ECE 4750 PSET 3

Tim Yao (ty252)

Nov 6, 2015

Worked with Gautam Ramaswamy, Gaurab Bhattacharya, and Sacheth Hegde.

1 Tree Network Topologies

Part	Topology	B_c	b (bits/cycle)	γ_{max}	Θ_{term} (bits/cycle)
Part 1.A	**Baseline Tree Topology**	2	32	2	16
Part 1.B	**Fat-Tree Topology**	8	32	1	32

Figure 1: Ideal Terminal Throughput for Tree Topologies

Part	Topology	H_D	H_r	t_r (cycles)	H_c	t_c (cycle)	L/b (cycle)	t_0 (cycle)
Part 1.A	**Baseline Tree Topology**	4	3	2	2	1	3	11
Part 1.B	**Fat-Tree Topology**	4	3	3.83	2	1	3	16.5

Figure 2: Zero-Load Latency for Tree Topologies

1.a Performance of Baseline Tree Topology

To calculate the minimum bisection channel count, I first found the minimum bisection cut. This happens to be the cut on the channels that connect the two level-2 routers. This cut cuts 2 channels, so the B_c is 2.
To calculate the max channel, there are 3 possibilities. It can be the channels connecting between the nodes and the level-1 routers, level-1 and level-2 routers, and between the two level-2 routers. Because we are dealing with uniform random routing, all channels of one level has the same load.
The channel load for channels between nodes and level-1 routers are 1. In uniform random routing, the amount traffic sent from and to a node is always equal to 1.
The channel load for channels between level-1 and level-2 routers are 1.5. To calculate this, I first look at the amount of traffic that is sent from one node to all other nodes that require crossing the this channel. For example, all traffic that go from node 0 to 2,3,4,5,6,7 need to cross this channel. Each unit of traffic is 1/8, so for each node that resides under (in the sense of a tree) that channel is 6/8. Because there are 2 nodes (ie. node 0 and 1) under that channel, the max channel load is 2*6/8 = 12/8 = 3/2.
The channel load for channels between level-2 routers are 2. We use the same idea from the previous calculation to calculate this max load. Each node on the left side needs to send to 4 nodes on the right side, therefore, each node contributes 4*1/8 = 1/2 unit of traffic. There are 4 nodes, so the max load is 4*1/2 = 2. From this, we see that the max channel load is between the two level 2 routers. And using the ideal terminal throughput equation, $\Theta_{term}=B_c/\gamma_{max}$, I calculated that Θ_{term}=32/2=16.

The network diameter is 4 router hops. One example is the path going from node 0 to node 7. This minimum path takes 4 router hops.
The average number of router hops is calculated using the following method. The average number of router hops for going from 0 to itself and all other nodes is the same from 1 to itself and all other nodes, from 2 to itself and all other nodes, and etc. Therefore, we can calculate the overall average number of router hops by simply analyzing the traffic going from 1 node to itself and all other nodes. If we pick 0, then the analysis is as following:

0->0: 1
0->1: 1
0->2: 3
0->3: 3
0->4: 4
0->5: 4
0->6: 4
0->7: 4
The average is then: (1+1+3+3+4+4+4+4)/8=24/8=3 hops.
Because all of the routers are radix-2, the average per-hop router latency is 2.
The method to calculate the average channel hop count is the same as calculating the average router hop count.
0->0: 0
0->1: 0
0->2: 2
0->3: 2
0->4: 3
0->5: 3
0->6: 3
0->7: 3
The average is then: (2+2+3+3+3+3)/8=16/8=2 hops.
The average per-hop channel latency is 1 cycle as stated in the assumption.
Because the message is 96 bits and the channel width is 32 bits, the serialization latency is 96/32=3 cycles.
Now, we can calculate the zero-load latency.
t_0=$H_r t_r$+$H_c t_c$+L/b
t_0=3*2+2*1+3=11 cycles

1.b Performance of Fat-Tree Topology

We use the same approach as in the previous section to calculate the max channel load and ideal terminal throughput for the baseline tree topology. Again, the channels connecting nodes to level-1 routers have a load of 1. The total load on all channels between level-1 and level-2 routers are the same (because this is a uniforma random traffic pattern), but now there are twice the number of channels. Therefore, the channel load is 6/8=3/4. The total load on all channels between the two level-2 routers are also the same, but because there are now 4 times the number of channels, the channel load is 1/2. Therefore, the max channel load is on the channels between nodes and level-1 routers. The max channel load is 1 and the ideal terminal throughput is 32/1=32.

Because the layout topology and channel width of the fat-tree is the same as the baseline, the network diameter, average router hop count, average channel hop count, average per-hop channel latency, and serialization latency are the same.
The only difference here is the average per-hop router latency due to different radix routers. To calculate this, we take a similar approach to calculating the average hop counts. We use a weighted average for each node-pair due to the different latencies of different routers.
0->0: 3
0->1: 3
0->2: 3+5+3 = 11
0->3: 3+5+3 = 11
0->4: 3+5+5+3 = 16
0->5: 3+5+5+3 = 16
0->6: 3+5+5+3 = 16
0->7: 3+5+5+3 = 16
The average is: (3+3+11+11+16+16+16+16)/24=92/24=23/6=3.833
From this, we can calculate the zero-load latency.
t_0=3*23/6+2*1+3=16.5 cycles

1.c Integrating Processors, Memories, and Networks

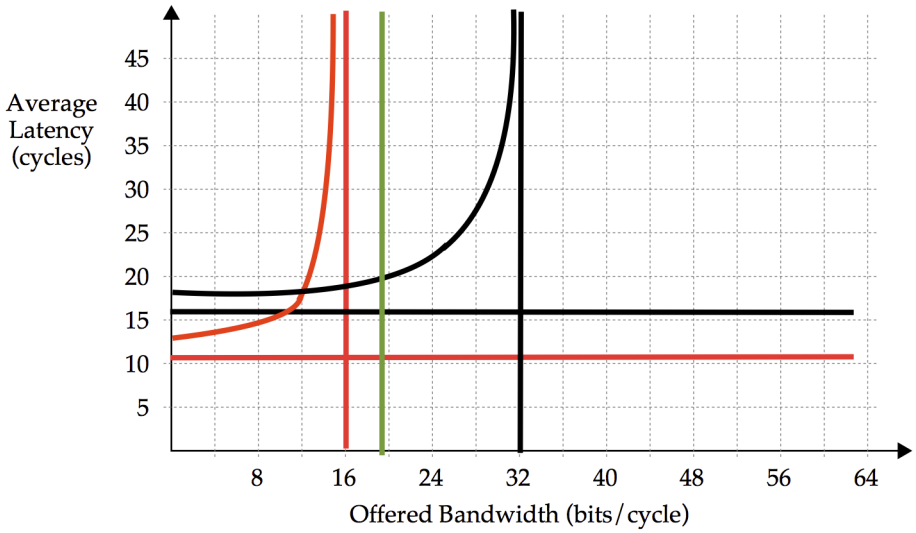

Figure 3: Average Latency versus Offered Bandwidth Curve

The red lines represent the baseline tree, the black lines represent the fat-tree, and the green line represents where the application will be located on the latency vs offered bandwidth curve.

As shown by the graph, the fat-tree will perform better on this application because the expected bandwidth of application is higher than the ideal terminal throughput of the baseline design but within the range of the fat-tree design. While the baseline design has a lower zero-load latency than the fat-tree design, it has a much lower ideal throughput than the fat-tree design. This means that if we used the baseline design, we will have to stall the processors and wait until the network is free enough to handle to the memory requests. This stalling can have a huge negative impact on the performance and completely negate the benefits of the lower zero-load latency of the baseline design. Therefore, the fat-tree will perform much better because we are operating well under its saturation point. This means that we can still achieve low latency while sustaining full expected throughput.

I do not think that this conclusion will apply across all applications, but it will apply to most. The reason is that the fat-tree design has a much higher (50%) latency than the baseline design. The purpose of a cache is to lower the memory access latency, so low latency is very important, especially for the L1 cache. The only time that fat-tree is a better design is when the program contains more than 15% of load and store word instructions. The 10% to 15% is roughly where the latency vs offered bandwidth curves of the 2 designs intersect. Therefore, for applications that does not require frequent memory accesses, the baseline design will perform better due to the lower latency. If our programs contain over 15% of load and store instructions, then the fat-tree is a better design because we will not hit network saturation until over 30% of load and store instructions. This means that we are able to sustain a decently low latency for longer. From the many assembly programs that I have seen throughout the course (VVadd, Array incrementer, etc), the amount of load and store instructions per program (or per iteration) is usually 20% to 40%. Therefore, I believe that if one were to choose only one implementation, the fat-tree will be the better choice as it will avoid the need to stall in more programs. However, as I stated earlier, the fat-tree will not always perform better than the baseline design, especially for programs with few memory accesses.

Worked with Gautam Ramaswamy, Gaurab Bhattacharya, and Sacheth Hegde.

2 Channel and Router Microarchitecture

2.a Throughput with One Element of Buffering per Channel Queue

Cycle:	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
pkt0	I	R0	L0	L1	R1	O														
pkt1		I	R0	R0	L0	L1	R1	O												
pkt2			I	q	R0	R0	L0	L1	R1	O										
pkt3				I	I	q	R0	R0	L0	L1	R1	O								
pkt4						I	I	q	R0	R0	L0	L1	R1	O						
pkt5								I	I	q	R0	R0	L0	L1	R1	O				
pkt6										I	I	q	R0	R0	L0	L1	R1	O		
pkt7												I	I	q	R0	R0	L0	L1	R1	O

Figure 4: Pipeline Diagram for Elastic Buffering with One-Element Channel Buffers

As show by the bolded vertical lines, in steady state, it takes 2 cycles to move a packet. This gives a peak terminal throughput of 1/2=0.5 packets per cycle.
The ideal flow control should provide 1 packet per cycle. This design cannot achieve the throughput of ideal flow control due the stalling required to wait for the channel queues to become empty.

2.b Throughput with Two Elements of Buffering per Channel Queue

Cycle:	1	2	3	4	5	6	7	8	9	10	11	12	13
pkt0	I	R0	L0	L1	R1	O							
pkt1		I	R0	L0	L1	R1	O						
pkt2			I	R0	L0	L1	R1	O					
pkt3				I	R0	L0	L1	R1	O				
pkt4					I	R0	L0	L1	R1	O			
pkt5						I	R0	L0	L1	R1	O		
pkt6							I	R0	L0	L1	R1	O	
pkt7								I	R0	L0	L1	R1	O

Figure 5: Pipeline Diagram for Elastic Buffering with Two-Element Channel Buffers

As show by the bolded vertical lines, in steady state, it takes 1 cycle to move a packet. This gives a peak terminal throughput of 1 packet per cycle.
The ideal flow control should provide 1 packet per cycle. This design is able to achieve the throughput of ideal flow control.

2.c Pipeline Diagram for Round-Robin Arbitration

Cycle:	1	2	3	4	5	6	7	8	9	10	11	12	13
pkt0-2	I	R0	L0	L1	R1	O							
pkt0-3	I	R0	R0	L0	L1	R1	O						
pkt1-2		I	R0	R0	L0	L1	R1	O					
pkt1-3		I	q	R0	R0	L0	L1	R1	O				
pkt2-2			I	q	R0	R0	L0	L1	R1	O			
pkt2-3			I	q	q	R0	R0	L0	L1	R1	O		
pkt3-2				I	q	q	R0	R0	L0	L1	R1	O	
pkt3-3				I	q	q	q	R0	R0	L0	L1	R1	O

Figure 6: Pipeline Diagram for Round-Robin Arbitration

There is no steady state. As shown by the pipeline diagram, the number of elements in the $R_{0,1}$ queue continues to increase. This is due to the the $C_{0,3}$ channel not able to sustain the required throughput (2 phits per cycle). This means that packets will need to be stalled in R0 during arbitration and therefore cause future packets to be continuously queued up.

2.d Global Fairness for Round-Robin Arbitration

No, the network does not have global strong fairness.
Assume that initially, the priority goes to the top input port. In our traffic pattern, inputs 0, 1, and 2 will try to send their packets to output 1, and input 3 will try to send its packets to output 3. First, we see that both inputs 0 and 1 try to access channel $C_{0,0}$, so round-robin arbitration will occur. This means on channel $C_{0,0}$, inputs 0 and 1 will each receive 50% of the bandwidth. In $R_{0,1}$, because input 2 will access the top output to channel $C_{0,2}$ and input 3 will access the bottom output to channel $C_{0,3}$. Therefore, no arbitration is needed and each channel can sustain full bandwidth from a single input. In $R_{1,1}$, the bottom input coming from channel $C_{0,3}$ will always go to output 3, so again, no arbitration is needed. This means that the requests from input 3 will always be served. In $R_{1,0}$, both inputs want to access output 1, so arbitration is needed. This means that 50% of the bandwidth comes from the top input (coming from channel $C_{0,0}$) and the other 50 comes from the bottom input (from channel $C_{0,2}$). This means that on output 1, 50% of the bandwidth comes from input 2, 25% from input 1, and 25% from input 0. From this we can see that requests from all inputs are NOT served equally often.

3 In-Order Superscalar Processors

3.a Pipeline Diagram for Single-Issue PARCv1 Processor

Cycle:	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
lw r1 , 0(r2)	F	D	X	M	W													
lw r3 , 0(r4)		F	D	X	M	W												
mul r1, r1, r6			F	D	X	M	W											
mul r3, r3, r7				F	D	X	M	W										
addu r8, r1, r3					F	D	X	M	W									
addu r9, r9, r8						F	D	X	M	W								
addiu r2, r2, 4							F	D	X	M	W							
addiu r4, r4, 4								F	D	X	M	W						
addiu r10, r10, -1									F	D	X	M	W					
bne r10, r0, loop										F	D	X	M	W				
opA											F	D	-	-	-			
opB												F	-	-	-	-		
lw r1 , 0(r2)													F	D	X	M	W	
lw r3 , 0(r4)														F	D	X	M	W

Figure 7: Pipeline Diagram for Single-Issue PARCv1 Processor

As shown by the bold vertical lines, each loop takes 12 cycles to execute. The CPI is therefore 12/10 = 1.2. The IPC is 1/CPI = 0.833.
CPI = 1.2 cycles/instruction
IPC = 0.83 instructions/cycle

3.b Pipeline Diagram for Dual-Issue PARCv1 Processor

Cycle:	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
lw r1 , 0(r2)	F	D	B0	B1	W												
lw r3 , 0(r4)	F	D	D	B0	B1	W											
mul r1, r1, r6		F	F	D	A0	A1	W										
mul r3, r3, r7		F	F	D	D	A0	A1	W									
addu r8, r1, r3				F	F	D	A0	A1	W								
addu r9, r9, r8				F	F	D	D	B0	B1	W							
addiu r2, r2, 4						F	F	D	A0	A1	W						
addiu r4, r4, 4						F	F	D	B0	B1	W						
addiu r10, r10, -1								F	D	B0	B1	W					
bne r10, r0, loop								F	D	D	A0	A1	W				
opA									F	F	D	-	-	-			
opB									F	F	D	-	-	-			
opC											F	-	-	-	-		
opD											F	-	-	-	-		
lw r1 , 0(r2)												F	D	B0	B1	W	
lw r3 , 0(r4)												F	D	D	B0	B1	W

Figure 8: Pipeline Diagram for Dual-Issue PARCv1 Processor

As shown by the bold vertical lines, each loop takes 11 cycles to execute. The CPI is therefore 11/10 = 1.1. The IPC is 1/CPI = 0.910.
CPI = 1.1 cycles/instruction
IPC = 0.91 instructions/cycle

3.c Optimized Pipeline Diagram for Dual-Issue PARCv1 Processor

```
1  lw r1 , 0(r2)
2  addiu r2, r2, 4
3  lw r3 , 0(r4)
4  addiu r4, r4, 4
5  mul r1, r1, r6
6  addiu r10, r10, -1
7  mul r3, r3, r7
8  addu r8, r1, r3
9  addu r9, r9, r8
10 bne r10, r0, loop
```

Cycle:	1	2	3	4	5	6	7	8	9	10	11	12	13
lw r1 , 0(r2)	F	D	B0	B1	W								
addiu r2, r2, 4	F	D	A0	A1	W								
lw r3 , 0(r4)		F	D	B0	B1	W							
addiu r4, r4, 4		F	D	A0	A1	W							
mul r1, r1, r6			F	D	A0	A1	W						
addiu r10, r10, -1			F	D	B0	B1	W						
mul r3, r3, r7				F	D	A0	A1	W					
addu r8, r1, r3				F	D	D	B0	B1	W				
addu r9, r9, r8					F	F	D	B0	B1	W			
bne r10, r0, loop					F	F	D	A0	A1	W			
opA							F	D	-	-	-		
opB							F	D	-	-	-		
opC								F	-	-	-	-	
opD								F	-	-	-	-	
lw r1 , 0(r2)									F	D	B0	B1	W
addiu r2, r2, 4									F	D	A0	A1	W

Figure 9: Optimized Pipeline Diagram for Dual-Issue PARCv1 Processor

As shown by the bold vertical lines, each loop takes 8 cycles to execute. The CPI is therefore $8/10 = 0.8$. The IPC is $1/\text{CPI} = 1.25$.
CPI = 0.8 cycles/instruction
IPC = 1.25 instructions/cycle

3.d Optimized Pipeline Diagram for Quad-Issue PARCv1 Processor

```
1  lw r1 , 0(r2)
2  addiu r2, r2, 4
3  lw r3 , 0(r4)
4  addiu r4, r4, 4
5  mul r1, r1, r6
6  addiu r10, r10, -1
7  mul r3, r3, r7
8  addu r8, r1, r3
9  addu r9, r9, r8
10 bne r10, r0, loop
```

Cycle:	1	2	3	4	5	6	7	8	9	10	11	12
lw r1 , 0(r2)	F	D	B0	B1	W							
addiu r2, r2, 4	F	D	A0	A1	W							
lw r3 , 0(r4)	F	D	H0	H1	W							
addiu r4, r4, 4	F	D	G0	G1	W							
mul r1, r1, r6		F	F	D	A0	A1	W					
addiu r10, r10, -1		F	F	D	B0	B1	W					
mul r3, r3, r7		F	F	D	G0	G1	W					
addu r8, r1, r3		F	F	D	D	H0	H1	W				
addu r9, r9, r8				F	F	D	B0	B1	W			
bne r10, r0, loop				F	F	D	A0	A1	W			
opA				F	F	D	G0	-	-			
opB				F	F	D	H0	-	-			
opC						F	D	-	-	-		
opD						F	D	-	-	-		
opE						F	D	-	-	-		
opF						F	D	-	-	-		
opG							F	-	-	-	-	
opH							F	-	-	-	-	
opI							F	-	-	-	-	
opJ							F	-	-	-	-	
lw r1 , 0(r2)								F	D	B0	B1	W
addiu r2, r2, 4								F	D	A0	A1	W
lw r3 , 0(r4)								F	D	H0	H1	W
addiu r4, r4, 4								F	D	G0	G1	W

Figure 10: Optimized Pipeline Diagram for Quad-Issue PARCv1 Processor

As shown by the bold vertical lines, each loop takes 7 cycles to execute. The CPI is therefore $7/10 = 0.7$. The IPC is $1/\text{CPI} = 1.43$.
CPI = 0.7 cycles/instruction
IPC = 1.43 instructions/cycle

3.e Instruction Level Parallelism

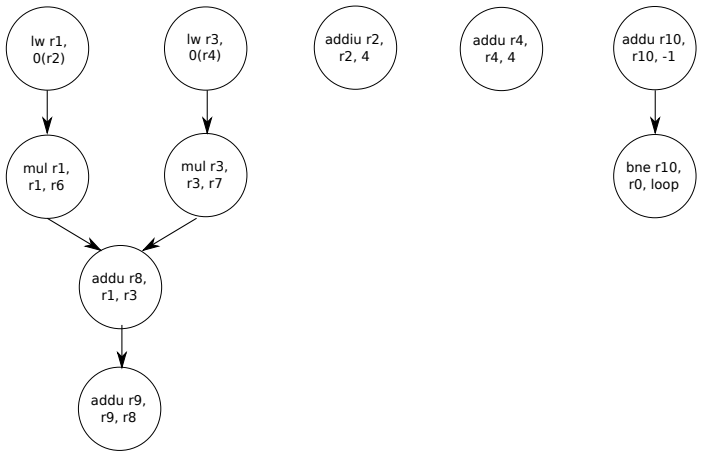

Figure 11: Instruction Dependency Graph for Single Iteration

The longest path contains 4 nodes. The ideal ILP for a single iteration is $10/4 = 2.5$.

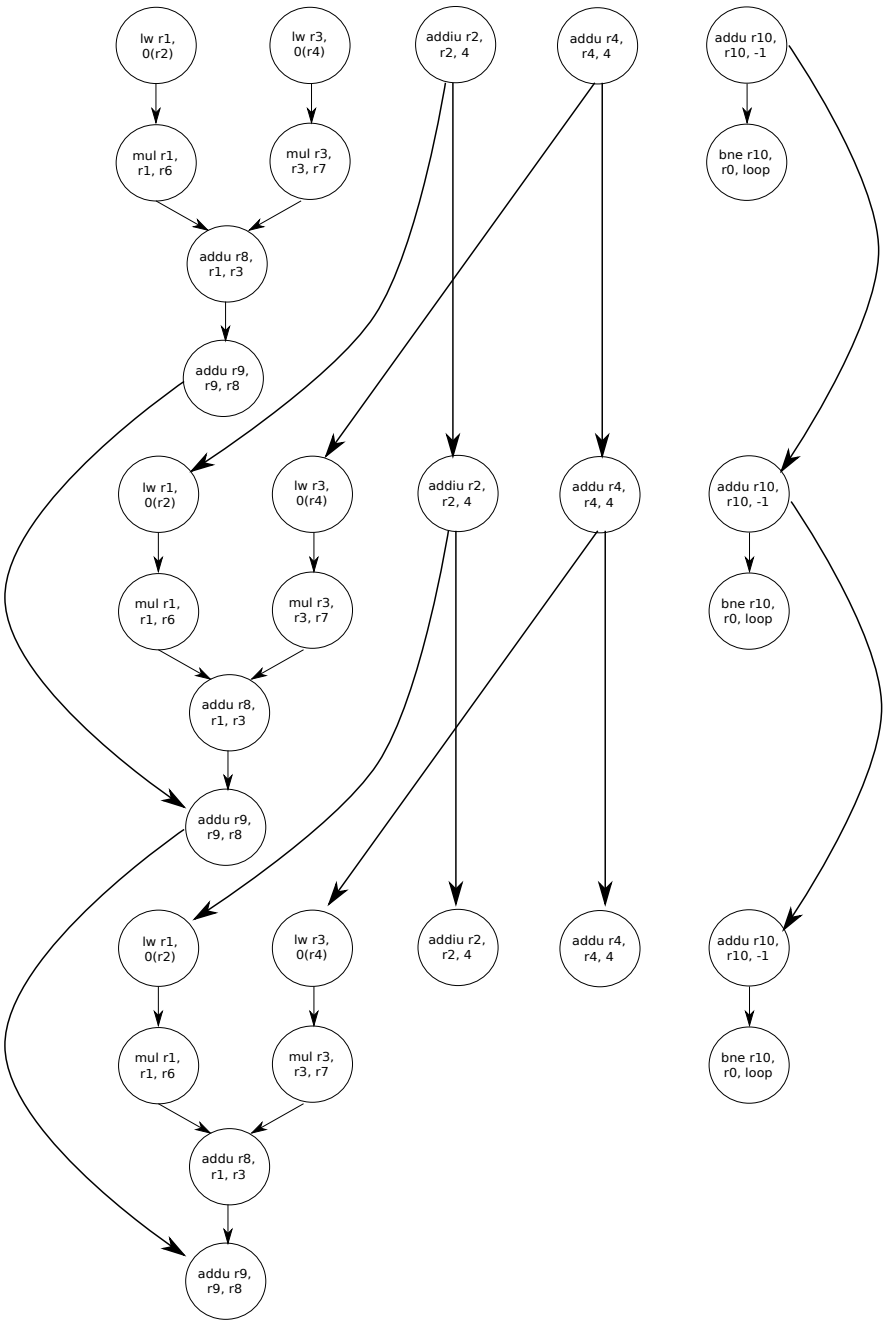

Figure 12: Instruction Dependency Graph for Three Iterations

The longest path contains 6 nodes. The ideal ILP for three iterations is $30/6 = 5$.

The ideal ILP for N iterations of the loop is simply $10N/(3+N)$.

The IPC of the quad-issue processor is less than the ideal ILP due to several different reasons. The first is that the quad issue processor can only execute at most 4 instructions simultaneously. This thereby limits the IPC to a max of 4. Then, the load word instructions are resolved in the second functional unit in the pipeline, which means that a RAW hazard on the next set of instructions will need to be stalled by 1 cycle. There is also a RAW hazard within a fetch block, so this requires 1 cycle of stalling. Finally, the branch instruction introduces another 2 cycles of delay. There are also 2 squashed instructions in the fetch block with the branch instruction, so this reduced the number of executed instructions for calculating the IPC. In the ideal case, we can execute 12 instructions in 3 cycles, but after adding in the squashed instructions and various delays, we are actually executing 10 instructions in 7 cycles.