Assignment 4

Kevin Martin
Albert (Jun) Yang
Daniel Kalish
Syracuse University CIS655
Summer 2020, Tuesday @ 9:00pm EST

Installation

We ran the assignment on Windows 10 which ended up dictating the majority of our setup. First we installed a fresh version of Cygwin and used its terminal for our experiments. Upon installation of Cygwin, we selected the "Dev" package which includes many basic programming tools. Most importantly, it included GCC, which we used for compiling the program. Of course, after every modification to the files, we would rerun the "Make" command to allow for updates.

Next we downloaded the most recent version of SimpleScalar, which happens to be simplesim-3v0e.tgz. Because we were not planning on creating our own test scripts, we did not need the simpletools or simpleutils packages.

Once in our Cygwin terminal, we untarred the .tgz file using the command "tar –xzvf simplesim-3v0e.tgz". This created the "simplesim-3.0" directory where all the C files and tests are housed. Next we issued the command "make config-pisa" to create the PISA-architecture files. Because PISA is closer than the provided ALPHA ISA to MIPS, we thought it would make more sense to focus on that. Finally, we issued the last "make" command to generate our functional directory.

Problem

The problem we wish to observe is what is the optimal number of sets for the level 1 cache. We realized that the number of sets can have drastic impacts to performance and wanted to see where we could find improvements.

Testing

In order to test, we used the sim-cache simulation and ran seven different set numbers across three of the provided tests, for a total of 21 runs. For batch sizes, we chose standard powers of two, beginning with 32 and ending at 2048. The multiple tests helped ensure our results were not anomalies. In looking through the little endian tests, we didn't really know what each test did, so we tested them all out. For our purposes, test-math, test-fmath, and test-llong all provided sufficient trials. We found it quite surprising when we ran test-lswlr only to find it prints "hello world."

When running each scenario, we simply used the command line to change each run. We were able to select the scenario, test, modify the cache to our needs, and select the output directory for the results. The actual test we just allowed to print into the terminal (as it was not necessary for our analysis). A sample command would be as follows:

./sim-cache -cache:il1 il1:**32**:32:1:l -redir:sim labresults/math1_**32**.out ./tests-pisa/bin.little/**test-math**

Note the bolded red items which we modified on each subsequent run. Once all runs had been completed, we exported each .out file into an Excel spreadsheet and graphed the results. This helped to visualize the effect set number had.

One interesting note is that, of the 68 items SimpleScalar records, only eight were impacted by set number: il1.hits, il1.misses, il1.replacements, il1.miss_rate, il1.repl_rate, ul2.accesses, ul2.hits, and ul2.miss_rate. None of the three tests looked at things involving writebacks or replacements. Additionally, the "sim_elapsed_time" for every run was always "1", leading us to believe the program just rounded up to the nearest second. Our machines yielded almost instant results, certainly no time delay apparent to the human eye. Finally, we removed simulation-specific differences, so for example "sim_num_insn" is not examined. Here we see that each test has a different number of instances to perform, but because all are well into the thousands, we believe this to be irrelevant. In short, test-fmath had about 1.8x as many operations as llong, and math had about 4x as many as fmath, or about 7.2x as many as llong. Overall, we fell as though each test is representative and should be included.

Results

Fortunately, all three tests provided mostly the same result, just with varying degrees of intensity. This leads us to believe that our conclusions are sound. Across the board, we see improvements for all things related to the instruction level 1 cache as the number of sets were increased. Hits go up while replacements, miss rate, and replacement rate all go down, which is the ideal direction for each test. We also note that the more instructions each test has, in general, the more pronounced the impact. So for example on misses, test-math (the longest test) saw the sharpest decline in misses as more sets were added while test-llong (the shortest test) had a more tempered decline.

The exact opposite can be said for the (unified) level 2 cache: hits decline and the miss rate increases. However, it should be noted that while the trend is poor, the overall number of hits is an order of magnitude less than the hits for the level 1 cache. One benefit we did note is that the number of accesses decreases at level 2 as we increase the sets in level 1. This makes sense intuitively because as the performance at level 1 increases, the need to access level 2 should decrease.

Overall, we conclude that adding more sets is the overwhelming decision to maximize performance. Despite the decline in performance at the second level, the gain at the first level is not only more dramatic, but also more important to the system overall. The level 1 cache is closer to the processor, times are faster, overhead is lower, so the goal should always be to have as many level 1 cache hits as possible. Our only caveat is that, in the real world, it may not be feasible to implement such high sets in a level 1 cache. We can certainly understand how cost and/or physical size may limit the ability to put this into action. In looking across all the graphs, we do note mild inflection points, but in general we would recommend to increase the set size as much as possible while still maintaining any budgetary concerns.

Screenshots

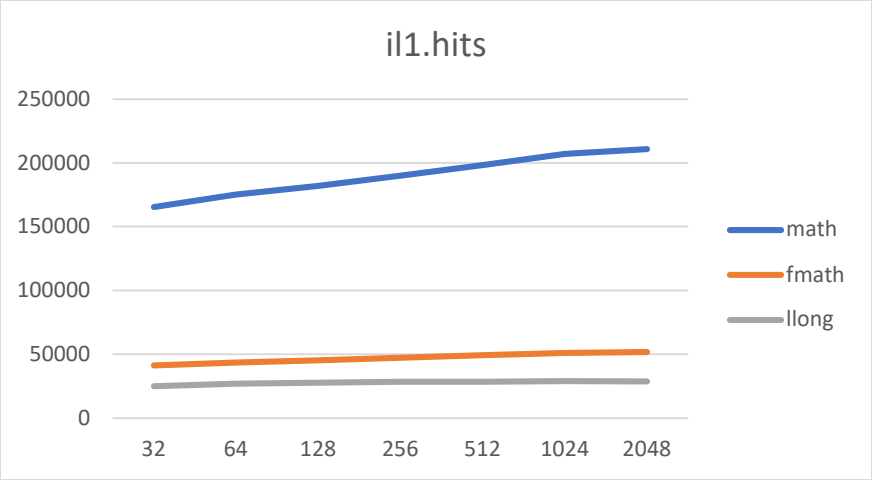
Sample output of a single run with full command line arguments:

```
Captain@DESKTOP-HHVOQFA /simplesim-3.0
$ ./sim-cache -cache:il1 il1:32:32:1:l -redir:sim labresults/math1_32.out ./tests-pisa/bin.little/test-math
pow(12.0, 2.0) == 144.000000
pow(10.0, 3.0) == 1000.000000
pow(10.0, -3.0) == 0.001000
str: 123.456
x: 123.000000
str: 123.456
x: 123.456000
str: 123.456
x: 123.456000
123.456 123.456000 123 1000
sinh(2.0) = 3.62686
sinh(3.0) = 10.01787
h=3.60555
atan2(3,2) = 0.98279
pow(3.60555,4.0) = 169
169 / exp(0.98279 * 5) = 1.24102
3.93117 + 5*log(3.60555) = 10.34355
cos(10.34355) = -0.6068,  sin(10.34355) = -0.79486
x      0.5x
x0.5      x
x    0.5x
-1e-17 != -1e-17 Worked!
```

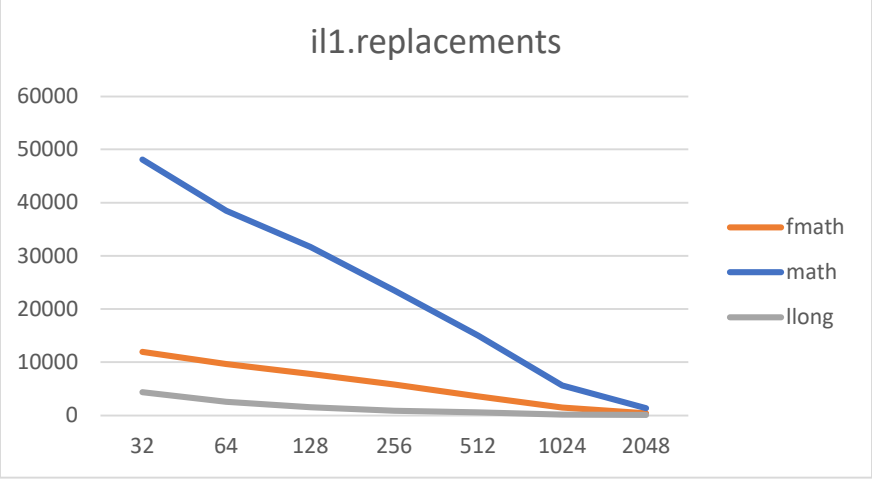
Summary of results (partial, full sheet continues):

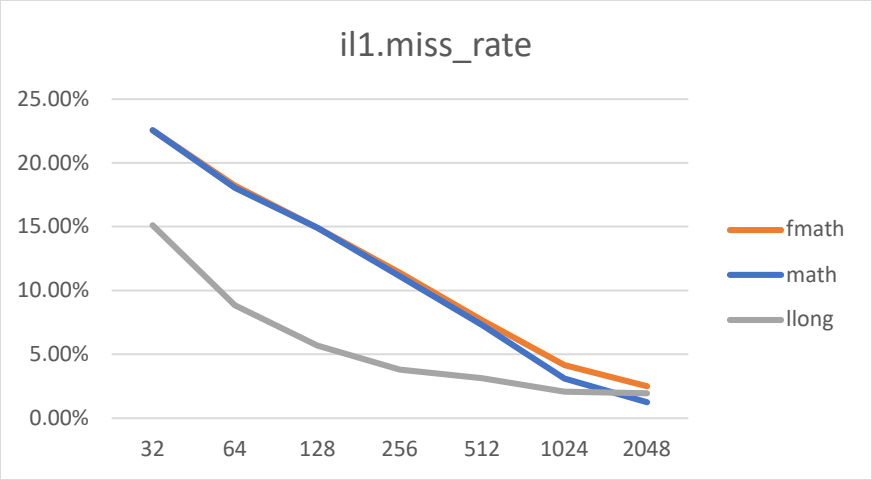
	Sets in the cache (block size: 32) test-math							Sets in the cache (block size: 32) test-fmath									Sets in the cache (block size: 32) test-llong					
Test	32	64	128	256	512	1024	2048	32	64	128	256	512	1024	2048	32	64	128	256	512	1024	2048
sim_num_insn	213586	213586	213586	213586	213586	213586	213586	53345	53345	53345	53345	53345	53345	53345	29528	29528	29528	29528	29528	29528	29528
sim_num_refs	56849	56849	56849	56849	56849	56849	56849	16294	16294	16294	16294	16294	16294	16294	10137	10137	10137	10137	10137	10137	10137
sim_elapsed_time	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
sim_inst_rate	213586	213586	213586	213586	213586	213586	213586	53345	53345	53345	53345	53345	53345	53345	29528	29528	29528	29528	29528	29528	29528
il1.accesses	213586	213586	213586	213586	213586	213586	213586	53345	53345	53345	53345	53345	53345	53345	29528	29528	29528	29528	29528	29528	29528
il1.hits	165407	175057	181761	189828	198030	206984	210892	41310	43637	45389	47268	49264	51133	52005	25066	26915	27855	28408	28608	28913	28953
il1.misses	48179	38529	31825	23758	15556	6602	2694	12035	9708	7956	6077	4081	2212	1340	4462	2613	1673	1120	920	615	575
il1.replacements	48147	38465	31697	23502	15046	5646	1394	12003	9644	7828	5821	3597	1459	432	4430	2549	1545	875	571	170	117
il1.writebacks	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
il1.invalidations	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
il1.miss_rate	0.2256	0.1804	0.149	0.1112	0.0728	0.0309	0.0126	0.2256	0.182	0.1491	0.1139	0.0765	0.0415	0.0251	0.1511	0.0885	0.0567	0.0379	0.0312	0.0208	0.0195
il1.repl_rate	0.2254	0.1801	0.1484	0.11	0.0704	0.0264	0.0065	0.225	0.1808	0.1467	0.1091	0.0674	0.0274	0.0081	0.15	0.0863	0.0523	0.0296	0.0193	0.0058	0.004
il1.wb_rate	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
il1.inv_rate	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
dl1.accesses	57424	57424	57424	57424	57424	57424	57424	16597	16597	16597	16597	16597	16597	16597	10431	10431	10431	10431	10431	10431	10431
dl1.hits	56624	56624	56624	56624	56624	56624	56624	16023	16023	16023	16023	16023	16023	16023	9957	9957	9957	9957	9957	9957	9957
dl1.misses	800	800	800	800	800	800	800	574	574	574	574	574	574	574	474	474	474	474	474	474	474
dl1.replacements	544	544	544	544	544	544	544	318	318	318	318	318	318	318	218	218	218	218	218	218	218
dl1.writebacks	411	411	411	411	411	411	411	263	263	263	263	263	263	263	200	200	200	200	200	200	200
dl1.invalidations	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
dl1.miss_rate	0.0139	0.0139	0.0139	0.0139	0.0139	0.0139	0.0139	0.0346	0.0346	0.0346	0.0346	0.0346	0.0346	0.0346	0.0454	0.0454	0.0454	0.0454	0.0454	0.0454	0.0454
dl1.repl_rate	0.0095	0.0095	0.0095	0.0095	0.0095	0.0095	0.0095	0.0192	0.0192	0.0192	0.0192	0.0192	0.0192	0.0192	0.0209	0.0209	0.0209	0.0209	0.0209	0.0209	0.0209
dl1.wb_rate	0.0072	0.0072	0.0072	0.0072	0.0072	0.0072	0.0072	0.0158	0.0158	0.0158	0.0158	0.0158	0.0158	0.0158	0.0192	0.0192	0.0192	0.0192	0.0192	0.0192	0.0192
dl1.inv_rate	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ul2.accesses	49390	39740	33036	24969	16767	7813	3905	12872	10545	8793	6914	4918	3049	2177	5136	3287	2347	1794	1594	1289	1249
ul2.hits	48203	38553	31849	23782	15580	6626	2718	12044	9717	7965	6086	4090	2221	1349	4621	2772	1832	1279	1079	774	734
ul2.misses	1187	1187	1187	1187	1187	1187	1187	828	828	828	828	828	828	828	515	515	515	515	515	515	515
ul2.replacements	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ul2.writebacks	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ul2.invalidations	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ul2.miss_rate	0.024	0.0299	0.0359	0.0475	0.0708	0.1519	0.304	0.0643	0.0785	0.0942	0.1198	0.1684	0.2716	0.3803	0.1003	0.1567	0.2194	0.2871	0.3231	0.3995	0.4123
ul2.repl_rate	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ul2.wb_rate	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ul2.inv_rate	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
itlb.accesses	213586	213586	213586	213586	213586	213586	213586	53345	53345	53345	53345	53345	53345	53345	29528	29528	29528	29528	29528	29528	29528
itlb.hits	213563	213563	213563	213563	213563	213563	213563	53325	53325	53325	53325	53325	53325	53325	29516	29516	29516	29516	29516	29516	29516

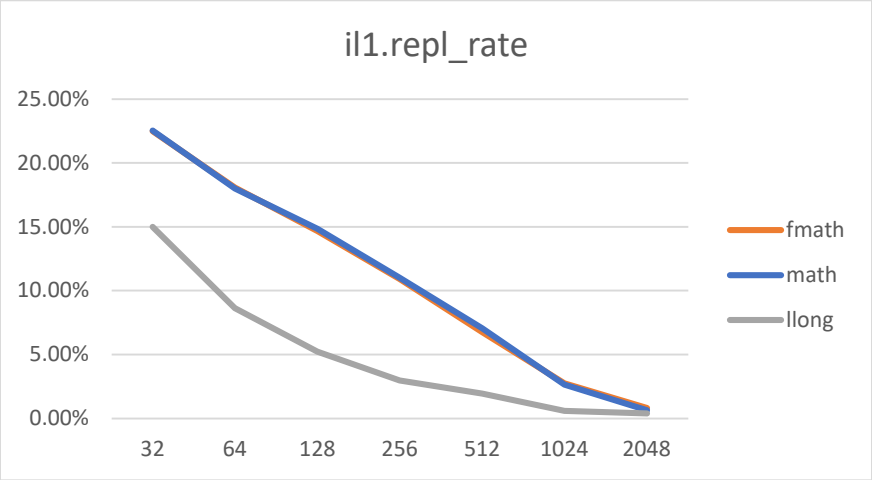
Level 1 graph results for all eight items impacted:

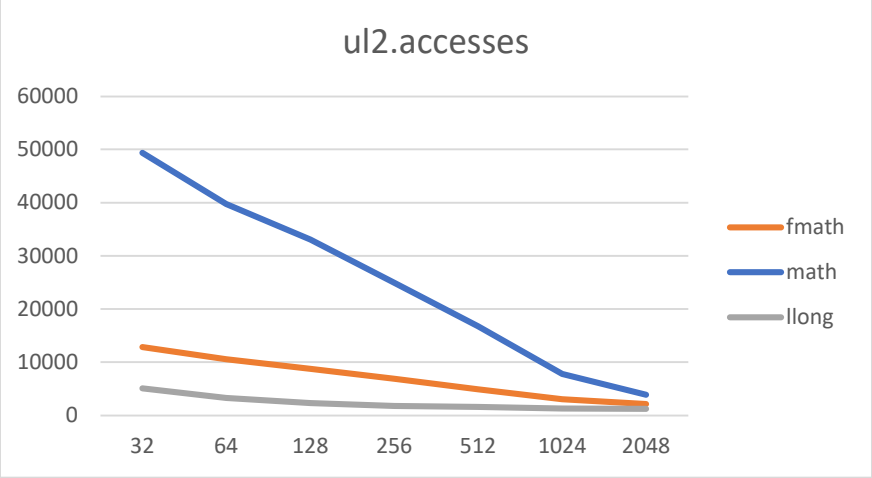

il1.hits
250000
200000
150000
100000
50000
0
32
64
128
256
512
1024
2048
math
fmath
llong

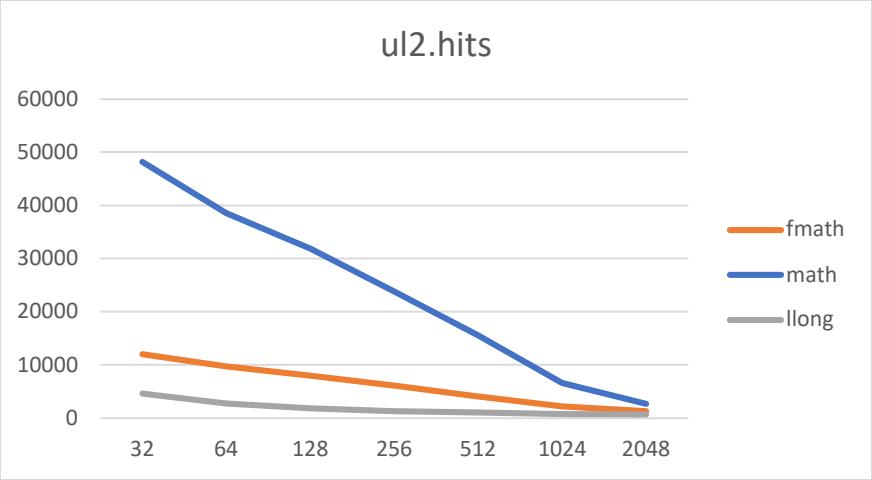

il1.misses
60000
50000
40000
30000
20000
10000
0
32
64
128
256
512
1024
2048
math
fmath
llong

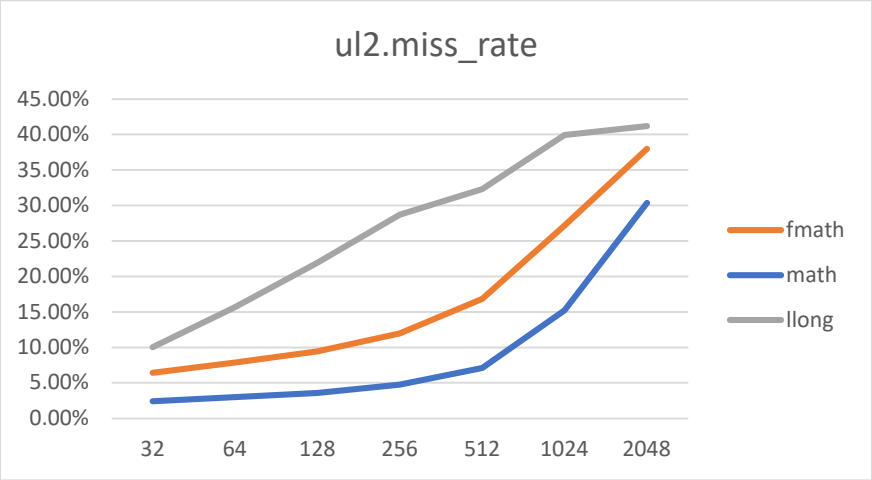

il1.replacements
60000
50000
40000
30000
20000
10000
0
32
64
128
256
512
1024
2048
fmath
math
llong


il1.miss_rate
25.00%
20.00%
15.00%
10.00%
5.00%
0.00%
32
64
128
256
512
1024
2048
fmath
math
llong


il1.repl_rate
25.00%
20.00%
15.00%
10.00%
5.00%
0.00%
32
64
128
256
512
1024
2048
fmath
math
llong


ul2.accesses
60000
50000
40000
30000
20000
10000
0
32
64
128
256
512
1024
2048
fmath
math
llong


ul2.hits
60000
50000
40000
30000
20000
10000
0
32
64
128
256
512
1024
2048
fmath
math
llong


ul2.miss_rate
45.00%
40.00%
35.00%
30.00%
25.00%
20.00%
15.00%
10.00%
5.00%
0.00%
32
64
128
256
512
1024
2048
fmath
math
llong