Introduction.

This simulation of cache memory is implemented as a data structure and algorithm in the C++ programming language. The data structure is defined as a vector of vectors that represents the cache as a set of blocks (or lines) of memory. The algorithm is defined as a set of instructions that reads each address from a trace file, generates the fields for the corresponding line, and either recognizes if the line is already stored (i.e. a hit), or otherwise stores the line, in the cache implementation. The command line user interface reads arguments of cache size, line size, and lines per set, which are passed to a function that augments the behavior of the algorithm depending on the values of the arguments, and also prints the ratio of hits over ‘addresses read in’ (or ‘cache accesses’).

In order to assess cache performance with respect to the design parameters of size, associativity and replacement policy, the user inputs different values that produce different cache configurations, and outputs to the interface corresponding results which are herein compared and analyzed.

Description of tests.

From a trace file are read 515,683 32-bit addresses which are processed according to cache configuration. Base cache configurations are specified for reproduction across different cache sizes of 128, 256, 1024, 4096, and 32768 bytes. Each base configuration is designated by a series of indicators as follows:

L[4/8] [N/2/4/F]A FIFO/LRU/[blank]

The first indicator (L[4/8]) represents line size: either 4 (L4) or 8 (L8) bytes. The second indicator ([N/2/4/F]A) represents associativity: either direct mapped (NA), 2-way associative (2A), 4-way associative (4A), or fully associative (FA). The third indicator represents replacement policy: either FIFO, LRU, or not applicable (blank).

Cache performance is assessed with respect to cache design parameters of size, associativity, and replacement policy. Parameter values for each cache configuration are specified as follows:

Cache design parameter	L4 NA	L8 NA	L4 2A FIFO	L8 2A FIFO	L4 4A FIFO	L8 4A FIFO	L4 FA FIFO
Size (bytes)	128	128	128	128	128	128	128
	256	256	256	256	256	256	256
	1024	1024	1024	1024	1024	1024	1024
	4096	4096	4096	4096	4096	4096	4096
	32768	32768	32768	32768	32768	32768	32768
Associativity	Direct mapped	Direct mapped	2-way associative	2-way associative	4-way associative	4-way associative	Fully associative
Replacement policy	FIFO = LRU	FIFO = LRU	FIFO	FIFO	FIFO	FIFO	FIFO

Cache design parameter	L8 FA FIFO	L4 2A LRU	L8 2A LRU	L4 4A LRU	L8 4A LRU	L4 FA LRU	L8 FA LRU
Size (bytes)	128	128	128	128	128	128	128
	256	256	256	256	256	256	256
	1024	1024	1024	1024	1024	1024	1024
	4096	4096	4096	4096	4096	4096	4096
	32768	32768	32768	32768	32768	32768	32768
Associativity	Fully associative	2-way associative	2-way associative	4-way associative	4-way associative	Fully associative	Fully associative
Replacement policy	FIFO	LRU	LRU	LRU	LRU	LRU	LRU

Arguments of cache size, line size, and lines per set are passed to a function that defines the parameter values as specified above. The user inputs these arguments on the command line as follows:

User input argument	Data type	L4 NA	L8 NA	L4 2A FIFO	L8 2A FIFO	L4 4A FIFO	L8 4A FIFO	L4 FA FIFO
Cache size (bytes)	size_t	128	128	128	128	128	128	128
	size_t	256	256	256	256	256	256	256
	size_t	1024	1024	1024	1024	1024	1024	1024
	size_t	4096	4096	4096	4096	4096	4096	4096
	size_t	32768	32768	32768	32768	32768	32768	32768
Line size (bytes)	size_t	4	8	4	8	4	8	4
Lines per set	size_t	1	1	2	2	4	4	32
	size_t							64
	size_t							256
	size_t							1024
	size_t							8192
Update on use	bool	0	0	0	0	0	0	0
		1	1					

User input argument	Data type	L8 FA FIFO	L4 2A LRU	L8 2A LRU	L4 4A LRU	L8 4A LRU	L4 FA LRU	L8 FA LRU
Cache size (bytes)	size_t	128	128	128	128	128	128	128
	size_t	256	256	256	256	256	256	256
	size_t	1024	1024	1024	1024	1024	1024	1024
	size_t	4096	4096	4096	4096	4096	4096	4096
	size_t	32768	32768	32768	32768	32768	32768	32768
Line size (bytes)	size_t	8	4	8	4	8	4	8
Lines per set	size_t	16	2	2	4	4	32	16
	size_t	32					64	32
	size_t	128					256	128
	size_t	512					1024	512
	size_t	4096					8192	4096
Update on use	bool	0	1	1	1	1	1	1

The design parameter of associativity is defined by the user arguments of ‘lines per set’ and ‘number of lines'. If lines per set is 1, associativity is direct mapped, and fields for the line number and tag are processed. If lines per set is equal to the number of lines (or cache size divided by line size), associativity is fully associative, and a field for the tag is processed. Otherwise, associativity is n-way associative, with n being equal to lines per set, and fields for the set number and tag are processed.

The design parameter of replacement policy is defined by the user argument of ‘update on use’. If update on use is 0, replacement policy is FIFO (first-in, first-out). If update on use is 1, replacement policy is LRU (least recently used), and hit lines are updated to reflect most recent use.

The design parameter and the user argument of ‘cache size’ are equivalent. On the command line, the user may indicate separate arguments by formatting the input with either a space or a carriage return between arguments. The following is an example of an I/O sequence on the command line:

```
32768
8
4096
1
0.956818
```

← input: cache size
← input: line size
← input: lines per set
← input: update on use
← output: hit rate

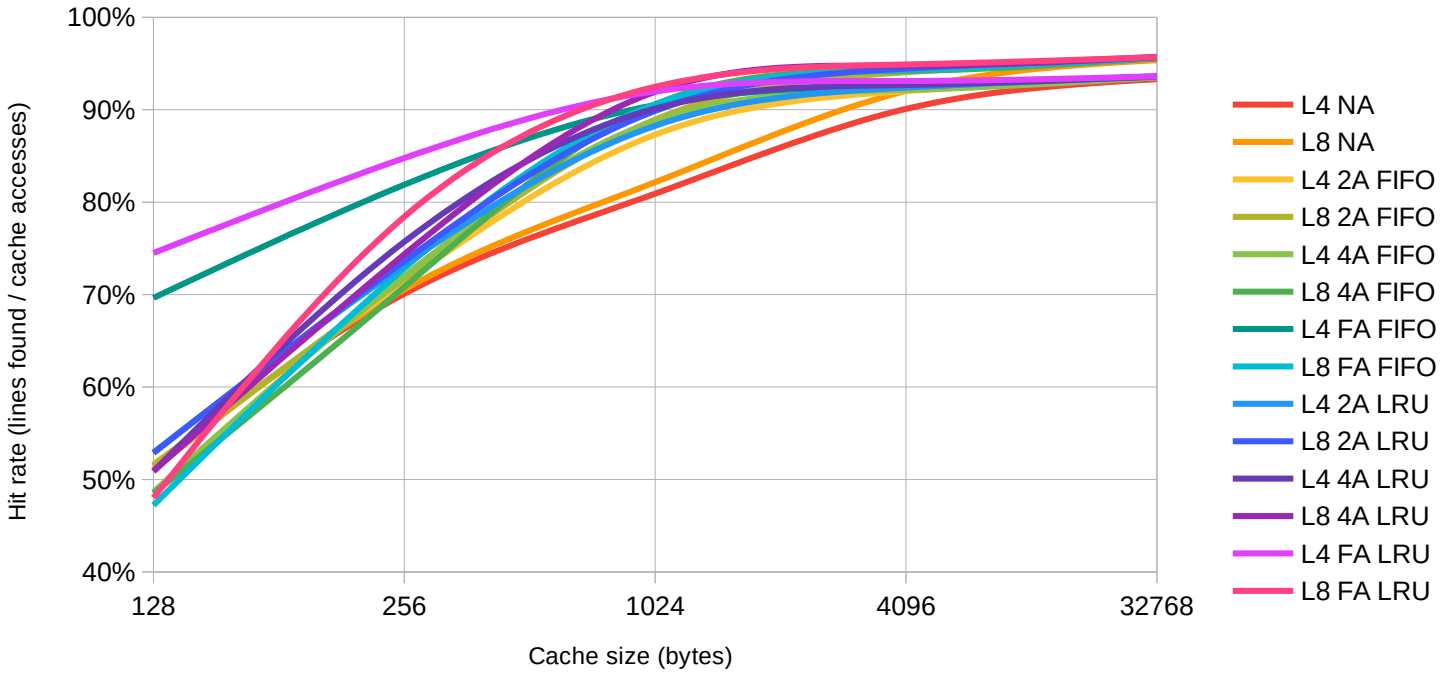
Results.

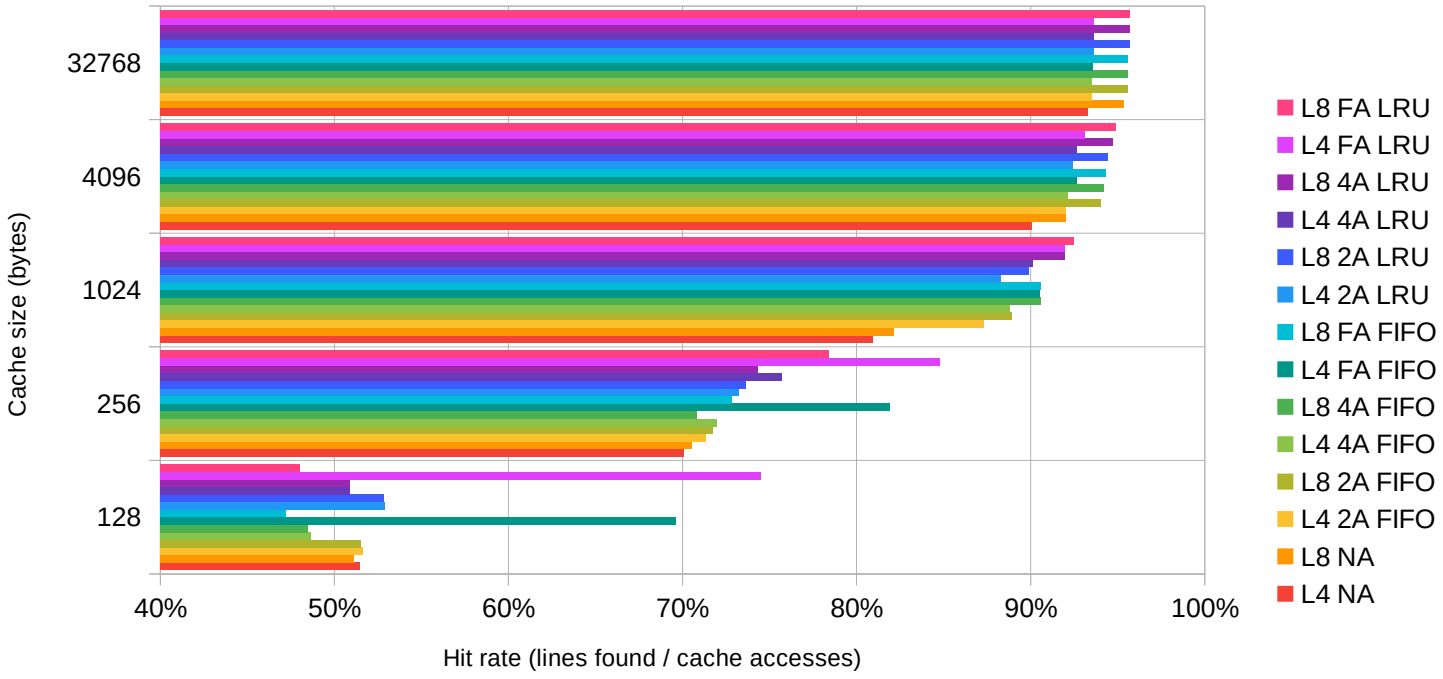
The hit rates for each cache configuration are as follows:

Cache size	L4 NA	L8 NA	L4 2A FIFO	L8 2A FIFO	L4 4A FIFO	L8 4A FIFO	L4 FA FIFO
128	0.514741	0.511023	0.516468	0.515408	0.486559	0.484879	0.696424
256	0.700867	0.705478	0.713692	0.717441	0.719882	0.708226	0.818860
1024	0.809214	0.821606	0.873005	0.889360	0.887834	0.905694	0.905060
4096	0.900864	0.920308	0.920145	0.940588	0.921487	0.941916	0.926676
32768	0.933052	0.953609	0.935051	0.955897	0.935057	0.955884	0.935520

Cache size	L8 FA FIFO	L4 2A LRU	L8 2A LRU	L4 4A LRU	L8 4A LRU	L4 FA LRU	L8 FA LRU
128	0.472075	0.529162	0.528614	0.508958	0.508927	0.745018	0.480231
256	0.728374	0.732177	0.736421	0.757097	0.743488	0.847670	0.784201
1024	0.906016	0.882651	0.898773	0.901228	0.919427	0.919879	0.924845
4096	0.943035	0.924023	0.944530	0.926779	0.947450	0.931053	0.948918
32768	0.955758	0.936060	0.956915	0.936054	0.956894	0.936157	0.956818

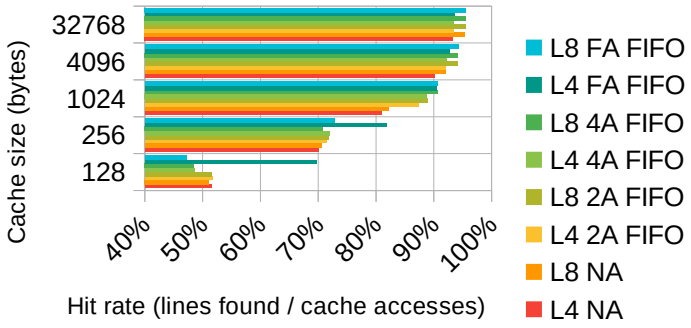
The following charts visualize hit rate (ratio of lines found vs. cache accesses) as a function of cache size (in bytes) for all base configurations:

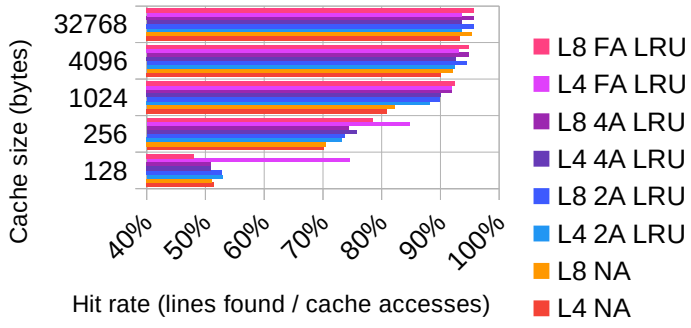




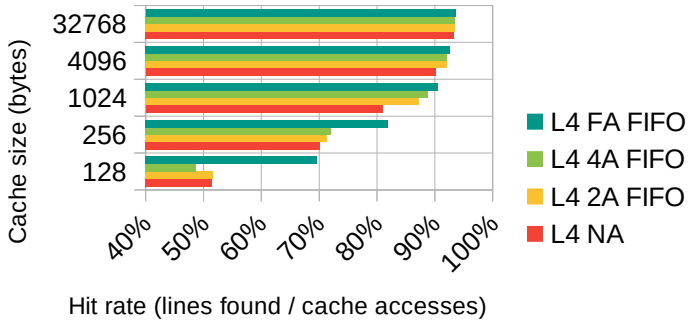
These results show that the hit rate for all base configurations increases when cache size increases. Additional patterns are more visible after controlling for other design parameters.

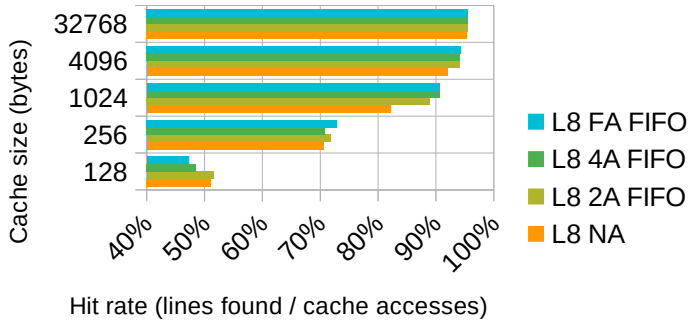
The following charts visualize hit rate as a function of cache size for base configurations with the same replacement method.

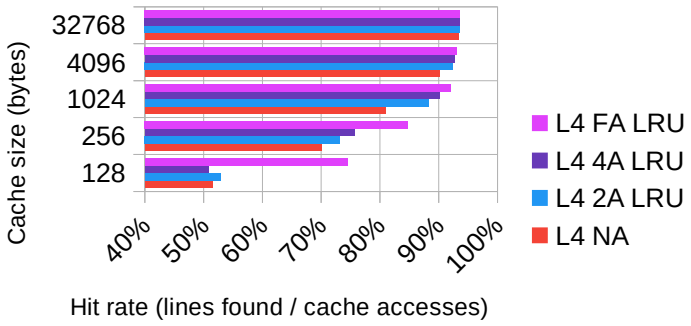


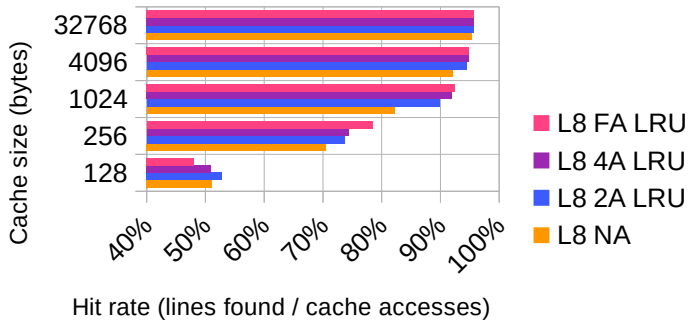


These results show that, as cache size increases, the hit rate for all base configurations more often increases when associativity increases. This pattern becomes even more visible after controlling for line size, as within the following charts.



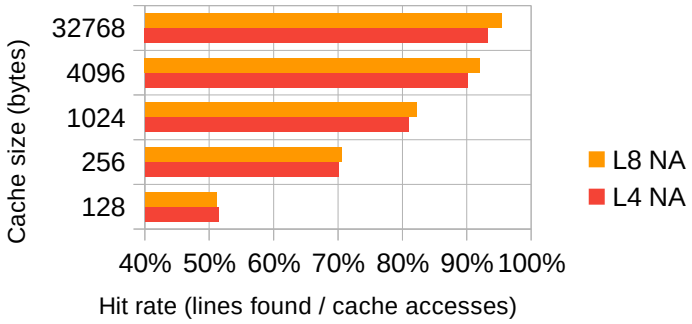


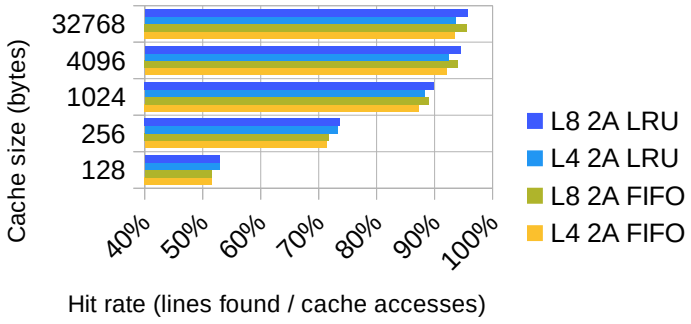


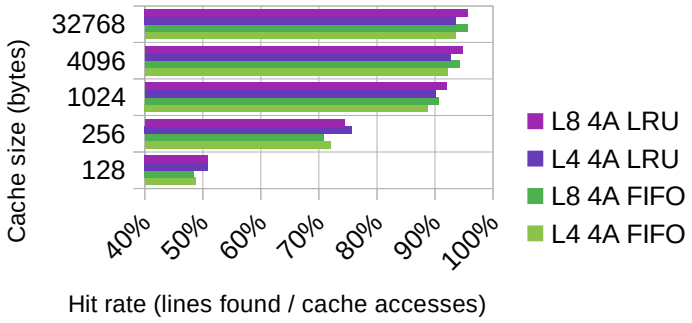


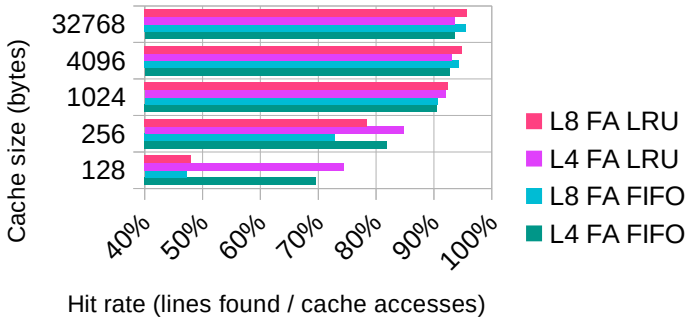
While increased associativity does not always coincide with increased hit rate, increased associativity appears to more consistently coincide with increased hit rate as cache size increases, although the magnitude of the increase in hit rate appears to diminish for much larger cache sizes.

Another parameter that appears to affect the hit rate is the replacement method. The following charts visualize hit rate as a function of cache size for base configurations with the same associativity.









These results show that, when holding line size constant, the hit rate for base configurations with associativity increases when the replacement method switches from FIFO to LRU. For base configurations without associativity (i.e. direct mapped), the hit rate is the same for both replacement methods, FIFO and LRU.

Conclusions.

The patterns that appear in the results suggest possible relationships between cache performance in terms of hit rate and certain cache design parameters: size, associativity, and replacement policy. One parameter (associativity) produced outliers for an implied relationship with hit rate while others did not. In this case, outliers were produced for the smaller cache sizes (128 and 256 bytes), where the impact of spacial limitations may have been so pronounced as to outweigh the impact of associativity.

A parameter that did not produce outliers for an implied relationship with hit rate is cache size. For all base configurations, when holding other parameters as well as line size constant, a larger cache size resulted in an increased hit rate. This result makes theoretical sense in that a cache that stores more lines will be more likely to preserve rather than replace that line when the same line is later accessed, resulting in a hit rather than miss. Therefore, one may conclude that, under normal conditions, cache performance improves for a larger cache size, holding all other considerations constant.

For configurations with a larger cache size (1024 bytes and above), the parameter of associativity did not produce outliers for an implied relationship with hit rate. In all such cases, increased associativity resulted in an increased hit rate. Similar to the result for cache size parameter, this result makes theoretical sense in that a set that stores more lines (increased lines per set for increased associativity) will be more likely to preserve rather than replace that line when the same line is later accessed, resulting in a hit rather than miss. Therefore, one may conclude that, under normal conditions, cache performance improves for increased associativity where cache size exceeds 1024 bytes, holding all other considerations constant.

Another parameter that did not produce outliers for an implied relationship with hit rate is replacement policy. For all base configurations, when holding all other parameters as well as line size constant, switching the replacement policy from FIFO to LRU resulted in an increased hit rate. This result makes theoretical sense in that a line that was recently accessed (or used), which may be more likely to be accessed in the future, is moved to the back of the queue of lines to be replaced, so that future accesses are more certain to result in hits rather than misses. Therefore, one may conclude that, under normal conditions, cache performance improves when switching the replacement policy from FIFO to LRU, holding all other considerations constant.