CS/ECE 347, Spring 1998
Lab Assignment 1:
Trace-Driven Cache Simulation

SOLUTIONS

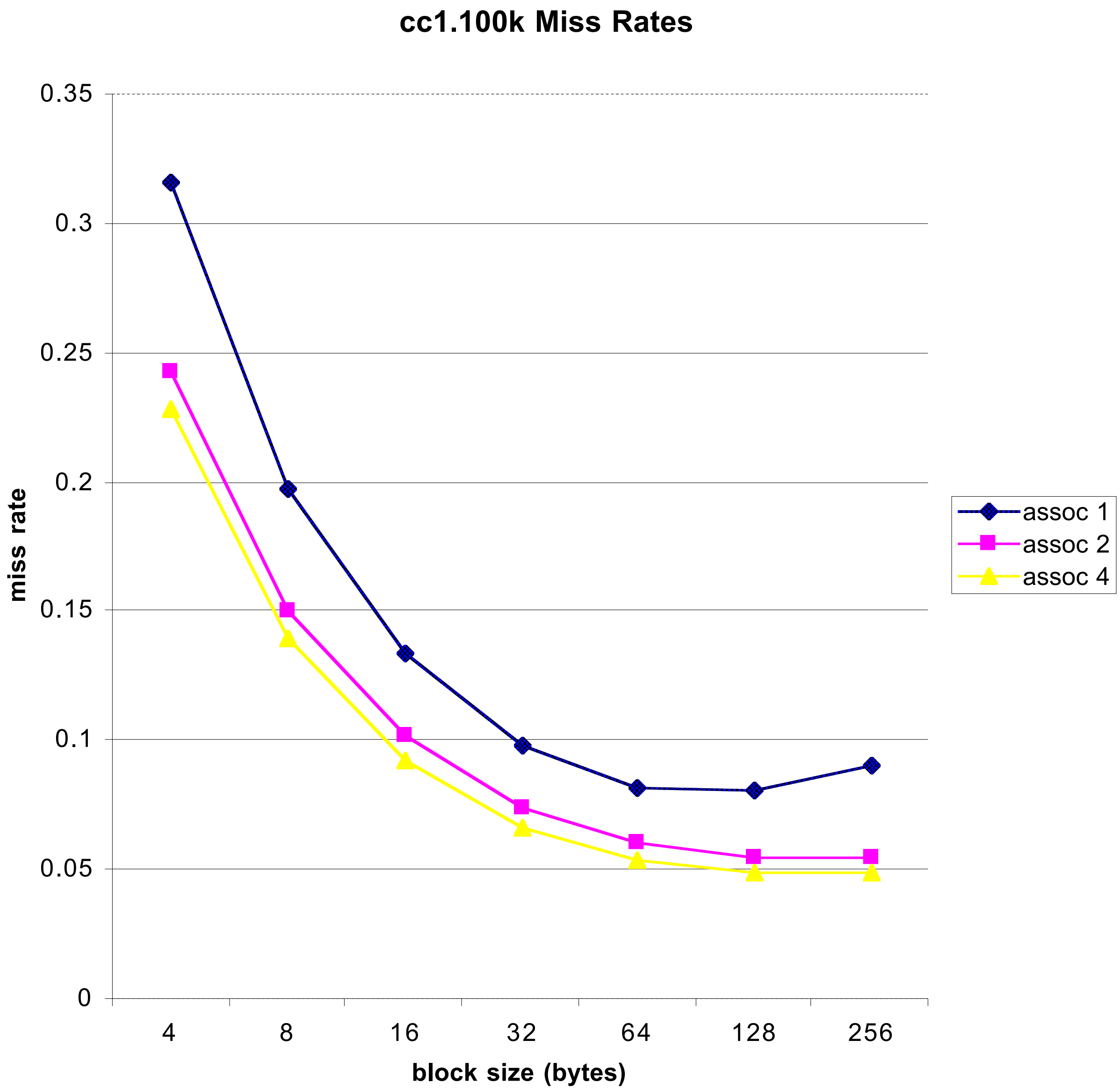
Part I: Exploring Associativity and Block Size

1.1) See the following pages for tables and graphs.

1.2) Small blocks mean shorter sequences of consecutive locations are situated in a block, so that the compulsory misses (associated with the access to the first byte in a block) dominate and result in a high miss rate. Increased block sizes reduce that effect and make increased use of spatial locality, which leads to a decrease in the miss rate. However, increased block sizes reduce the number of blocks in the cache, which means increased conflict misses, so that the miss rate increases. Hence, the miss rate is nonmonotonic with the block size.

1.3) Benchmark `tex` has the lowest miss rates for all block sizes and all associativity values. This can be explained with the fact that a wordprocessing program is manipulating strings of characters, so that it is repeatedly accessing sequential memory locations, therefore making good use of both temporal and spatial locality.
Benchmark `cc1` has the highest miss rate. As a compiler, it scans through the instructions, that are being compiled, in the order of their execution and following branches and jumps to their destinations. Because of the high frequency of jumps and branches (these occur approximately every 5th instruction), `cc1` is unable to make good use of spatial and temporal locality.
Finally, `spice` has a miss rate between the above two benchmarks. It processes data in matrices, which implies good use of spatial locality, i.e., there will be a miss only on the access to the first byte in a block.

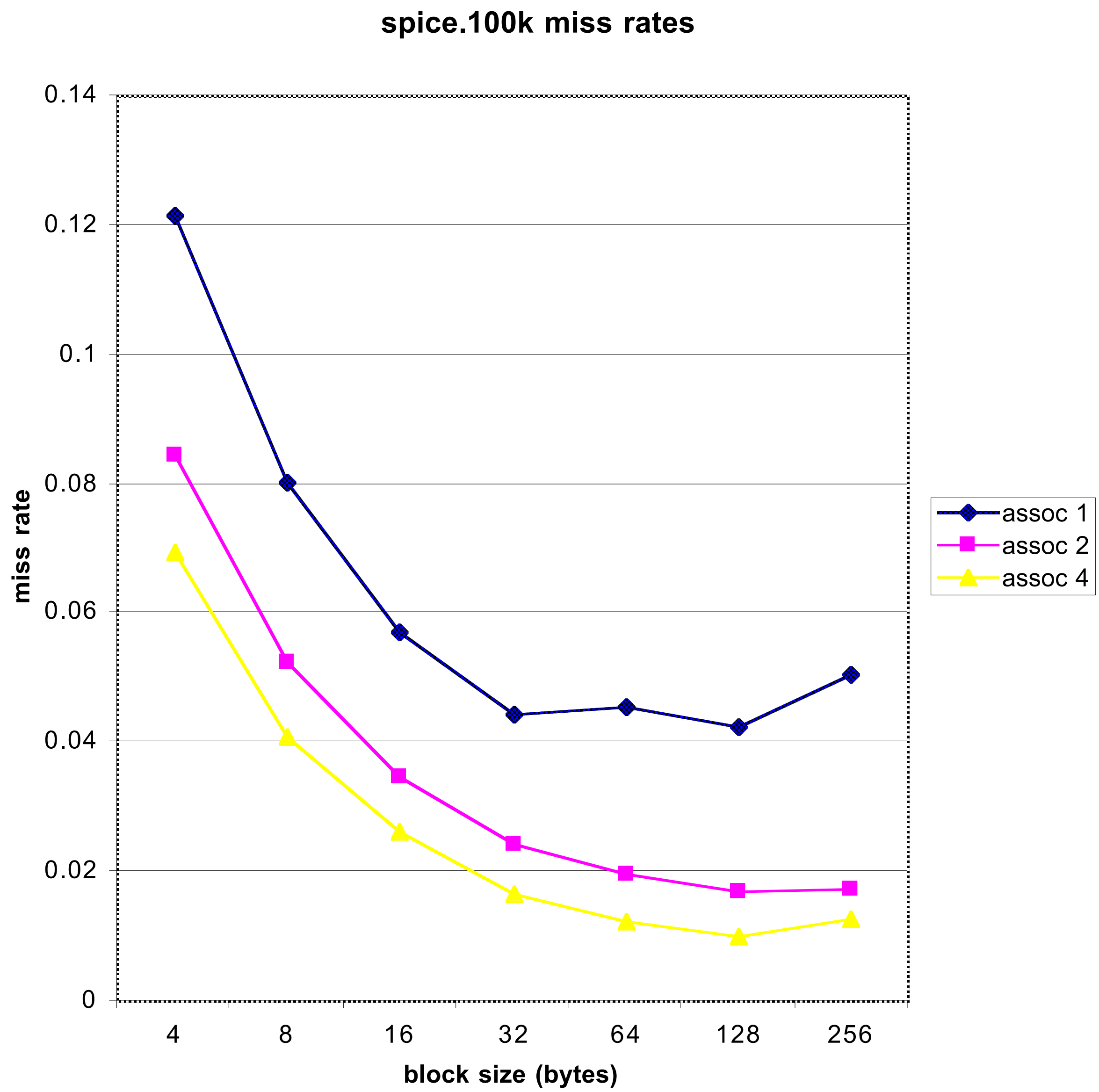
1.4) Increasing the associativity reduces the conflict misses, as two addresses that will map to the same location in a direct-mapped cache will now be accommodated simultaneously within the same set in a set associative cache. This results in a decrease in the miss rate.



cc1.100k

	block size (bytes)						
	4	8	16	32	64	128	256
assoc 1	0.3167	0.1979	0.134	0.0985	0.0815	0.0806	0.0906
assoc 2	0.2434	0.1505	0.1023	0.0744	0.0604	0.0546	0.0549
assoc 4	0.2281	0.1394	0.093	0.0669	0.0536	0.0494	0.0491

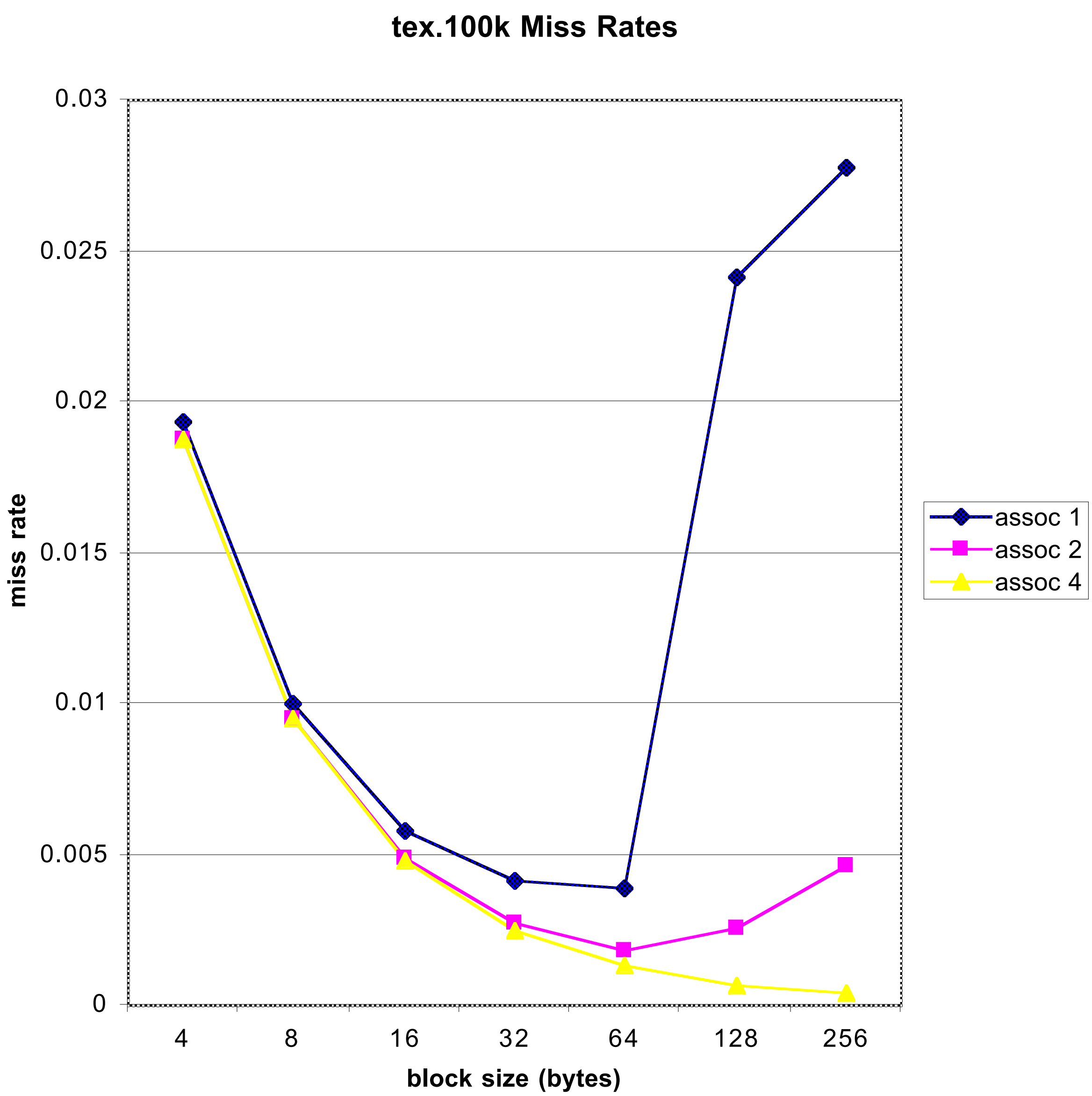
Block size of 256 bytes and associativity of 4 minimize the aggregate miss rate for `cc1`.



spice.100k

	block size (bytes)						
	4	8	16	32	64	128	256
assoc 1	0.1215	0.0803	0.0572	0.0444	0.0454	0.0426	0.0506
assoc 2	0.0843	0.0525	0.0347	0.0243	0.0197	0.017	0.0173
assoc 4	0.0694	0.041	0.0263	0.0164	0.0124	0.0099	0.0127

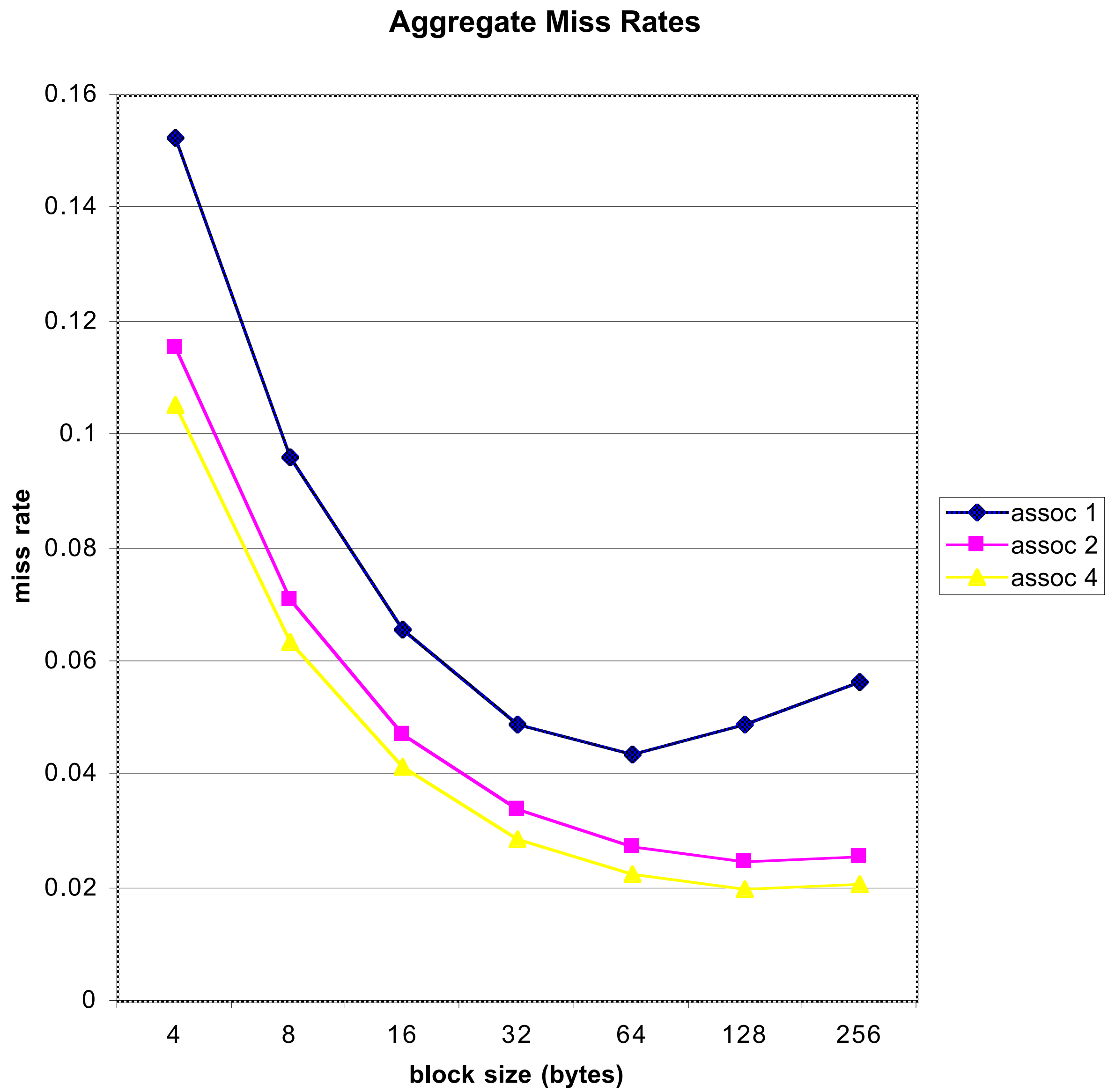
Block size of 128 bytes and associativity of 4 minimize the aggregate miss rate for `spice`.



tex.100k

	block size (bytes)						
	4	8	16	32	64	128	256
assoc 1	0.0193	0.01	0.0058	0.0041	0.0039	0.0241	0.0278
assoc 2	0.0188	0.0095	0.0049	0.0027	0.0018	0.0026	0.0046
assoc 4	0.0188	0.0095	0.0048	0.0025	0.0013	0.0007	0.0004

Block size of 256 bytes and associativity of 4 minimize the aggregate miss rate for `tex`.



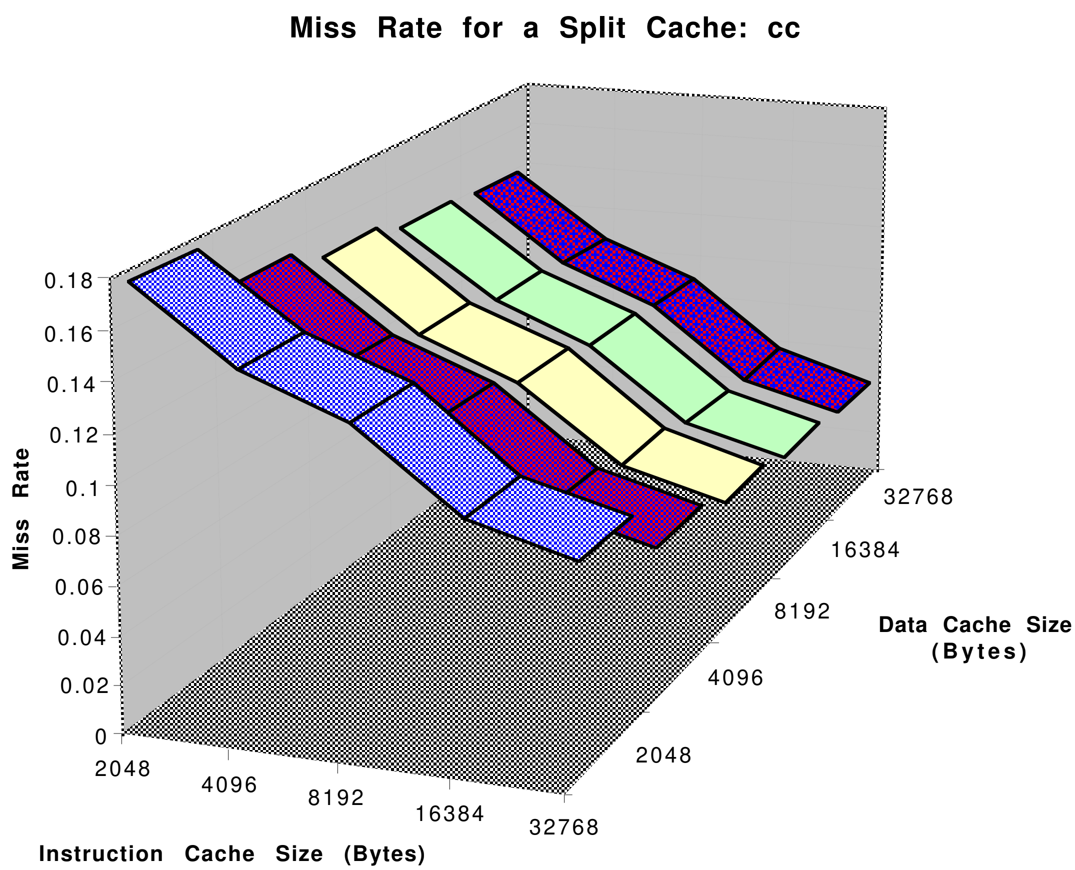
Aggregate miss rates

	block size (bytes)						
	4	8	16	32	64	128	256
assoc 1	0.1525	0.096067	0.065667	0.049	0.0436	0.0491	0.056333
assoc 2	0.1155	0.070833	0.0473	0.0338	0.0273	0.024733	0.0256
assoc 4	0.105433	0.0633	0.041367	0.0286	0.022433	0.02	0.020733

Block size of 128 bytes and associativity of 4 minimize the aggregate miss rate for the three benchmarks.

Part II: Exploring Cache Size and Splitting

2.1)

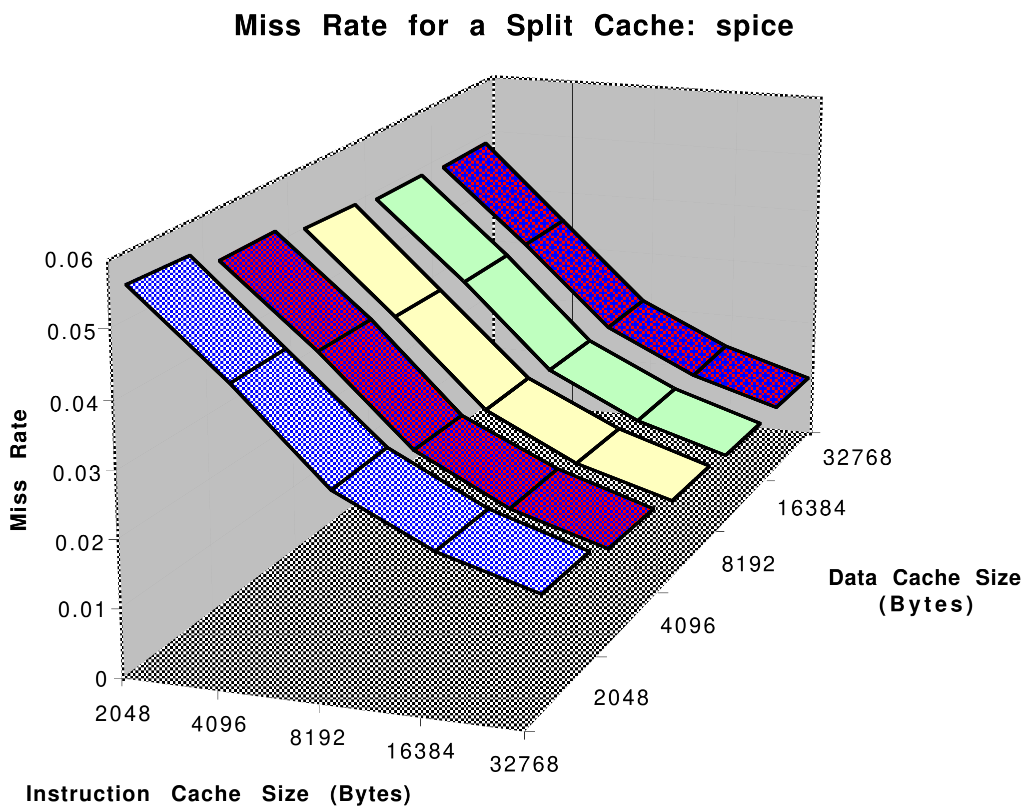


cc1.100k Miss Rates

Data Cache Size (K)

Instruction Cache Size (K)		2	4	8	16	32
	2	0.176	0.1543	0.147	0.1417	0.1392
	4	0.146	0.1243	0.117	0.1117	0.1092
	8	0.1298	0.1081	0.1008	0.0955	0.093
	16	0.0974	0.0757	0.0684	0.0631	0.0606
	32	0.0859	0.0642	0.0569	0.0516	0.0491

Unified Cache	
4K	0.16
8K	0.134
16K	0.0893
32K	0.0637
64K	0.0499

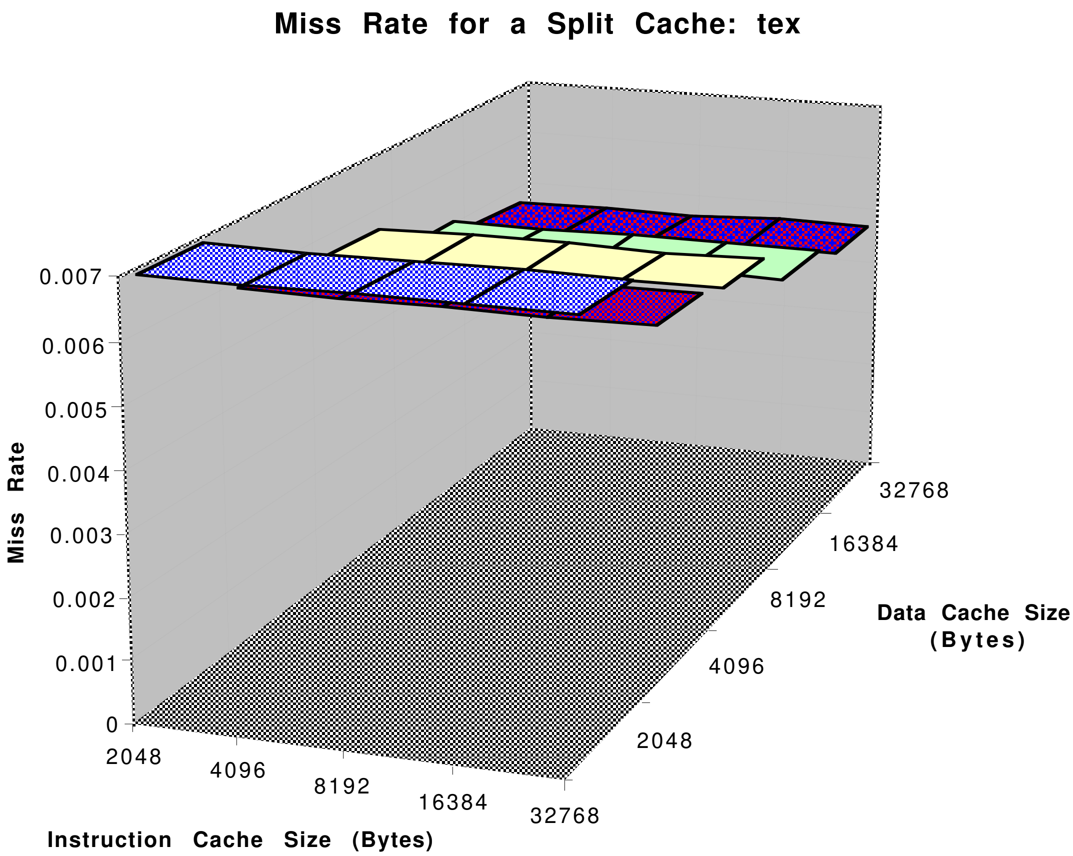


spice.100k Miss Rates

Data Cache Size (K)

Instruction Cache Size (K)		2	4	8	16	32
	2	0.0551	0.0521	0.0503	0.049	0.0488
	4	0.0426	0.0396	0.0377	0.0365	0.0362
	8	0.0293	0.0263	0.0244	0.0232	0.0229
	16	0.0223	0.0192	0.0174	0.0161	0.0159
	32	0.0179	0.0149	0.0131	0.0118	0.0116

Unified Cache	
4K	0.0779
8K	0.0572
16K	0.0225
32K	0.0131
64K	0.012

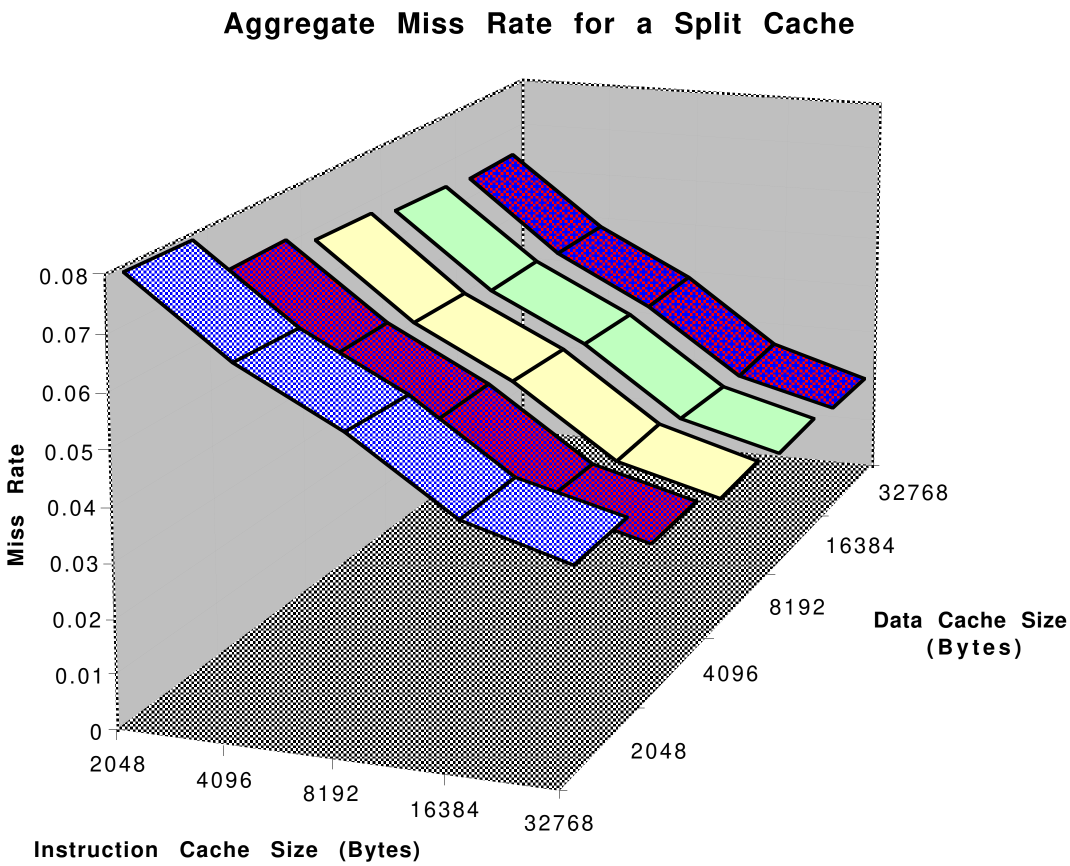


tex.100k Miss Rates

Data Cache Size (K)

Instruction Cache Size (K)		2	4	8	16	32
	2	0.0069	0.0059	0.0057	0.0051	0.0048
	4	0.0069	0.0059	0.0057	0.0051	0.0048
	8	0.0069	0.0059	0.0057	0.0051	0.0048
	16	0.0069	0.0059	0.0057	0.0051	0.0048
	32	0.0069	0.0059	0.0057	0.0051	0.0048

Unified Cache	
4K	0.0277
8K	0.0058
16K	0.0052
32K	0.0048
64K	0.0048



aggregate.100k Miss Rates

Data Cache Size (K)

Instruction Cache Size (K)		2	4	8	16	32
	2	0.079333	0.070767	0.067667	0.065267	0.064267
	4	0.065167	0.0566	0.053467	0.0511	0.050067
	8	0.055333	0.046767	0.043633	0.041267	0.040233
	16	0.0422	0.0336	0.0305	0.0281	0.0271
	32	0.0369	0.028333	0.025233	0.022833	0.021833

Unified Cache	
4K	0.088533
8K	0.065667
16K	0.039
32K	0.0272
64K	0.022233

Clearly, the more you have of either data or instruction cache, the better off you are.

For benchmarks `cc1` and `spice`, a larger instruction cache is more beneficial to lowering the miss rate than is a larger data cache. This suggests that either there are more instruction accesses than data accesses in these programs, or that the instructions accessed are from such locations that they result in a higher number of conflict misses in the small direct mapped instruction caches than in the larger (also direct mapped) instruction caches. As far as `tex` is concerned, its miss rate is constant with increasing the size of the instruction cache, which means that the 2KB of the smallest instruction cache are sufficient to hold all the instructions. The identical behavior of `cc1` and `spice`, and their higher miss rates compared to `tex`, dominate in determining the aggregate miss rate, which is similarly influenced more by an increase of the instruction cache size than by an increase of the data cache size.

An uneven split will make sense if a program accesses more distinct data locations than instructions (as is the case with benchmark `tex`) or vice versa.

2.2) A unified cache will be a win over a split cache if instructions and data will not map simultaneously to the same cache locations. In such cases, it has the potential of more flexibly allocating the cache space, conditional on the needs of each program.

For all 3 benchmarks, and hence for the aggregate workload, the miss rate of the unified cache is lower than or the same (within 0.1%) as that of the split cache of equal total size, for total cache sizes of 16K, 32K, and 64K. The reverse holds true for most of the smaller cache sizes, as then instructions and data conflict more often for the few locations in the unified cache.

Part III: Cost-Based Optimization

3.1)
Cost = (# of sets) * [(# of blocks per set)*(Total # of bits per block) + (LRU bits per set)]

Cost = S*[E*(8*B + t + 2) + E*(E-1)]

Cost = S*E*(8*B + t + E + 1)

On the other hand, the address is split into t bits of tag, $\log_2 S$ bits of a set index, and $\log_2 B$ bits of a byte index withing the selected block, so that given 32 address bits, the number of tag bits t can be determined from:

$t = (\text{TotalNumberOfAddressBits}) - \log_2 S - \log_2 B = 32 - \log_2 S - \log_2 B$

3.2)
1.1) Cost of cache (bits)

		block size (bytes)						
		4	*8*	*16*	*32*	*64*	*128*	*256*
	1	108544	87040	76288	70912	68224	66880	66208
E	*2*	112640	89088	77312	71424	68480	67008	66272
	4	118784	92160	78848	72192	68864	67200	66368

2.1) Cost of cache (bits): E=1, B=16

Instruction Cache Size (bytes) \ Data Cache Size (bytes)	2048	4096	8192	16384	32768
2048	38656	57728	95616	170880	320384
4096	57728	76800	114688	189952	339456
8192	95616	114688	152576	227840	377344
16384	170880	189952	227840	303104	452608
32768	320384	339456	377344	452608	602112

2.2) Cost of unified cache (bits): E=1, B=16

Unified Cache Size (bytes)

4096	8192	16384	32768	65536
38400	76288	151552	301056	598016

3.3)

The 250Kbit limit does not allow a 32KB unified cache, so that the best configuration is a split cache with 16KB for data and 8KB for instructions, or vice versa.

The optimum configurations are as follows:

Benchmark	Instruction Cache (KB)	Data Cache (KB)	Associativity	Block size (bytes)	Miss rate
cc1.100k	8	16	32	256	1.997%
tex.100k	8	16	4	2048	0.013%
spice.100k	8	16	8	512	0.242%
aggregate	8	16	32	256	0.795%

The `cc1` cache performance is so much poorer than the rest that it becomes the dominating consideration in the aggregate performance.