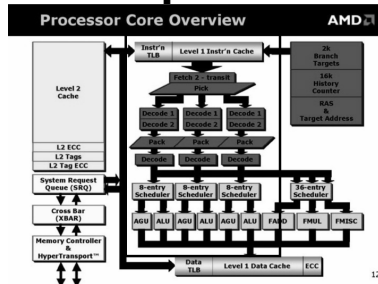
- Any (scalar) processor that can "execute" more than one instruction per cycle.
- Multiple instructions can be fetched per cycle
- Decode logic can decide which instructions are independent
 - Decide when an instruction is ready to fire
- Multiple functional units
- All this for instruction-level parallelism!

Michael Gordon

4

6.035 ©MIT Fall 2006

Multiple Issue



- The Opteron is a 3-way superscalar:
 - decode & execute & retire 3 x86-64 instructions per cycle

Michael Gordon

5

6.035 ©MIT Fall 2006

Dependencies

- Remember there are 3 types of dependencies:
 - True dependence
 - read after write (RAW)
 - Anti-dependence
 - write after read (WAR)
 - Output dependence
 - write after write (WAW)

Michael Gordon

6

6.035 ©MIT Fall 2006

Register Renaming

- Used to eliminate artificial dependencies
 - Anti-dependence (WAR)
 - Output dependence (WAR)
- Basic rule
 - All instructions that are in-flight write to a distinct destination register

Michael Gordon

7

6.035 ©MIT Fall 2006

Register Renaming

- Registers of x86-64 ISA (%rax, %rsi, %r11, etc) are logical registers
- When an instruction is decoded, its logical register destination is assigned to a physical register
- The total number of physical (rename) registers is:
 - $\text{instructions_inflight} + \text{architectural regs} =$
 - $72 + 16 = 96$

Michael Gordon

8

6.035 ©MIT Fall 2006

Register Renaming

- Hardware maintains a mapping between the logical regs and the rename regs
 - Reorder Buffer (72 entries)
- Architectural state is in 16 entry register file
 - 2 entries for each logical register
 - One speculative and one committed
 - Maintains architectural visible state
 - Used for exceptions and miss-prediction

Michael Gordon

9

6.035 ©MIT Fall 2006

Out-of-Order Execution

- The Opteron can also execute instructions out of program order
 - Respect the data-flow (RAW) dependencies between instructions
- Hardware schedulers executes an instruction whenever all of its operands are available
 - Not program order, dataflow order!
- The Reorder Buffer orders instructions back into program order

Michael Gordon

10

6.035 ©MIT Fall 2006

Out-of-Order Execution

- When an instruction is placed in the reorder buffer:
 - Get its operands
 - if no in-flight writer of operand, get it from architectural speculative register
 - if in-flight writer, get writer's ID from state and wait for value to be produced
 - Rename dest register, update state to remember this is most recent writer of dest
- Reorder Buffer:
 - Forward results of instruction to consumers in reorder buffer
 - Write value to speculative architectural register

Michael Gordon

11

6.035 ©MIT Fall 2006

Branch Prediction

- The outcome of a branch is known at the end of the pipeline
- We don't have to have to stall the pipeline while the branch is being resolved
- Use local and global information to predict the next PC for each instruction
 - Return address stack for ret instruction
- Extremely accurate for integer codes
 - >95% in studies
- Pipeline continues with the predicted PC as its next PC

Michael Gordon

12

6.035 ©MIT Fall 2006

Speculative Execution

- Continue execution of program with predicted next PC for conditional branches
- An instruction can commit its result to architectural registers
 - Only after every prior instruction has committed
- 3 instructions can be retired per cycle

Michael Gordon

13

6.035 ©MIT Fall 2006

Speculative Execution

- Each instruction that is about to commit is checked:
 - Make sure that no exceptions have occurred
 - Make sure that no branch miss-predictions have occurred (for conditional branches)
- If a miss-prediction or an exception occurs
 - Flush the pipeline
 - Clear the reorder buffer and schedulers
 - Forget any speculative architectural registers
 - Forward the correct PC to the fetch unit for miss-prediction
 - Call exception handler for exception

Michael Gordon

14

6.035 ©MIT Fall 2006

Main Memory

- It is very expensive to go to main memory!
- Optimize for the memory hierarchy
- Keep needed values as close to the processor as possible
- Your control:
 - 16 Logical Regs
 - Main Memory
- Under the hood
 - Rename Regs -> Logical Regs -> L1 -> L2 -> Memory

Michael Gordon

15

6.035 ©MIT Fall 2006

Load/Store Buffering

- 3 cycle latency for loads that hit in L1 d-cache
- Load/Store Buffer with 12 entries
 - Handles memory requests
 - Entry for each load/store
 - They wait until their address is calculated
 - Loads check the L/S buffer before accessing the
 - If address is in the L/S buffer, get the value from that entry
 - Otherwise, the oldest load probes the cache
 - Stores just wait for their value to be ready
 - Cannot write to cache because it might be speculative
 - Stores must be retired to write to memory
- Complex mechanisms for maintaining program order of loads and stores

Michael Gordon

16

6.035 ©MIT Fall 2006

Four Cores!

- Dual cores per processor, dual processor per board
- You can map different iterations of a forpar loop to different cores
 - Data-level parallelism
- Memory coherence is maintained by a global cache-coherence mechanism
 - Snooping mechanism
 - All cores see same state of d-cache
- More on this in the future!

Michael Gordon

17

6.035 ©MIT Fall 2006

Example

```
//load array's last addr into %rdi
loop:
  mov (%rdi), %r10
  add %r11, %r10
  mov %r10, (%rdi)
  sub $8, %rdi
  bge loop
```

Michael Gordon

18

6.035 ©MIT Fall 2006

Example

Load/Store Buffer

Architectural State			ID	Op	Addr	Value
Reg	Committed	Speculative	1			
%rdi	80		2			
%r10			3			
%r11	5		4			
			5			
			6			

ID	Op	Dest	Src1	Src2
1				
2				
3				
4				
5				
6				
7				
8				

ID	Op	Dest	Src1	Src2
9				
10				
11				
12				
13				
14				
15				
16				

```

loop:
mov(%rdi), %r10
add %r11, %r10
mov %r10, (%rdi)
sub $8, %rdi
bge loop
    
```

Michael Gordon 19 6.035 ©MIT Fall 2006

Example

Load/Store Buffer

Architectural State			ID	Op	Addr	Value
Reg	Committed	Speculative	1			
%rdi	80		2			
%r10			3			
%r11	5		4			
			5			
			6			

ID	Op/Val	Dest	Src1	Src2
1				
2				
3				
4				
5				
6				
7				
8				

ID	Op	Dest	Src1	Src2
9				
10				
11				
12				
13				
14				
15				
16				

```

loop:
mov(%rdi), %r10
add %r11, %r10
mov %r10, (%rdi)
sub $8, %rdi
bge loop
    
```

Michael Gordon 20 6.035 ©MIT Fall 2006

Example

Load/Store Buffer

Architectural State			ID	Op	Addr	Value
Reg	Committed	Speculative	1			
%rdi	80		2	st	80	RB2
%r10			3			
%r11	5		4			
			5			
			6			

ID	Op/Val	Dest	Src1	Src2
1	mov	r10		LS1
2	add	r10	RB1	r11
3	mov		LS2	RB2
4	sub	rdi	rdi	\$8
5	bge	loop	RB4	
6				
7				
8				

ID	Op	Dest	Src1	Src2
9				
10				
11				
12				
13				
14				
15				
16				

```

loop:
mov(%rdi), %r10
add %r11, %r10
mov %r10, (%rdi)
sub $8, %rdi
bge loop
    
```

Michael Gordon 21 6.035 ©MIT Fall 2006

Example

Load/Store Buffer

Architectural State			ID	Op	Addr	Value
Reg	Committed	Speculative	1			
%rdi	80		2	st	80	RB2
%r10			3			
%r11	5		4			
			5			
			6			

ID	Op/Val	Dest	Src1	Src2
1	mov	r10		LS1
2	add	r10	RB1	r11
3	mov	LS2	RB2	
4	sub	rdi	rdi	\$8
5	bge	loop	RB4	
6	mov	r10		LS3
7	add	r10	RB6	r11
8	mov	LS4	RB7	

ID	Op	Dest	Src1	Src2
9	sub	rdi	RB4	\$8
10	bge	loop	RB9	
11				
12				
13				
14				
15				
16				

```

loop:
mov(%rdi), %r10
add %r11, %r10
mov %r10, (%rdi)
sub $8, %rdi
bge loop
    
```

Michael Gordon 22 6.035 ©MIT Fall 2006

Example: "Cycle 1"

Load/Store Buffer

Architectural State			ID	Op	Addr	Value
Reg	Committed	Speculative	1			
%rdi	100		2	st	80	RB2
%r10			3	ld	RB4	??
%r11	5		4	st	RB4	RB7
			5	ld	RB9	??
			6	st	RB9	RB12

ID	Op/Val	Dest	Src1	Src2
1	mov	r10		LS1
2	add	r10	RB1	r11
3	mov	LS2	RB2	
4	sub	rdi	rdi	\$8
5	bge	loop	RB4	
6	mov	r10		LS3
7	add	r10	RB6	r11
8	mov	LS4	RB7	

ID	Op	Dest	Src1	Src2
9	sub	rdi	RB4	\$8
10	bge	loop	RB9	
11	mov	r10		LS5
12	add	r10	RB11	r11
13	mov	LS6	RB12	
14	sub	rdi	RB9	\$8
15	bge	loop	RB14	
16				

```

loop:
mov(%rdi), %r10
add %r11, %r10
mov %r10, (%rdi)
sub $8, %rdi
bge loop
    
```

Michael Gordon 23 6.035 ©MIT Fall 2006

Example: "Cycle 2"

Load/Store Buffer

Architectural State			ID	Op	Addr	Value
Reg	Committed	Speculative	1			
%rdi	80		2	st	80	RB2
%r10			3	ld	RB4	??
%r11	5		4	st	RB4	RB7
			5	ld	RB9	??
			6	st	RB9	RB12

ID	Op/Val	Dest	Src1	Src2
1	mov	r10		LS1
2	add	r10	RB1	r11
3	mov	LS2	RB2	
4	72			
5	bge	loop	RB4	
6	mov	r10		LS3
7	add	r10	RB6	r11
8	mov	LS4	RB7	

ID	Op	Dest	Src1	Src2
9	sub	rdi	RB4	\$8
10	bge	loop	RB9	
11	mov	r10		LS5
12	add	r10	RB11	r11
13	mov	LS6	RB12	
14	sub	rdi	RB9	\$8
15	bge	loop	RB14	
16				

```

loop:
mov(%rdi), %r10
add %r11, %r10
mov %r10, (%rdi)
sub $8, %rdi
bge loop
    
```

Michael Gordon 24 6.035 ©MIT Fall 2006

Example: "Cycle 2"

Load/Store Buffer

Architectural State			ID	Op	Addr	Value
Reg	Committed	Speculative	1			10
%rdi	80	72 (RB4)	2	st	80	RB2
%r10			3	ld	72	??
%r11	5		4	st	72	RB7
			5	ld	RB9	??
			6	st	RB9	RB12

loop:
mov (%rdi), %r10
add %r11, %r10
mov %r10, (%rdi)
sub \$8, %rdi
bge loop

Reorder Buffer					
ID	Op/Val	Dest	Src1	Src2	
1	mov	r10		LS1	
2	add	r10	RB1	r11	
3	mov	LS2	RB2		
4	72				
5	bge	loop	72		
6	mov	r10		LS3	
7	add	r10	RB6	r11	
8	mov	LS4	RB7		

Michael Gordon 25 6.035 ©MIT Fall 2006

Example: "Cycle 2"

Load/Store Buffer

Architectural State			ID	Op	Addr	Value
Reg	Committed	Speculative	1			10
%rdi	80	72 (RB4)	2	st	80	RB2
%r10			3	ld	72	??
%r11	5		4	st	72	RB7
			5	ld	RB9	??
			6	st	RB9	RB12

loop:
mov (%rdi), %r10
add %r11, %r10
mov %r10, (%rdi)
sub \$8, %rdi
bge loop

Reorder Buffer					
ID	Op/Val	Dest	Src1	Src2	
1	mov	r10		LS1	
2	add	r10	RB1	r11	
3	mov	LS2	RB2		
4	72				
5	bge	loop	RB4		
6	mov	r10		LS3	
7	add	r10	RB6	r11	
8	mov	LS4	RB7		

Michael Gordon 26 6.035 ©MIT Fall 2006

Example: "Cycle 2"

Load/Store Buffer

Architectural State			ID	Op	Addr	Value
Reg	Committed	Speculative	1			10
%rdi	80	72 (RB4)	2	st	80	RB2
%r10			3	ld	72	??
%r11	5		4	st	72	RB7
			5	ld	RB9	??
			6	st	RB9	RB12

loop:
mov (%rdi), %r10
add %r11, %r10
mov %r10, (%rdi)
sub \$8, %rdi
bge loop

Reorder Buffer					
ID	Op/Val	Dest	Src1	Src2	
1	mov	r10		10	
2	add	r10	RB1	r11	
3	mov	LS2	RB2		
4	72				
5	bge	loop	RB4		
6	mov	r10		LS3	
7	add	r10	RB6	r11	
8	mov	LS4	RB7		

Michael Gordon 27 6.035 ©MIT Fall 2006

Example: "Cycle 2"

Load/Store Buffer

Architectural State			ID	Op	Addr	Value
Reg	Committed	Speculative	1			10
%rdi	80	72 (RB4)	2	st	80	RB2
%r10			3	ld	72	??
%r11	5		4	st	72	RB7
			5	ld	RB9	??
			6	st	RB9	RB12

loop:
mov (%rdi), %r10
add %r11, %r10
mov %r10, (%rdi)
sub \$8, %rdi
bge loop

Reorder Buffer					
ID	Op/Val	Dest	Src1	Src2	
1	mov	r10		10	
2	add	r10	RB1	r11	
3	mov	LS2	RB2		
4	72				
5	bge	loop	RB4		
6	mov	r10		LS3	
7	add	r10	RB6	r11	
8	mov	LS4	RB7		

Michael Gordon 28 6.035 ©MIT Fall 2006

Example: "Cycle 3"

Load/Store Buffer

Architectural State			ID	Op	Addr	Value
Reg	Committed	Speculative	1			10
%rdi	80	64 (RB9)	2	st	80	RB2
%r10	10 (RB1)		3		9	
%r11	5		4	st	72	RB7
			5	ld	64	??
			6	st	RB9	RB12

loop:
mov (%rdi), %r10
add %r11, %r10
mov %r10, (%rdi)
sub \$8, %rdi
bge loop

Reorder Buffer					
ID	Op/Val	Dest	Src1	Src2	
1	10				
2	add	r10	10	r11	
3	mov	LS2	RB2		
4	72				
5					
6	mov	r10		LS3	
7	add	r10	RB6	r11	
8	mov	LS4	RB7		

Michael Gordon 29 6.035 ©MIT Fall 2006

Example: "Cycle 4"

Load/Store Buffer

Architectural State			ID	Op	Addr	Value
Reg	Committed	Speculative	1			10
%rdi	80	64 (RB9)	2	st	80	15
%r10	15 (RB2)		3		9	
%r11	5		4	st	72	RB7
			5		8	
			6	st	RB9	RB12

loop:
mov (%rdi), %r10
add %r11, %r10
mov %r10, (%rdi)
sub \$8, %rdi
bge loop

Reorder Buffer					
ID	Op/Val	Dest	Src1	Src2	
1	10				
2	15				
3	mov	LS2	15	rdi	
4	72				
5					
6	9				
7	add	r10	9	r11	
8	mov	LS4	RB7		

Michael Gordon 30 6.035 ©MIT Fall 2006

Example: "Cycle 5"

Load/Store Buffer

Architectural State			ID	Op	Addr	Value
Reg	Committed	Speculative	1			10
%rdi	80	64 (RB9)	2			
%r10	15 (RB2)		3			9
%r11	5		4	st	72	14
			5			8
			6	st	RB9	RB12

loop:
 mov(%rdi), %r10
 add %r11, %r10
 mov %r10, (%rdi)
 sub \$8, %rdi
 bge loop

Reorder Buffer

ID	Op/Val	Dest	Src1	Src2	ID	Op/Val	Dest	Src1	Src2
1	10				9	64			
2	15				10				
3					11	mov	r10		8
4	72				12	add	r10	RB11	r11
5					13	mov	LS6	RB12	
6	9				14	56			
7	14				15	bge	loop	56	
8	mov	LS4	14		16	mov	r10	56	LS7

Michael Gordon 31 6.035 ©MIT Fall 2006

Example: Steady State

Instructions Decoded:

```

loop: //iteration 1
ld
movl
add
st
mov2
sub
bge
loop: //iteration 2
...
loop: //iteration 3
...
loop: //iteration 4
...
loop: //iteration 5
...
loop: //iteration 6
...

```

Cycle					
1	2	3	4	5	6
ld	movl	movl	movl	movl	movl
	bge	bge	bge	mov2	mov2
	ld	add	add	bge	bge
	ld	ld	ld	add	add
	sub	sub	st	ld	ld
			sub	st	st
				sub	sub

Instructions from 5 iterations can fire in one cycle

Michael Gordon 32 6.035 ©MIT Fall 2006

Steady-State

- In each cycle:
 - Instructions issued from different iterations of the original loop
- Remember that the Opteron can fire 3 instructions per cycle
 - not 5 like on previous slide
- What does this say about instruction scheduling techniques?
 - List scheduling, trace scheduling, loop unrolling, software pipelining, etc.

Michael Gordon 33 6.035 ©MIT Fall 2006

Conclusions

- Opteron is doing many things under the hood
 - Multiple issue
 - Reordering
 - Speculation
 - Multiple levels of caching
- Your optimizations should compliment what is going on
- Use your "global" knowledge of the program
 - Redundancy removal
 - Register allocation
 - What else?

Michael Gordon 34 6.035 ©MIT Fall 2006

Benchmarking

- We will use total cycle count of the entire benchmark
- tyner and silver have updated kernels that support updating/querying of performance counters
- You can benchmark they entire program and also a section of code you denote
- Use the "benchmark" program in /u/mgordon/6035/bin
 - benchmark a.out

Michael Gordon 35 6.035 ©MIT Fall 2006

Benchmarking Methodology

- Compile the program with your compiler
- Assemble with:


```
gcc4 negative.s -pthread -lpapiex -l6035 -L/u/mgordon/6035/lib64
```
- There is a new library 6.035 in /u/mgordon/6035/lib64
 - if using it, you must use the "benchmark" command to run the application

Michael Gordon 36 6.035 ©MIT Fall 2006

Benchmarking Methodology

- You can also define a caliper in the code
 - A section of code that you would like to know more about
- Wrap the section in:


```
start_caliper()
...
end_caliper()
```

Michael Gordon

37

6.035 ©MIT Fall 2006

Sample Output

```
loop
libmonitor debug: (P20167.Tox0) monitor_fni_process()
PapEx Version: 0.99rc2
Executable: /tmp/gordon/6.035/sampleloop
Processor: AMD K8 Revision C
Clockrate: 1993.465945
Parent Process ID: 20156
Process ID: 20167
Hostname: tyner
Domain: PAPI_TOT_CYC:NO_WRITE
Domain: User
Real cycles: 435
Real cycles: 860899
Proc cycles: 127
Proc cycles: 249782
PAPI_TOT_CYC: 150002

Caliper 1:
Execution: 1
Real cycles: 18
Real cycles: 32333
Proc cycles: 18
Proc cycles: 32328
PAPI_TOT_CYC: 32226

Event descriptions:
Event: PAPI_TOT_CYC
Derived: No
Short Description: Total cycles
Long Description: Total cycles
Developer's Notes:

Start: Wed Nov 15 23:51:48 2006
Finish: Wed Nov 15 23:51:48 2006
libmonitor debug: (P20167.Tox0) monitor_fni_library()
```

- PAPI_TOT_CYC:
- User Cycles!
 - Does not count cycle cycles in kernel/system calls
 - Does not count cycles swapped out by OS scheduler
 - Extremely accurate

- Caliper is given after whole program totals

Michael Gordon

38

6.035 ©MIT Fall 2006

Benchmarking Example

negative.dcf: invert():

```
void invert() {
    callout("start_caliper");
    forpar i = 0, size {
        imageOut[i] = 255 - imageIn[i];
    }
    callout("end_caliper");
}
```

Michael Gordon

39

6.035 ©MIT Fall 2006

Benchmarking Example

```
...
    lea imageIn_0(%rip), %rax
    mov -8(%rbp), %r11
    mov 0(%rax, %r11, 8), %r10
    mov %r10, -96(%rbp)

    mov $255, %r10
    mov %r10, -56(%rbp)

    mov -96(%rbp), %r10
    sub %r10, -56(%rbp)

    lea imageOut_0(%rip), %rax
    mov -8(%rbp), %r11
    mov -56(%rbp), %r10
    mov %r10, 0(%rax, %r11, 8)
...

```

User Cycles:
3,415,072

Michael Gordon

40

6.035 ©MIT Fall 2006

Benchmarking Example

```
    lea imageIn_0(%rip), %rax
    mov -8(%rbp), %r11
    mov 0(%rax, %r11, 8), %r9

    mov $255, %r10
    sub %r9, %r10
    User Cycles
    2,952,540

    lea imageOut_0(%rip), %rax
    mov -8(%rbp), %r11
    mov %r10, 0(%rax, %r11, 8)
```

Michael Gordon

41

6.035 ©MIT Fall 2006

Project Guidelines

- Don't use GCC for experimentation!
- Writeup needs to be very detailed
- Discuss techniques presented in class that are not beneficial when targeting a superscalar
 - Should be in design proposal
- For each optimization:
 - Prove that it is a win by hand applying and benchmarking
 - Give a detailed explanation of the implementation
 - Argue that it is general and correct
- Just Step 1 for Design Proposal

Michael Gordon

42

6.035 ©MIT Fall 2006