



Technical University of Denmark

F19
02346
DTU DIPLOM
DISTRIBUTED AND PARALLEL SYSTEMS
MANDATORY ASSIGNMENT 1
GRUPPE 17
21. FEBRUAR 2019



JOACHIM S. MORTENSEN
s175179



MIKAEL KRISTENSEN
s094766



NICOLAI KAMMERSGÅRD
s143780

Problem 1 (group): Clocks

Handed-in C File: *timing.c*

We implemented the different types of clocks in the 3 given methods and tested each clock method 2 times, with 1000*1000 operations and 1000000*1000000 operations, we got the following elapsed time:

1000*1000 operations, elapsed time in seconds:

[illegible]

Estimated resolution in nanoseconds:

gettimeofday	4.788875580
clock_gettime(CLOCK_REALTIME)	3.239393234
clock_gettime(CLOCK_MONOTONIC)	3.159250133

10000*10000 operations, elapsed time in seconds:

[illegible]

Estimated resolution in nanoseconds:

gettimeofday	2.897350788
clock_gettime(CLOCK_REALTIME)	2.806341648
clock_gettime(CLOCK_MONOTONIC)	2.809610129

So what we see is we get an resolution going towards 2ns. We can get the actual resolution for the two clocks of `clock_gettime` by using the method `clock_getres`. This gives us a resolution of 1 nanosecond. This is a hardcoded value as these clocks are "high-resolution". A clock that isn't high resolution would return 1/Hz instead. as we can see the estimate is a bit off, but actually quite close.

Which timing function would you suggest to use? `gettimeofday` is marked as obsolete and `clock_gettime` is there as its successor and recommended to use instead. With regards to `clock_gettime`, we would rather use the id `CLOCK_MONOTONIC` is the more reliable clock as it increases linearly while the clock `CLOCK_REALTIME` has a chance of leaping forward or backwards and can be set by the function `clock_settime`.

What do the compiler options -O0 -lrt mean? -O0 removes a large number of optimisation passes. -lrt specifies that we want to link with the "rt" library, as -l tells the compiler to link with the library name that follows.

How would you expect the timing to behave with varying workload if the clock had a resolution of say 100 μs (microseconds)? Did you see any such effect in your experiments? If not, why not so? The less accurate the clock the larger the workload has to be to get an accurate timing. We can see this our estimate with `gettimeofday`, as this clock is far less accurate than either of the other two timing functions, so the estimate on small workloads is more off compared to the other two. As the workload increases the accuracy of our estimate does as well, as shown in the 10000 * 10000 example where the `gettimeofday` clock is fairly accurate.

Problem 2 (group): Cache performance

Handed-in C File: *cachetest.c*, *matrix_sum.c*

In Appendix A and Appendix B we see the different runs of cache test with different timing functions. Appendix B uses the given `get_seconds` method which relies on cpu cycles and counting how many of them have gone by. Appendix A uses the `clock_gettime` with the clock id `CLOCK_MONOTONIC_RAW`, which we met in exercise 1.4. Comparing the two runs, we see that the read+write time have a bigger variance when using clock cycles rather than the `clock_gettime`. The variance makes it harder to spot the switches between caches, why the `clock_gettime` is preferable for spotting cache misses.

The file *cachetest.c* contains 2 do-while loops. The first do while loop runs for half a second and tries to make as many access to memory as possible in this timeframe. the memory spaces which are accessed are offset by the stride value. Then the second do while makes the same amount of operations as the first do while, specified by the "steps" variable, but this time does not access memory but the dummy value from the register. This gives us the overhead time, and if we subtract the runtime of the second do while from the first we get the actual time taken to read/write to memory. This time in nanoseconds is what is printed in the read+write column.

Many of the variables have the "register" keyword in their definitions which suggests to the compiler that the variable will be heavily used and accessed so keeping it in a register would be a good idea so the processor doesn't have to constantly fetch the value of the variables from memory. This reduces the runtime of the program drastically.

In Appendix A we see the output from 1 run of the *cachetest* file. In this output we can see from line 1 to 40 a read+write time of under 0.5ns. Once the cache size increases to 64KB we have an increase in read+write time, as the whole dataset can no longer be loaded into the L1 cache at the same time. This means that the L1 cache should be around 32KB.

The cache-misses seem to start when we stride 64 bytes. This tells us that the cache line size is about 64 bytes as we keep missing.

We see the same pattern occurring again at a cache size of around 32MB where the the read+write time goes from around 2-3ns i L2 cache to around 4-9ns. This should mean that we now have gone from the L2 cache to either L3 or main memory, and that the L2 cache is of size 16MB. We believe that there are three levels of cache as accessing the main memory would take much longer than the 10ns we get when we miss our L2 cache size.

Summing Matrices and the impact of caching

Running the two sum functions clearly shows that `sum1` which iterates over each column in each row in succession, is way better performing than the `sum2` function, which does the opposite (iterates over each row in each column in succession). `sum1` is faster than `sum2` by around a factor 8 because of the way arrays are stored in memory. `sum1` runs through memory sequentially while `sum2` takes big strides in memory locations, leading to a lot of cache misses. The smaller the matrix the less impact this has as we will get less and less cache misses.

How do you need to write your programs to suit the memory hierarchy? To maximize performance when accessing memory one should try to optimize ones data structure so that the accessing can happen sequentially.

Problem 3 (individual): Java vs C

s094766

C is a function language which means that the need for breaking code into functions is quite important, since there is no possibility to define functionality in different classes and reference it as objects.

Java is compiled into java bytecode while C is compiled into assembly.

Java works with references which gives some degree of access to memory, however in C we have full access to memory, while being able to manipulate the reference/pointer. This gives many more possibilities in how you can reference memory spaces, however comes with way more responsibility.

Java runs its bytecode on a virtual machine, a layer of abstraction between the processor and the code.

Java has a garbage collector that takes care of memory management for you, whereas in C you are fully responsible for cleaning your memory usage.

Problem 4 (individual): MSI Cache Protocol

s143780

The different steps is shown below.

	M	Mb	C1	C1s	C2	C2s
Step 0	x=5	null	null	I	null	I
Step 1a	x=5	BusRd	null	I	null	I
Step 1b	x=5	null	x=5	S	null	I
Step 2a	x=5	BusRd	x=5	S	null	I
Step 2b	x=5	null	x=5	S	x=5	S
Step 3a	x=5	BusUpgr	x=5	S	x=5	S
Step 3b	x=5	null	x=6	M	null	I
Step 4a	x=5	null	x=6	M	null	I
Step 5a	x=5	BusRdX	x=6	M	null	I
Step 5b	x=6	null	null	I	x=6	M
Step 5c	x=6	null	null	I	x=8	M
Step 6a	x=6	BusRd	null	I	x=8	M
Step 6b	x=8	null	x=8	S	x=8	S

A pro of using the MSI Cache protocol is that you ensure that the data is aligned across caches, and therefore can scale the access to same memory locations quite well. A con is that it takes a lot of traffic on the bus to keep this data coherence. The MOSI, MESI and MOESI protocols solve some of this extra traffic by adding more states.

Problem 5 (individual): Omega switching network

s175179

1. State which mode setting would be needed for a message to be sent from processor 0 to memory module 6? (1A crossed, . . .)

To get to module 6 from CPU0, the switch 1A has to be in crossed mode, the switch 2B has to be in straight mode and finally the switch 3C in crossed mode.

2. State which mode setting would be needed for a message to be sent from processor 6 to memory module 2?

To get to module 2 from CPU6, the switch 1B has to be in crossed mode, the switch 2C has to be in straight mode and finally the switch 3A in straight mode.

3. Given these settings, would be possible to send a message from processor 2 to processor 5?

No, as seen in Appendix C, they do not share any memory banks.

4. Again given the settings of 1. and 2., which memory modules can be reached from processor 1?

Memory bank 6, 7 and 8 can be reached from CPU1 with the given settings.

5. If all switches are set straight, how does the network connect processors and memory modules?

1 to 1, 2 to 2, 3 to 3 etc. See Appendix D.

6. Is it possible to connect processor no. i with memory module $(i+1) \bmod 8$ for all processors at the same time?

Yes it is possible when the switches are configured as follows:

1A straight, 1B straight, 1C straight, 1D crossed.

2A straight, 2B straight, 2C crossed, 2D crossed.

3A crossed, 3B crossed, 3C crossed, 3D crossed.

See Appendix E.

Appendix A - CacheTest run on n-62-27-23

1 Size (bytes): 4096, Stride (bytes): 4, read+write: 0.32 ns (steps: 499)
2 Size (bytes): 4096, Stride (bytes): 8, read+write: 0.30 ns (steps: 503)
3 Size (bytes): 4096, Stride (bytes): 16, read+write: 0.29 ns (steps: 498)
4 Size (bytes): 4096, Stride (bytes): 32, read+write: 0.26 ns (steps: 492)
5 Size (bytes): 4096, Stride (bytes): 64, read+write: 0.20 ns (steps: 464)
6 Size (bytes): 4096, Stride (bytes): 128, read+write: 0.26 ns (steps: 378)
7 Size (bytes): 4096, Stride (bytes): 256, read+write: 0.35 ns (steps: 375)
8 Size (bytes): 4096, Stride (bytes): 512, read+write: 0.32 ns (steps: 382)
9 Size (bytes): 4096, Stride (bytes): 1024, read+write: 0.66 ns (steps: 380)
10 Size (bytes): 8192, Stride (bytes): 4, read+write: 0.31 ns (steps: 508)
11 Size (bytes): 8192, Stride (bytes): 8, read+write: 0.31 ns (steps: 507)
12 Size (bytes): 8192, Stride (bytes): 16, read+write: 0.29 ns (steps: 503)
13 Size (bytes): 8192, Stride (bytes): 32, read+write: 0.28 ns (steps: 499)
14 Size (bytes): 8192, Stride (bytes): 64, read+write: 0.25 ns (steps: 494)
15 Size (bytes): 8192, Stride (bytes): 128, read+write: 0.20 ns (steps: 471)
16 Size (bytes): 8192, Stride (bytes): 256, read+write: 0.24 ns (steps: 384)
17 Size (bytes): 8192, Stride (bytes): 512, read+write: 0.33 ns (steps: 380)
18 Size (bytes): 8192, Stride (bytes): 1024, read+write: 0.33 ns (steps: 377)
19 Size (bytes): 16384, Stride (bytes): 4, read+write: 0.32 ns (steps: 499)
20 Size (bytes): 16384, Stride (bytes): 8, read+write: 0.32 ns (steps: 502)
21 Size (bytes): 16384, Stride (bytes): 16, read+write: 0.30 ns (steps: 507)
22 Size (bytes): 16384, Stride (bytes): 32, read+write: 0.29 ns (steps: 506)
23 Size (bytes): 16384, Stride (bytes): 64, read+write: 0.29 ns (steps: 494)
24 Size (bytes): 16384, Stride (bytes): 128, read+write: 0.24 ns (steps: 492)
25 Size (bytes): 16384, Stride (bytes): 256, read+write: 0.19 ns (steps: 472)
26 Size (bytes): 16384, Stride (bytes): 512, read+write: 0.25 ns (steps: 377)
27 Size (bytes): 16384, Stride (bytes): 1024, read+write: 0.34 ns (steps: 373)
28 Size (bytes): 32768, Stride (bytes): 4, read+write: 0.33 ns (steps: 497)
29 Size (bytes): 32768, Stride (bytes): 8, read+write: 0.32 ns (steps: 497)
30 Size (bytes): 32768, Stride (bytes): 16, read+write: 0.32 ns (steps: 496)
31 Size (bytes): 32768, Stride (bytes): 32, read+write: 0.32 ns (steps: 496)
32 Size (bytes): 32768, Stride (bytes): 64, read+write: 0.32 ns (steps: 491)
33 Size (bytes): 32768, Stride (bytes): 128, read+write: 0.30 ns (steps: 484)
34 Size (bytes): 32768, Stride (bytes): 256, read+write: 0.30 ns (steps: 458)
35 Size (bytes): 32768, Stride (bytes): 512, read+write: 0.31 ns (steps: 422)
36 Size (bytes): 32768, Stride (bytes): 1024, read+write: 0.31 ns (steps: 365)
37 Size (bytes): 65536, Stride (bytes): 4, read+write: 0.32 ns (steps: 502)
38 Size (bytes): 65536, Stride (bytes): 8, read+write: 0.31 ns (steps: 507)
39 Size (bytes): 65536, Stride (bytes): 16, read+write: 0.32 ns (steps: 509)
40 Size (bytes): 65536, Stride (bytes): 32, read+write: 0.39 ns (steps: 468)
41 Size (bytes): 65536, Stride (bytes): 64, read+write: 1.41 ns (steps: 238)
42 Size (bytes): 65536, Stride (bytes): 128, read+write: 1.42 ns (steps: 236)
43 Size (bytes): 65536, Stride (bytes): 256, read+write: 1.45 ns (steps: 230)
44 Size (bytes): 65536, Stride (bytes): 512, read+write: 1.37 ns (steps: 233)
45 Size (bytes): 65536, Stride (bytes): 1024, read+write: 2.76 ns (steps: 137)
46 Size (bytes): 131072, Stride (bytes): 4, read+write: 0.33 ns (steps: 498)
47 Size (bytes): 131072, Stride (bytes): 8, read+write: 0.33 ns (steps: 498)
48 Size (bytes): 131072, Stride (bytes): 16, read+write: 0.33 ns (steps: 496)

49 Size (bytes): 131072, Stride (bytes): 32, read+write: 0.42 ns (steps: 457)
50 Size (bytes): 131072, Stride (bytes): 64, read+write: 1.45 ns (steps: 235)
51 Size (bytes): 131072, Stride (bytes): 128, read+write: 1.47 ns (steps: 232)
52 Size (bytes): 131072, Stride (bytes): 256, read+write: 1.44 ns (steps: 235)
53 Size (bytes): 131072, Stride (bytes): 512, read+write: 1.39 ns (steps: 237)
54 Size (bytes): 131072, Stride (bytes): 1024, read+write: 2.80 ns (steps: 141)
55 Size (bytes): 262144, Stride (bytes): 4, read+write: 0.32 ns (steps: 506)
56 Size (bytes): 262144, Stride (bytes): 8, read+write: 0.31 ns (steps: 508)
57 Size (bytes): 262144, Stride (bytes): 16, read+write: 0.39 ns (steps: 472)
58 Size (bytes): 262144, Stride (bytes): 32, read+write: 0.74 ns (steps: 356)
59 Size (bytes): 262144, Stride (bytes): 64, read+write: 1.90 ns (steps: 195)
60 Size (bytes): 262144, Stride (bytes): 128, read+write: 1.91 ns (steps: 193)
61 Size (bytes): 262144, Stride (bytes): 256, read+write: 1.90 ns (steps: 194)
62 Size (bytes): 262144, Stride (bytes): 512, read+write: 1.72 ns (steps: 207)
63 Size (bytes): 262144, Stride (bytes): 1024, read+write: 2.94 ns (steps: 136)
64 Size (bytes): 524288, Stride (bytes): 4, read+write: 0.32 ns (steps: 509)
65 Size (bytes): 524288, Stride (bytes): 8, read+write: 0.33 ns (steps: 503)
66 Size (bytes): 524288, Stride (bytes): 16, read+write: 0.44 ns (steps: 455)
67 Size (bytes): 524288, Stride (bytes): 32, read+write: 1.05 ns (steps: 291)
68 Size (bytes): 524288, Stride (bytes): 64, read+write: 2.35 ns (steps: 166)
69 Size (bytes): 524288, Stride (bytes): 128, read+write: 2.58 ns (steps: 154)
70 Size (bytes): 524288, Stride (bytes): 256, read+write: 2.64 ns (steps: 151)
71 Size (bytes): 524288, Stride (bytes): 512, read+write: 2.25 ns (steps: 171)
72 Size (bytes): 524288, Stride (bytes): 1024, read+write: 2.38 ns (steps: 163)
73 Size (bytes): 1048576, Stride (bytes): 4, read+write: 0.32 ns (steps: 501)
74 Size (bytes): 1048576, Stride (bytes): 8, read+write: 0.31 ns (steps: 507)
75 Size (bytes): 1048576, Stride (bytes): 16, read+write: 0.44 ns (steps: 452)
76 Size (bytes): 1048576, Stride (bytes): 32, read+write: 1.05 ns (steps: 291)
77 Size (bytes): 1048576, Stride (bytes): 64, read+write: 2.36 ns (steps: 165)
78 Size (bytes): 1048576, Stride (bytes): 128, read+write: 2.63 ns (steps: 152)
79 Size (bytes): 1048576, Stride (bytes): 256, read+write: 2.79 ns (steps: 145)
80 Size (bytes): 1048576, Stride (bytes): 512, read+write: 2.22 ns (steps: 173)
81 Size (bytes): 1048576, Stride (bytes): 1024, read+write: 2.40 ns (steps: 163)
82 Size (bytes): 2097152, Stride (bytes): 4, read+write: 0.32 ns (steps: 506)
83 Size (bytes): 2097152, Stride (bytes): 8, read+write: 0.32 ns (steps: 502)
84 Size (bytes): 2097152, Stride (bytes): 16, read+write: 0.43 ns (steps: 455)
85 Size (bytes): 2097152, Stride (bytes): 32, read+write: 1.04 ns (steps: 292)
86 Size (bytes): 2097152, Stride (bytes): 64, read+write: 2.39 ns (steps: 164)
87 Size (bytes): 2097152, Stride (bytes): 128, read+write: 2.67 ns (steps: 150)
88 Size (bytes): 2097152, Stride (bytes): 256, read+write: 2.83 ns (steps: 144)
89 Size (bytes): 2097152, Stride (bytes): 512, read+write: 2.24 ns (steps: 172)
90 Size (bytes): 2097152, Stride (bytes): 1024, read+write: 2.41 ns (steps: 163)
91 Size (bytes): 4194304, Stride (bytes): 4, read+write: 0.33 ns (steps: 504)
92 Size (bytes): 4194304, Stride (bytes): 8, read+write: 0.33 ns (steps: 503)
93 Size (bytes): 4194304, Stride (bytes): 16, read+write: 0.42 ns (steps: 458)
94 Size (bytes): 4194304, Stride (bytes): 32, read+write: 1.04 ns (steps: 292)
95 Size (bytes): 4194304, Stride (bytes): 64, read+write: 2.39 ns (steps: 164)
96 Size (bytes): 4194304, Stride (bytes): 128, read+write: 2.66 ns (steps: 151)
97 Size (bytes): 4194304, Stride (bytes): 256, read+write: 2.81 ns (steps: 144)
98 Size (bytes): 4194304, Stride (bytes): 512, read+write: 2.23 ns (steps: 173)

99 Size (bytes): 4194304, Stride (bytes):1024, read+write: 2.53 ns (steps: 156)
 100 Size (bytes): 8388608, Stride (bytes): 4, read+write: 0.32 ns (steps: 501)
 101 Size (bytes): 8388608, Stride (bytes): 8, read+write: 0.32 ns (steps: 504)
 102 Size (bytes): 8388608, Stride (bytes): 16, read+write: 0.43 ns (steps: 453)
 103 Size (bytes): 8388608, Stride (bytes): 32, read+write: 1.04 ns (steps: 292)
 104 Size (bytes): 8388608, Stride (bytes): 64, read+write: 2.37 ns (steps: 166)
 105 Size (bytes): 8388608, Stride (bytes): 128, read+write: 2.67 ns (steps: 150)
 106 Size (bytes): 8388608, Stride (bytes): 256, read+write: 2.83 ns (steps: 143)
 107 Size (bytes): 8388608, Stride (bytes): 512, read+write: 2.25 ns (steps: 172)
 108 Size (bytes): 8388608, Stride (bytes):1024, read+write: 2.78 ns (steps: 145)
 109 Size (bytes): 16777216, Stride (bytes): 4, read+write: 0.33 ns (steps: 505)
 110 Size (bytes): 16777216, Stride (bytes): 8, read+write: 0.32 ns (steps: 507)
 111 Size (bytes): 16777216, Stride (bytes): 16, read+write: 0.43 ns (steps: 458)
 112 Size (bytes): 16777216, Stride (bytes): 32, read+write: 1.05 ns (steps: 293)
 113 Size (bytes): 16777216, Stride (bytes): 64, read+write: 2.36 ns (steps: 166)
 114 Size (bytes): 16777216, Stride (bytes): 128, read+write: 2.68 ns (steps: 150)
 115 Size (bytes): 16777216, Stride (bytes): 256, read+write: 2.84 ns (steps: 142)
 116 Size (bytes): 16777216, Stride (bytes): 512, read+write: 2.25 ns (steps: 171)
 117 Size (bytes): 16777216, Stride (bytes):1024, read+write: 3.03 ns (steps: 136)
 118 Size (bytes): 33554432, Stride (bytes): 4, read+write: 0.36 ns (steps: 488)
 119 Size (bytes): 33554432, Stride (bytes): 8, read+write: 0.41 ns (steps: 461)
 120 Size (bytes): 33554432, Stride (bytes): 16, read+write: 0.85 ns (steps: 330)
 121 Size (bytes): 33554432, Stride (bytes): 32, read+write: 2.04 ns (steps: 185)
 122 Size (bytes): 33554432, Stride (bytes): 64, read+write: 4.67 ns (steps: 94)
 123 Size (bytes): 33554432, Stride (bytes): 128, read+write: 6.01 ns (steps: 75)
 124 Size (bytes): 33554432, Stride (bytes): 256, read+write: 6.16 ns (steps: 74)
 125 Size (bytes): 33554432, Stride (bytes): 512, read+write: 5.63 ns (steps: 80)
 126 Size (bytes): 33554432, Stride (bytes):1024, read+write: 5.92 ns (steps: 76)
 127 Size (bytes): 67108864, Stride (bytes): 4, read+write: 0.38 ns (steps: 479)
 128 Size (bytes): 67108864, Stride (bytes): 8, read+write: 0.47 ns (steps: 440)
 129 Size (bytes): 67108864, Stride (bytes): 16, read+write: 1.06 ns (steps: 290)
 130 Size (bytes): 67108864, Stride (bytes): 32, read+write: 2.60 ns (steps: 154)
 131 Size (bytes): 67108864, Stride (bytes): 64, read+write: 5.98 ns (steps: 76)
 132 Size (bytes): 67108864, Stride (bytes): 128, read+write: 8.38 ns (steps: 56)
 133 Size (bytes): 67108864, Stride (bytes): 256, read+write: 9.46 ns (steps: 50)
 134 Size (bytes): 67108864, Stride (bytes): 512, read+write: 8.94 ns (steps: 53)
 135 Size (bytes): 67108864, Stride (bytes):1024, read+write: 9.10 ns (steps: 52)
 136 Size (bytes):134217728, Stride (bytes): 4, read+write: 0.39 ns (steps: 475)
 137 Size (bytes):134217728, Stride (bytes): 8, read+write: 0.47 ns (steps: 441)
 138 Size (bytes):134217728, Stride (bytes): 16, read+write: 1.05 ns (steps: 292)
 139 Size (bytes):134217728, Stride (bytes): 32, read+write: 2.61 ns (steps: 153)
 140 Size (bytes):134217728, Stride (bytes): 64, read+write: 6.01 ns (steps: 75)
 141 Size (bytes):134217728, Stride (bytes): 128, read+write: 8.44 ns (steps: 55)
 142 Size (bytes):134217728, Stride (bytes): 256, read+write: 9.56 ns (steps: 49)
 143 Size (bytes):134217728, Stride (bytes): 512, read+write: 8.93 ns (steps: 53)
 144 Size (bytes):134217728, Stride (bytes):1024, read+write: 9.16 ns (steps: 51)

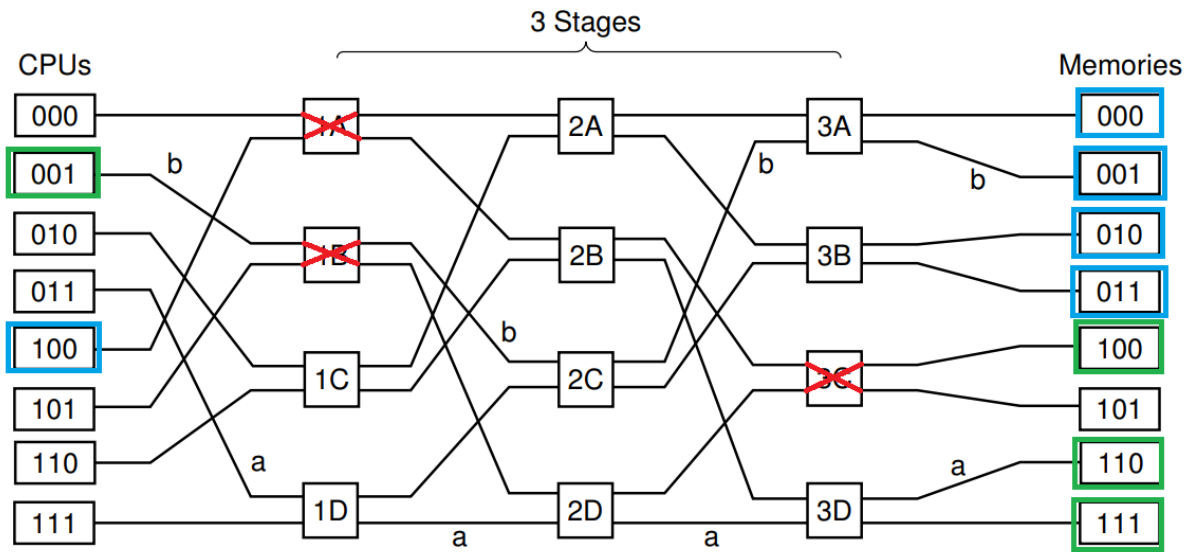
Appendix B - CacheTest run on n-62-27-23

1 Size (bytes): 4096, Stride (bytes): 4, read+write: 0.76 ns (steps: 238)
2 Size (bytes): 4096, Stride (bytes): 8, read+write: 0.81 ns (steps: 235)
3 Size (bytes): 4096, Stride (bytes): 16, read+write: 0.84 ns (steps: 225)
4 Size (bytes): 4096, Stride (bytes): 32, read+write: 0.83 ns (steps: 217)
5 Size (bytes): 4096, Stride (bytes): 64, read+write: 0.77 ns (steps: 207)
6 Size (bytes): 4096, Stride (bytes): 128, read+write: 0.80 ns (steps: 175)
7 Size (bytes): 4096, Stride (bytes): 256, read+write: 0.70 ns (steps: 172)
8 Size (bytes): 4096, Stride (bytes): 512, read+write: 0.76 ns (steps: 172)
9 Size (bytes): 4096, Stride (bytes): 1024, read+write: 0.73 ns (steps: 179)
10 Size (bytes): 8192, Stride (bytes): 4, read+write: 0.80 ns (steps: 237)
11 Size (bytes): 8192, Stride (bytes): 8, read+write: 0.86 ns (steps: 233)
12 Size (bytes): 8192, Stride (bytes): 16, read+write: 0.82 ns (steps: 231)
13 Size (bytes): 8192, Stride (bytes): 32, read+write: 0.85 ns (steps: 224)
14 Size (bytes): 8192, Stride (bytes): 64, read+write: 0.84 ns (steps: 214)
15 Size (bytes): 8192, Stride (bytes): 128, read+write: 0.83 ns (steps: 205)
16 Size (bytes): 8192, Stride (bytes): 256, read+write: 0.93 ns (steps: 172)
17 Size (bytes): 8192, Stride (bytes): 512, read+write: 0.70 ns (steps: 171)
18 Size (bytes): 8192, Stride (bytes): 1024, read+write: 0.68 ns (steps: 177)
19 Size (bytes): 16384, Stride (bytes): 4, read+write: 0.86 ns (steps: 233)
20 Size (bytes): 16384, Stride (bytes): 8, read+write: 0.86 ns (steps: 233)
21 Size (bytes): 16384, Stride (bytes): 16, read+write: 0.87 ns (steps: 229)
22 Size (bytes): 16384, Stride (bytes): 32, read+write: 0.88 ns (steps: 227)
23 Size (bytes): 16384, Stride (bytes): 64, read+write: 0.86 ns (steps: 221)
24 Size (bytes): 16384, Stride (bytes): 128, read+write: 0.83 ns (steps: 218)
25 Size (bytes): 16384, Stride (bytes): 256, read+write: 0.79 ns (steps: 202)
26 Size (bytes): 16384, Stride (bytes): 512, read+write: 0.85 ns (steps: 176)
27 Size (bytes): 16384, Stride (bytes): 1024, read+write: 0.71 ns (steps: 170)
28 Size (bytes): 32768, Stride (bytes): 4, read+write: 0.79 ns (steps: 241)
29 Size (bytes): 32768, Stride (bytes): 8, read+write: 0.87 ns (steps: 231)
30 Size (bytes): 32768, Stride (bytes): 16, read+write: 0.88 ns (steps: 228)
31 Size (bytes): 32768, Stride (bytes): 32, read+write: 0.97 ns (steps: 217)
32 Size (bytes): 32768, Stride (bytes): 64, read+write: 1.17 ns (steps: 205)
33 Size (bytes): 32768, Stride (bytes): 128, read+write: 1.00 ns (steps: 210)
34 Size (bytes): 32768, Stride (bytes): 256, read+write: 0.98 ns (steps: 204)
35 Size (bytes): 32768, Stride (bytes): 512, read+write: 1.00 ns (steps: 190)
36 Size (bytes): 32768, Stride (bytes): 1024, read+write: 0.95 ns (steps: 168)
37 Size (bytes): 65536, Stride (bytes): 4, read+write: 0.75 ns (steps: 241)
38 Size (bytes): 65536, Stride (bytes): 8, read+write: 0.80 ns (steps: 237)
39 Size (bytes): 65536, Stride (bytes): 16, read+write: 0.93 ns (steps: 226)
40 Size (bytes): 65536, Stride (bytes): 32, read+write: 1.01 ns (steps: 218)
41 Size (bytes): 65536, Stride (bytes): 64, read+write: 1.51 ns (steps: 179)
42 Size (bytes): 65536, Stride (bytes): 128, read+write: 1.49 ns (steps: 181)
43 Size (bytes): 65536, Stride (bytes): 256, read+write: 1.41 ns (steps: 177)
44 Size (bytes): 65536, Stride (bytes): 512, read+write: 1.42 ns (steps: 176)
45 Size (bytes): 65536, Stride (bytes): 1024, read+write: 1.54 ns (steps: 156)
46 Size (bytes): 131072, Stride (bytes): 4, read+write: 0.76 ns (steps: 238)
47 Size (bytes): 131072, Stride (bytes): 8, read+write: 0.87 ns (steps: 229)
48 Size (bytes): 131072, Stride (bytes): 16, read+write: 0.96 ns (steps: 219)

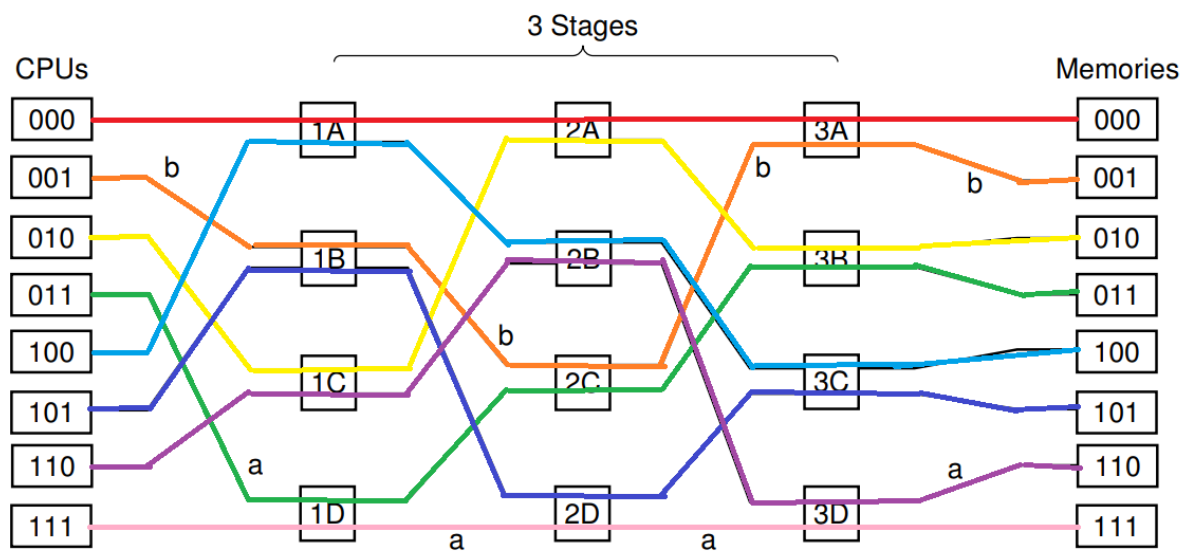
49 Size (bytes): 131072, Stride (bytes): 32, read+write: 0.99 ns (steps: 212)
50 Size (bytes): 131072, Stride (bytes): 64, read+write: 1.53 ns (steps: 177)
51 Size (bytes): 131072, Stride (bytes): 128, read+write: 1.56 ns (steps: 173)
52 Size (bytes): 131072, Stride (bytes): 256, read+write: 1.50 ns (steps: 173)
53 Size (bytes): 131072, Stride (bytes): 512, read+write: 1.47 ns (steps: 177)
54 Size (bytes): 131072, Stride (bytes): 1024, read+write: 1.65 ns (steps: 158)
55 Size (bytes): 262144, Stride (bytes): 4, read+write: 0.86 ns (steps: 232)
56 Size (bytes): 262144, Stride (bytes): 8, read+write: 0.87 ns (steps: 229)
57 Size (bytes): 262144, Stride (bytes): 16, read+write: 1.06 ns (steps: 208)
58 Size (bytes): 262144, Stride (bytes): 32, read+write: 1.41 ns (steps: 184)
59 Size (bytes): 262144, Stride (bytes): 64, read+write: 2.26 ns (steps: 137)
60 Size (bytes): 262144, Stride (bytes): 128, read+write: 2.46 ns (steps: 130)
61 Size (bytes): 262144, Stride (bytes): 256, read+write: 2.31 ns (steps: 134)
62 Size (bytes): 262144, Stride (bytes): 512, read+write: 2.41 ns (steps: 133)
63 Size (bytes): 262144, Stride (bytes): 1024, read+write: 2.68 ns (steps: 123)
64 Size (bytes): 524288, Stride (bytes): 4, read+write: 0.82 ns (steps: 231)
65 Size (bytes): 524288, Stride (bytes): 8, read+write: 0.84 ns (steps: 227)
66 Size (bytes): 524288, Stride (bytes): 16, read+write: 0.95 ns (steps: 220)
67 Size (bytes): 524288, Stride (bytes): 32, read+write: 1.56 ns (steps: 173)
68 Size (bytes): 524288, Stride (bytes): 64, read+write: 3.13 ns (steps: 115)
69 Size (bytes): 524288, Stride (bytes): 128, read+write: 3.36 ns (steps: 107)
70 Size (bytes): 524288, Stride (bytes): 256, read+write: 4.00 ns (steps: 95)
71 Size (bytes): 524288, Stride (bytes): 512, read+write: 3.36 ns (steps: 107)
72 Size (bytes): 524288, Stride (bytes): 1024, read+write: 3.60 ns (steps: 100)
73 Size (bytes): 1048576, Stride (bytes): 4, read+write: 0.95 ns (steps: 220)
74 Size (bytes): 1048576, Stride (bytes): 8, read+write: 0.89 ns (steps: 224)
75 Size (bytes): 1048576, Stride (bytes): 16, read+write: 1.03 ns (steps: 213)
76 Size (bytes): 1048576, Stride (bytes): 32, read+write: 1.47 ns (steps: 177)
77 Size (bytes): 1048576, Stride (bytes): 64, read+write: 2.75 ns (steps: 120)
78 Size (bytes): 1048576, Stride (bytes): 128, read+write: 3.56 ns (steps: 104)
79 Size (bytes): 1048576, Stride (bytes): 256, read+write: 3.70 ns (steps: 100)
80 Size (bytes): 1048576, Stride (bytes): 512, read+write: 3.24 ns (steps: 111)
81 Size (bytes): 1048576, Stride (bytes): 1024, read+write: 3.53 ns (steps: 102)
82 Size (bytes): 2097152, Stride (bytes): 4, read+write: 0.89 ns (steps: 224)
83 Size (bytes): 2097152, Stride (bytes): 8, read+write: 1.12 ns (steps: 205)
84 Size (bytes): 2097152, Stride (bytes): 16, read+write: 1.13 ns (steps: 203)
85 Size (bytes): 2097152, Stride (bytes): 32, read+write: 1.56 ns (steps: 173)
86 Size (bytes): 2097152, Stride (bytes): 64, read+write: 3.36 ns (steps: 107)
87 Size (bytes): 2097152, Stride (bytes): 128, read+write: 3.50 ns (steps: 103)
88 Size (bytes): 2097152, Stride (bytes): 256, read+write: 3.78 ns (steps: 98)
89 Size (bytes): 2097152, Stride (bytes): 512, read+write: 3.12 ns (steps: 112)
90 Size (bytes): 2097152, Stride (bytes): 1024, read+write: 3.50 ns (steps: 103)
91 Size (bytes): 4194304, Stride (bytes): 4, read+write: 0.95 ns (steps: 222)
92 Size (bytes): 4194304, Stride (bytes): 8, read+write: 1.23 ns (steps: 195)
93 Size (bytes): 4194304, Stride (bytes): 16, read+write: 1.62 ns (steps: 167)
94 Size (bytes): 4194304, Stride (bytes): 32, read+write: 1.94 ns (steps: 155)
95 Size (bytes): 4194304, Stride (bytes): 64, read+write: 4.70 ns (steps: 83)
96 Size (bytes): 4194304, Stride (bytes): 128, read+write: 4.64 ns (steps: 84)
97 Size (bytes): 4194304, Stride (bytes): 256, read+write: 4.22 ns (steps: 90)
98 Size (bytes): 4194304, Stride (bytes): 512, read+write: 3.12 ns (steps: 112)

99 Size (bytes): 4194304, Stride (bytes):1024, read+write: 3.43 ns (steps: 105)
100 Size (bytes): 8388608, Stride (bytes): 4, read+write: 1.01 ns (steps: 218)
101 Size (bytes): 8388608, Stride (bytes): 8, read+write: 1.28 ns (steps: 188)
102 Size (bytes): 8388608, Stride (bytes): 16, read+write: 2.37 ns (steps: 135)
103 Size (bytes): 8388608, Stride (bytes): 32, read+write: 3.78 ns (steps: 98)
104 Size (bytes): 8388608, Stride (bytes): 64, read+write: 7.05 ns (steps: 61)
105 Size (bytes): 8388608, Stride (bytes): 128, read+write: 6.67 ns (steps: 63)
106 Size (bytes): 8388608, Stride (bytes): 256, read+write: 6.27 ns (steps: 67)
107 Size (bytes): 8388608, Stride (bytes): 512, read+write: 3.13 ns (steps: 112)
108 Size (bytes): 8388608, Stride (bytes):1024, read+write: 3.52 ns (steps: 105)
109 Size (bytes): 16777216, Stride (bytes): 4, read+write: 0.96 ns (steps: 218)
110 Size (bytes): 16777216, Stride (bytes): 8, read+write: 1.24 ns (steps: 193)
111 Size (bytes): 16777216, Stride (bytes): 16, read+write: 2.39 ns (steps: 134)
112 Size (bytes): 16777216, Stride (bytes): 32, read+write: 2.41 ns (steps: 137)
113 Size (bytes): 16777216, Stride (bytes): 64, read+write: 10.00 ns (steps: 44)
114 Size (bytes): 16777216, Stride (bytes): 128, read+write: 14.06 ns (steps: 32)
115 Size (bytes): 16777216, Stride (bytes): 256, read+write: 9.78 ns (steps: 45)
116 Size (bytes): 16777216, Stride (bytes): 512, read+write: 7.64 ns (steps: 55)
117 Size (bytes): 16777216, Stride (bytes):1024, read+write: 4.58 ns (steps: 83)
118 Size (bytes): 33554432, Stride (bytes): 4, read+write: 0.95 ns (steps: 220)
119 Size (bytes): 33554432, Stride (bytes): 8, read+write: 1.45 ns (steps: 179)
120 Size (bytes): 33554432, Stride (bytes): 16, read+write: 2.46 ns (steps: 130)
121 Size (bytes): 33554432, Stride (bytes): 32, read+write: 4.88 ns (steps: 80)
122 Size (bytes): 33554432, Stride (bytes): 64, read+write: 8.60 ns (steps: 50)
123 Size (bytes): 33554432, Stride (bytes): 128, read+write: 15.16 ns (steps: 31)
124 Size (bytes): 33554432, Stride (bytes): 256, read+write: 19.17 ns (steps: 24)
125 Size (bytes): 33554432, Stride (bytes): 512, read+write: 17.14 ns (steps: 28)
126 Size (bytes): 33554432, Stride (bytes):1024, read+write: 17.69 ns (steps: 26)
127 Size (bytes): 67108864, Stride (bytes): 4, read+write: 0.90 ns (steps: 222)
128 Size (bytes): 67108864, Stride (bytes): 8, read+write: 1.22 ns (steps: 196)
129 Size (bytes): 67108864, Stride (bytes): 16, read+write: 2.37 ns (steps: 135)
130 Size (bytes): 67108864, Stride (bytes): 32, read+write: 4.81 ns (steps: 81)
131 Size (bytes): 67108864, Stride (bytes): 64, read+write: 10.48 ns (steps: 42)
132 Size (bytes): 67108864, Stride (bytes): 128, read+write: 17.14 ns (steps: 28)
133 Size (bytes): 67108864, Stride (bytes): 256, read+write: 18.46 ns (steps: 26)
134 Size (bytes): 67108864, Stride (bytes): 512, read+write: 16.21 ns (steps: 29)
135 Size (bytes): 67108864, Stride (bytes):1024, read+write: 18.40 ns (steps: 25)
136 Size (bytes):134217728, Stride (bytes): 4, read+write: 0.92 ns (steps: 218)
137 Size (bytes):134217728, Stride (bytes): 8, read+write: 1.42 ns (steps: 183)
138 Size (bytes):134217728, Stride (bytes): 16, read+write: 2.44 ns (steps: 135)
139 Size (bytes):134217728, Stride (bytes): 32, read+write: 5.13 ns (steps: 78)
140 Size (bytes):134217728, Stride (bytes): 64, read+write: 10.23 ns (steps: 43)
141 Size (bytes):134217728, Stride (bytes): 128, read+write: 18.08 ns (steps: 26)
142 Size (bytes):134217728, Stride (bytes): 256, read+write: 18.80 ns (steps: 25)
143 Size (bytes):134217728, Stride (bytes): 512, read+write: 17.78 ns (steps: 27)
144 Size (bytes):134217728, Stride (bytes):1024, read+write: 18.46 ns (steps: 26)

Appendix C - Problem 5 - Q3



Appendix D - Problem 5 - Q5



Appendix E - Problem 5 - Q6

