Assignment 3

1.Problem Solution : Branch Predictors

Benchmarks/Branch Predictors	Nottaken	Bimod	2lev	Comb	Perfect
Fibonacci	0.4469	0.6097	0.6533	0.6509	0.7007
Matmul	0.6222	0.7239	0.7239	0.7239	0.7258
Pi	0.4978	0.5507	0.5505	0.5507	0.5552
Whetstone	0.4456	0.5873	0.5892	0.5927	0.6074
Memcopy	0.4529	0.5085	0.5085	0.5085	0.5086

Table1. IPC Values for Different Branch Predictors for different Benchmarks

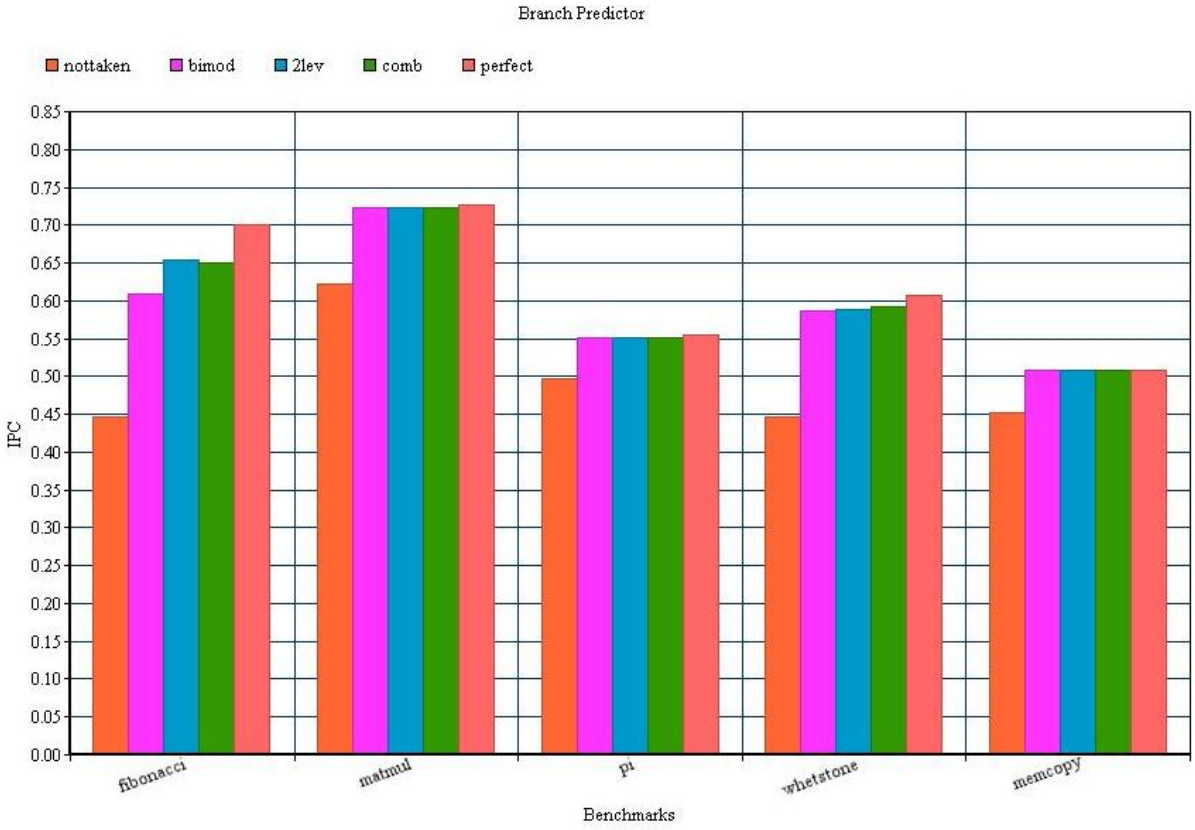

Fig1. Variation of IPC with Different Branch Predictors for different Benchmarks

2. Problem Solution : Inorder execution

Benchmarks	Inorder				
	1	2	4	8	defaut
Fibonacci	0.6528	0.7137	0.7466	0.7468	0.7468
Matmul	0.7239	0.7365	0.7368	0.7369	0.7369
Pi	0.5505	0.5699	0.5747	0.5754	0.5754
Whetstone	0.5892	0.7163	0.7353	0.7367	0.7127
Memcopy	0.5085	0.5348	0.5637	0.5637	0.5637

Table2. IPC with Inorder Execution and Different Processor width for different Benchmarks

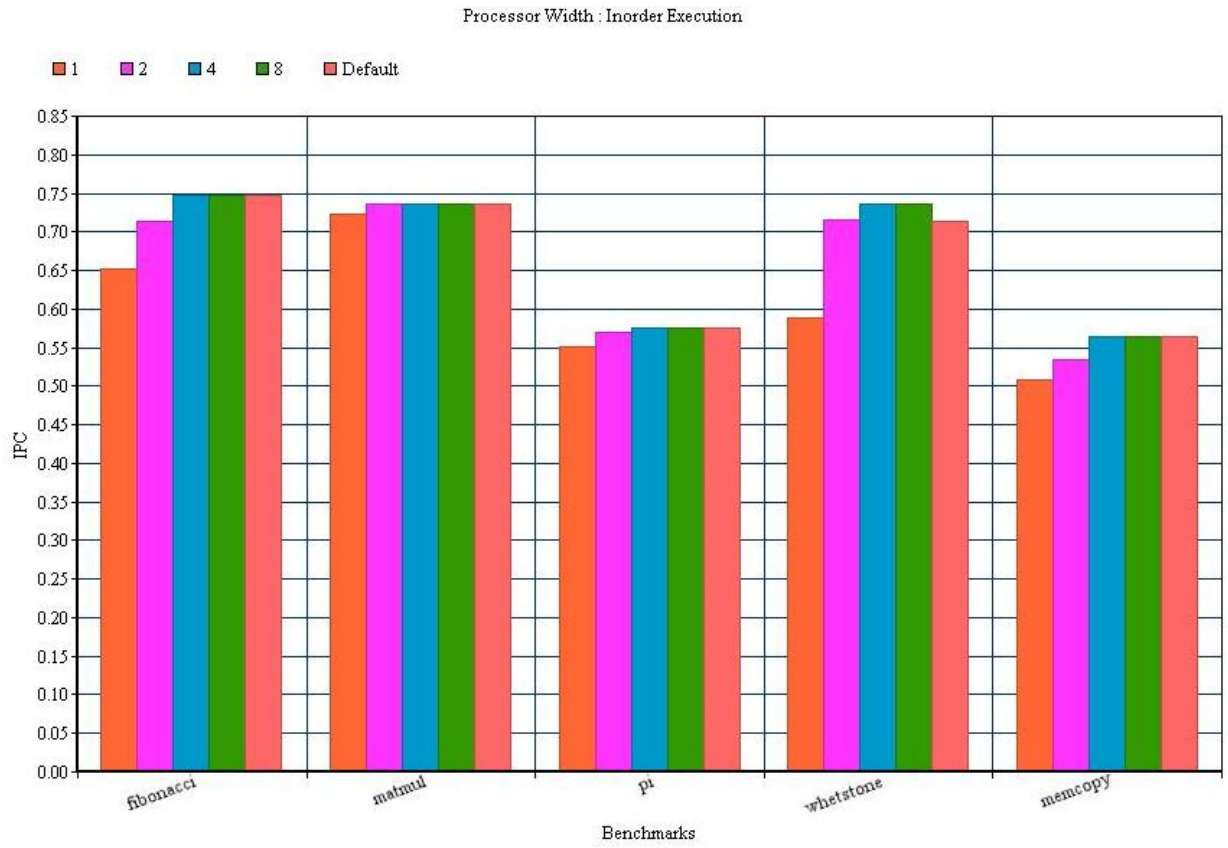

Fig2. Variation of IPC with Inorder Execution and Different Processor width for different Benchmarks

2. Problem Solution : Outorder Execution

Benchmarks	Outorder				
	1	2	4	8	defaut
Fibonacci	0.6772	1.2586	1.8944	2.0008	2.0094
Matmul	0.7474	1.485	2.019	2.1103	2.1103
Pi	0.6575	1.0257	1.1074	1.125	1.0502
Whetstone	0.6256	1.1285	1.3886	1.4225	1.1885
Memcopy	0.5216	0.8904	1.2035	1.1384	1.204

Table3. IPC with Out of order Execution and Different Processor width for different Benchmarks

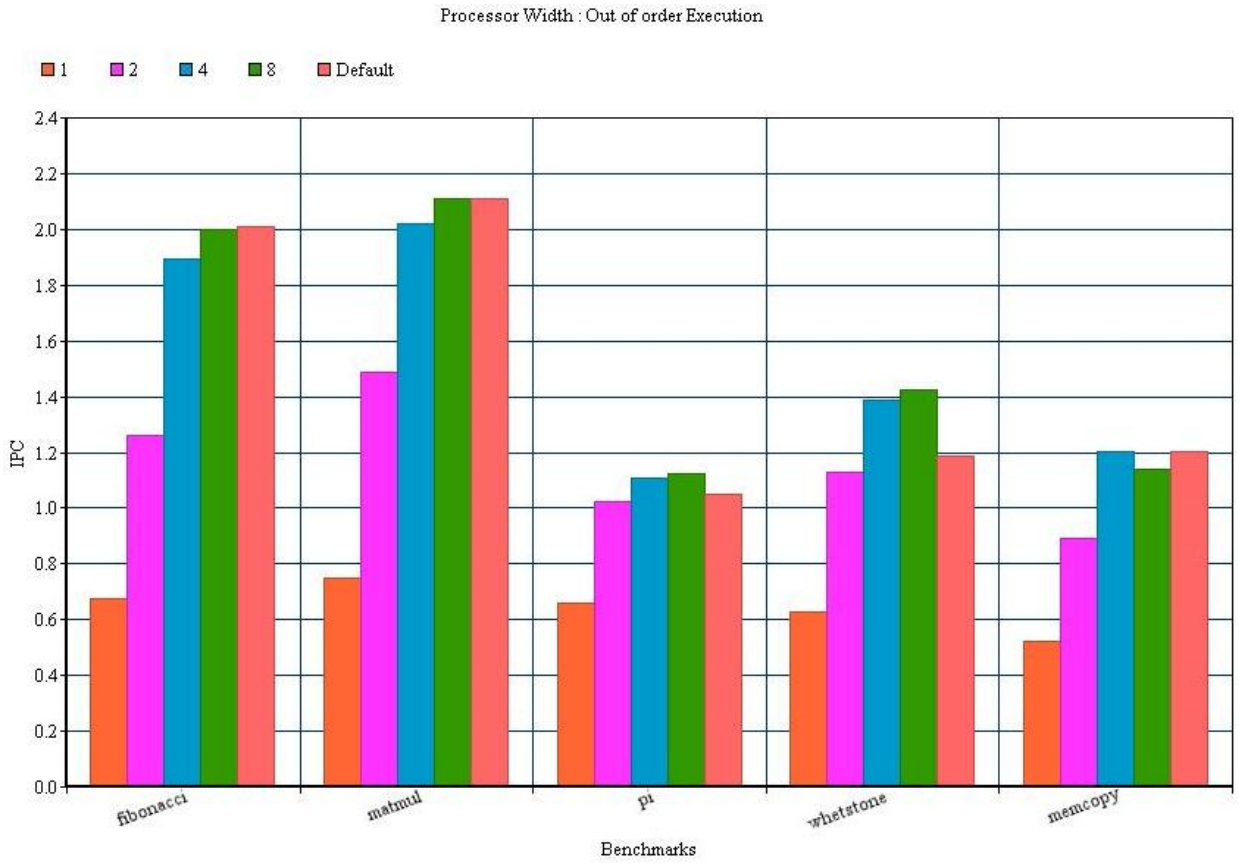

Fig3. Variation of IPC with Out of order Execution and Different Processor width for different Benchmarks

3. Problem Solution: Processor resources are Default

Benchmarks	Default				
	(16,8)	(32,8)	(32,16)	(64,16)	(64,32)
fibonacci	2.0094	2.0605	1.9805	2.0478	2.0758
matmul	2.1103	2.2288	2.7622	3.3346	3.7517
pi	1.0502	1.0657	1.3065	1.411	1.5414
whetstone	1.1885	1.1043	1.1466	0.9853	1.0281
memcopy	1.204	1.2173	1.5901	1.5995	1.8857

Table4. IPC by Increasing (ruu,lsq) for different benchmarks for default Processor Resources

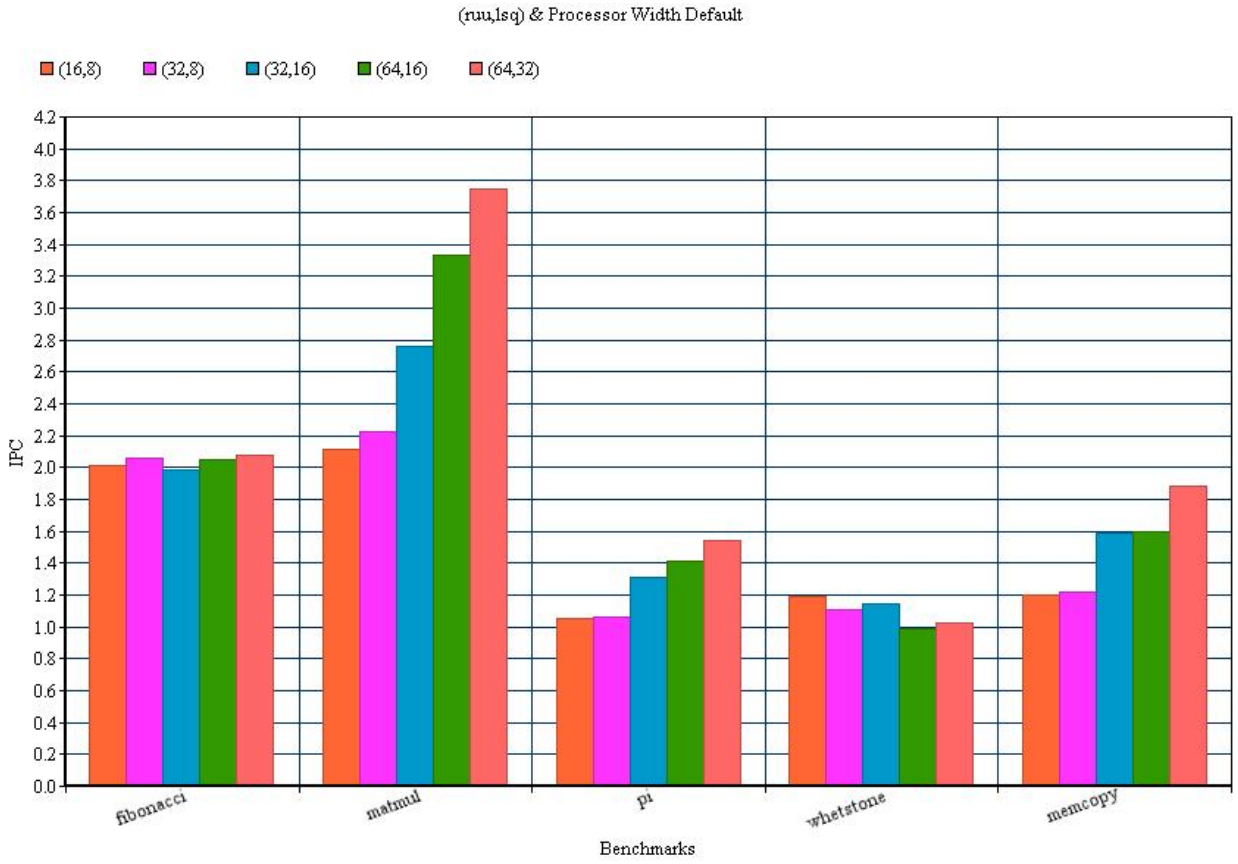

Fig4. Variation of IPC by Increasing (ruu,lsq) for different benchmarks for default Processor Resources

3. Problem Solution: Processor Resources Doubled

Benchmarks	Doubled				
	(16,8)	(32,8)	(32,16)	(64,16)	(64,32)
fibonacci	2.0008	2.0393	2.1915	2.1669	2.1884
matmul	2.1103	2.2288	2.7654	3.2355	3.9412
pi	1.125	1.1424	1.4433	1.519	1.6964
whetstone	1.4225	1.4979	1.5771	1.6181	1.7713
memcopy	1.1384	1.1503	1.5901	1.5996	1.8859

Table5.IPC by Increasing (ruu,lsq) for different benchmarks for **Doubled** Processor Resources

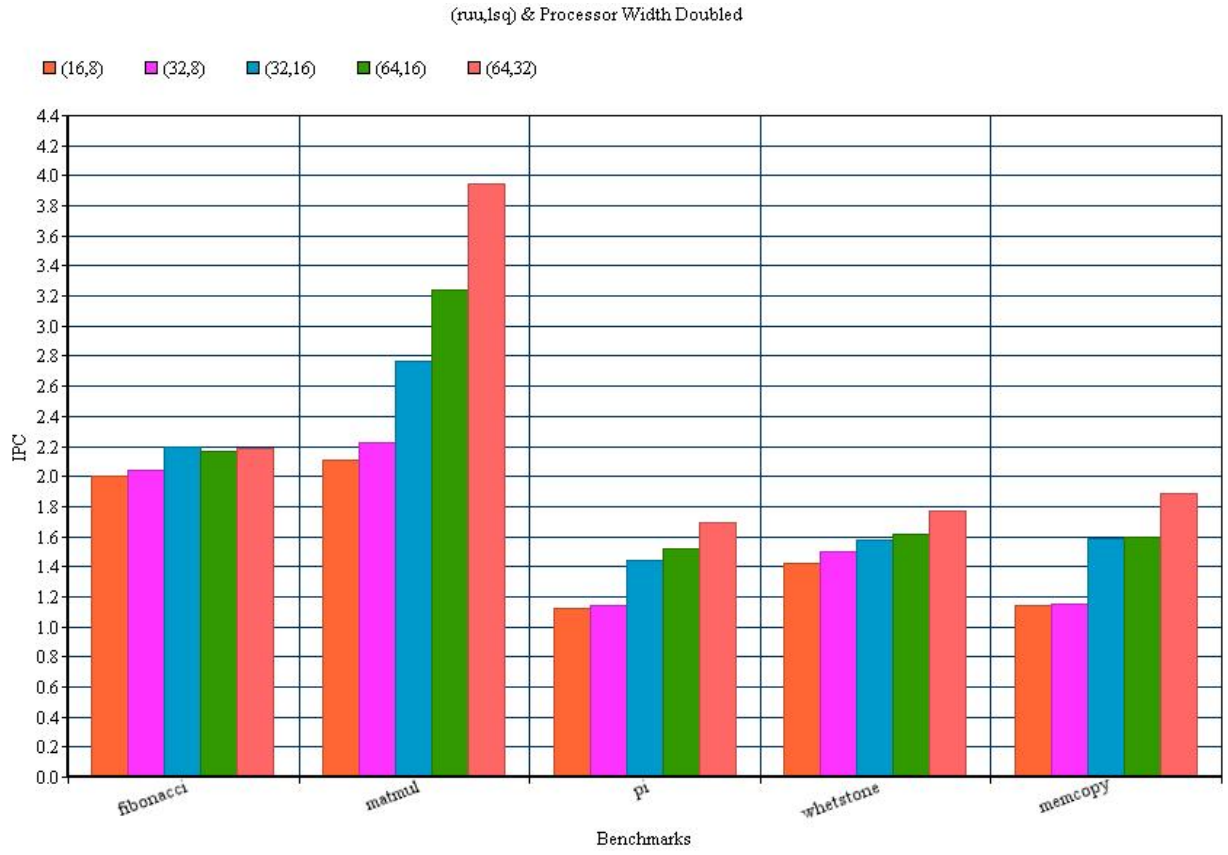

Fig5. Variation of IPC by Increasing (ruu,lsq) for different benchmarks for **Doubled** Processor Resources

4. Problem: Report for problem 1 to 3

In “Not Taken” branch predictor, the first condition of “for loop” will be always wrongly predicted. Hence the Benchmarks with the highest number of conditional loops i.e Whetstone have least IPC value and the Matmul with least conditional loops has the highest IPC. The IPC is better in Bimod and 2lev compared to “Not taken” because of better hysteresis. As can be observed in Table1 and Fig1 above, “comb” branch predictor performs better in all cases except Fibonacci as it contains only one conditional branch executed (n-1) times, so prediction local is always selected. “Perfect” branch predictor gives the best IPC values as it makes use of both local and global branch information but it also depends on the processor resources. Combinational branch predictor combines bimodal and 2-level predictors and hence has equivalent accuracy in prediction.

As the processor resources (fetch,decode,isuue,commit width) increases, IPC increases linearly as it resolves the structural stalls and hence more throughput. Fig2,3 and Table2,3 implies this. As the size of (ruu,lsq) are constant, the IPC Is constant even when there is an increase in processor resources from 4 to 8. And the out of order execution has better IPC compared to Inorder execution since it can re order the instruction and execute them independently. Even for the default processor width of 8,there are no instructions which can make use of the increased resources.

When the (ruu,lsq) are increased, the IPC values are not saturated anymore as there are more number of independent instructions which can be issued by resolving the data dependency. Fig4,5 and Table4,5 shows this. Also because of increase in Load store Queue IPC increases since more load store instructions can be issued. When the resources are doubled, the increase in IPC for matmul benchmark is very significant as it contains more memory access , floating point and integer computations which uses instruction queue for parallel execution.

5.Problem : Average Memory Access Time :

We can calculate the AMAT as below (Considering only the L2 Cache):

Average Memory Access Time = (Initial Chunk latency + (((Cache line size/memory bus width) - 1)*subsequent memory chunk)*uL2.miss_rate+default_latency for L2 Cache).

The default values are as below :

Mem_latency = 18cycles; L2 cache latency = 6cycles; Number of sets in L2 cache = 1024; Block size = 512; Associativity = 4;

So Default AMAT = 11.472 Cycles with IPC = 1.5894;

Now, the memory latency is increased from 18cycles to 200cycles. Then the

AMAT = 42.594 cycles with IPC of 0.5430.

To increase the IPC and also to minimize the AMAT value the following cache configurations are verified and corresponding AMAT values are tabulated in Table6.

Memory Latency	Number of Sets	Block Size	L2 Cache Latency	Associativity	IPC	Miss rate L2	AMAT
18	1024	64	6	4	1.5894	0.171	11.472
200	1024	64	6	4	0.543	0.171	42.594
200	1024	128	6	4	0.8056	0.0872	26.056
200	1024	128	7	8	1.0224	0.0578	20.294
200	1024	256	7	8	1.2467	0.0276	14.2312
200	2048	256	8	8	1.2481	0.0275	15.205
200	4096	256	9	8	1.2902	0.0116	12.0392
200	4096	128	9	8	0.9704	0.0553	21.719
200	4096	256	9	8	1.2902	0.0116	12.0392
200	8192	256	10	8	1.216	0.0116	13.0392
200	4096	512	9	8	1.4061	0.0058	10.8908
200	2048	512	8	8	1.5067	0.0058	9.8908
200	1024	512	7	8	1.5102	0.0136	11.4336
200	4096	1024	9	8	1.4717	0.0029	10.3166

Table5. IPC and AMAT values for different L2 cache configuration

The configuration of <nsets>:<block_size>:<Associativity> : <1024>:<512>:<8> gives the miss rate(0.0136) which is less than the default(0.171) and also IPC of 1.5102 and AMAT of 11.4336. Thus the memory latency is mitigated by increasing the block size and associativity of L2 cache.

The increase in block size will reduce the compulsory misses which helps in drastic reduction of miss rate. As approached in theoretical approaches, the 8way set associativity is almost equal to the fully set associativity. This reduces the conflict misses i.e more the associativity so less is the conflict when there are free cache blocks. This in turn decreases the missrate.