Cache Performance Analysis

This report is concerned with the presentation and analysis of the performance results of a direct mapped cache architecture which implements a bubble sort algorithm. The results are produced through a bespoke computer simulation program. This program analyses a file containing the main memory access requests made by the CPU throughout the execution of the algorithm for a range of 12 cache modes. Each of the cache modes, describes a potential setup of the cache memory's parameters (cache block size, number of cache blocks, and cache size). The aim is to examine how the different cache parameters affect its performance when implementing the specific algorithm.

The initial parameters relevant to the report are five. The first three are software specific and they are the *cache block size* (i.e. number of 16-bit words per cache block) the *number of cache blocks* (i.e. number of blocks in the cache) and the *cache size*, which is the product of the previous two parameters.
The remaining two parameters are hardware specific and are the $\underline{T_{MM}}$ (main memory access time) and the $\underline{T_{CM}}$ (cache memory access time).

Using the specified data for the cache block size and number of cache blocks for each of the 12 modes under examination, we get the following results.

	NRA *(external reads)*	***NWA*** *(external writes)*	***NCRH*** *(cache read hits)*	***NCRM*** *(cache read misses)*	***NCWH*** *(cache write hits)*	***NCWM*** *(cache write misses)*
Mode 1	866368	768812	3157440	216592	1125532	0
Mode 2	869376	811368	3265360	108672	1125532	0
Mode 3	887488	847920	3318564	55468	1125532	0
Mode 4	924160	884480	3345152	28880	1125532	0
Mode 5	948736	930560	3359208	14824	1125532	0
Mode 6	1102848	1067008	3365416	8616	1125532	0
Mode 7	1145088	1068176	3302464	71568	1125532	0
Mode 8	1132864	1062640	3303228	70804	1125532	0
Mode 9	1083840	1024112	3306292	67740	1125532	0
Mode 10	243072	230320	3358840	15192	1125532	0
Mode 11	1520	0	3373937	95	1125532	0
Mode 12	1520	0	3373937	95	1125532	0

Table 1. Primary metrics for every cache mode

The NCWH and NCWM columns stand out. To see why, it is important to first understand the NCWM column. For the given algorithm implementation, the only section in the given code where reads and writes occur is shown in Figure 1. This code implies that every location to which the CPU writes, has previously been accessed for reading purposes. And it is for this reason there are never any cache-write-misses. Thus, NCWM = 0 always. To address the NCWH column it is important to note this; the total write accesses is the sum of the cache write hits and cache write misses (analogously to total read accesses). The total write-accesses made by this implementation is a specific integer, and the cache write misses is equal to zero. It follows that the number of cache-write-hits is equal to the number of write accesses. It is for this reason that the NCWH column has the same value for each of the modes; the number of write accesses is independent of the cache mode, and completely dependent on the code implementation of the algorithm.

```
if (array[j] > array[j+1]) {

        // Swap the array elements
        temp = array[j];
        array[j] = array[j+1];
        array[j+1] = temp;

}
```

Figure 1. Bubble sort memory accesses from the lab script

The "primary" metrics obtained by the simulation are directly related to actual cache events. However, it is rather difficult to make any useful observations from this data, as it is not presented in a way in which we can make intuitive and useful observations. For this reason, more intentional "secondary" metrics must be used.
Note that for this report, it is assumed that T_{CM} = 8ns and T_{MM} = 80ns.

Cache Hit Ratio	$\boldsymbol{h} = \frac{NCRH + NCWH}{NCRH + NCWH + NCRM + NCWM}$	***Overall Speedup***	$\boldsymbol{S} = \frac{1}{(1-h) + \frac{h}{(T_{MM}/T_{CM})}}$
Average Access Time	$\boldsymbol{T_{Av}} = T_{CM} + (1-h)(T_{MM} * BlockSize)$	***Access Efficiency***	$\boldsymbol{e} = \frac{1}{1 + (1-h)(T_{MM}/T_{CM})}$

Table 2. "Secondary" cache performance metrics used for this report

	Cache Block Size	***Number of Cache Blocks***	***h***	***T_{Av}*** (in nano secs)	***S***	***e***
Mode 1	4	128	0.951864	23.40	6.977266	0.685781
Mode 2	8	64	0.975848	23.46	8.214462	0.809269
Mode 3	16	32	0.987673	23.80	9.001330	0.891461
Mode 4	32	16	0.993582	24.43	9.453890	0.940051
Mode 5	64	8	0.996705	24.87	9.712031	0.968207
Mode 6	128	4	0.998085	27.61	9.830583	0.981247
Mode 7	16	4	0.984094	28.36	8.747758	0.864659
Mode 8	16	8	0.984264	28.14	8.759469	0.865890
Mode 9	16	16	0.984945	27.27	8.806748	0.870867
Mode 10	16	64	0.996624	12.32	9.705093	0.967446
Mode 11	16	128	0.999979	8.03	9.998101	0.999789
Mode 12	16	256	0.999979	8.03	9.998101	0.999789

Table 3. Performance metrics of each cache mode

Going through the memory access file, the range of accessed addresses accessed is 13D8 to 19B3. This corresponds to a total of $19B3_{hex} - 13D8_{hex} = 1499_{dec}$ contiguous main memory addresses that are accessed by the program. It should therefore be of no surprise that modes 11 and 12, whose cache sizes are greater than the main memory size, display almost perfect scores in all secondary metrics.
Note that, $\text{Cache Size} = \text{Cache Block Size} * \text{Number of Cache Blocks}$

For modes 1 to 6, every next mode refers to a cache setup with double the block size and half the number of blocks. Although the size of the cache remains the same, it is easy to deduce that the cache's performance worsens (T_{Av} increases by 4.21ns), even though the hit ratio increases by almost 5%! The explanation is that although there are more cache hits, which means that there are fewer instances of copying blocks back and forth between the cache and the main, each of these instances becomes significantly more costly (timewise) since the number of addresses that need to be copied back and forth increases exponentially.

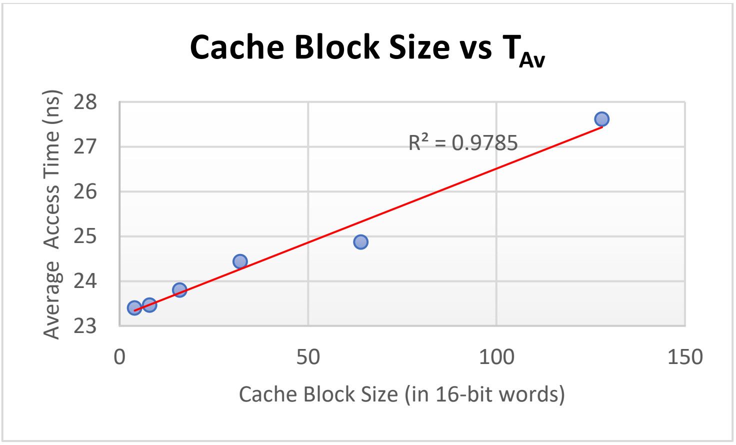

Figure 2. Regression of Access Times for modes 1 to 6

Performing a linear regression on the data of mode 1 to mode 6, we can see that the cache's block size is an almost perfect predictor ($r^2 \approx 1$) of the average access time of the memory system (at least for cache sizes relatively small in comparison to the main memory's size).

Examining modes 7 to 9 we can see that no significant improvement has been made in the average access time (T_{Av} decreases by 1.09 ns) despite a fourfold increase in the cache's size. This does not come as a huge surprise since, as stated previously, the block size is the main predictor of the average memory access time, considering that the cache size remains relatively small.

The performance of mode 10 lies between that of mode 9 and modes 11-12. Although the cache's block size remains small, which could lead to more misses and therefore more back and forth copying of data, this is counteracted by the increased size of the cache. This increased size, which is approximately 2/3 of the main memory's size, means that it is very rare for an external address, which would have to be very non-local to the currently accessed address, to be invoked and replace a cache block. Therefore, there will be few block replacements, which means fewer time penalties, which means reduced average access time. And this is exactly what is observed.

It must be noted that, so far, the discussion concerned the *Full Block Read* method. Therefore, all the metrics above can be thought of as a worst-case scenario, since an implementation of the *Early Restart* or the *Critical Word First* method, could provide increased performance by significantly reducing the penalty associated with cache misses. However, the performance gains of these methods are very specific to the program executed by the CPU as well as to the data used by that program. This makes any generalized modeling of the metrics extremely difficult to implement, all while the results of such a model would possibly not be of greater utility to us than the ones obtained through the simpler model.

Finally, it is important to emphasize the following: the performance metrics of this report are heavily tied to the specific implementation of the specific algorithm. More specifically, the metrics are tied to the spatial-locality of the bubble-sort implementation as well as to the size of the cache in relation to the main/external memory.