Best Cache Analysis

Sarthak Panda(2022CS51217)

Comparison Between Strategy

Certainly, the caches that give higher hit rates and lesser number of cycles will be favoured.

- *Hypothesis for Cycle Optimization:*

(1) **Write Allocate & Write Back Performs better than Write Allocate & Write Through**

Reasoning: In write allocate and write through there are usually very long delay when there is store miss. Because of write allocate we bring the data from the main memory to cache but after updating it in cache we need to main memory at that moment because of write through strategy. So, bringing the data and that time only sending back is not optimal choice, rather in write back the writing back to main memory is delayed which is acts as an optimization here.

Testing & Experimentation:

Tracefile_1's Cache Parameters: (a) 16 4 16 write-allocate write-through lru
(b) 16 4 16 write-allocate write-back lru

Tracefile_2's Cache Parameters: (a) 16 2 4 write-allocate write-through lru
(b) 16 2 4 write-allocate write-back lru

Tracefile_3's Cache Parameters: (a) 32 8 16 write-allocate write-through fifo
(b) 32 8 16 write-allocate write-back fifo

	Total Load	Total Store	Load Hit	Load Miss	Store Hit	Store Miss	Total Cycles
Tracefile_1(a)	484	516	270	214	255	261	397875
Tracefile_1(b)	484	516	270	214	255	261	304156
Tracefile_2(a)	485	515	175	310	184	331	117241
Tracefile_2(b)	485	515	175	310	184	331	102101
Tracefile_3(a)	516	484	402	114	386	98	279612
Tracefile_3(b)	516	484	402	114	386	98	95636

(2) **As size of the Cache increases, the number of cycles is likely to go down**

Reasoning: As the size of cache increases there are two events that we should consider. The first being as the cache size increases then we have more space to allocate recent data and hence possibility of conflict miss goes down hence the access to memory are likely to go down, hence the total number of cycles. Second event is that the single clock cycle itself becomes longer when cache crosses a threshold because larger memories are usually slower.so we should carefully trade-off.

Testing & Experimentation:

Tracefile_1's Cache Parameters: (a) 2 2 4 write-allocate write-through fifo

(b) 4 2 4 write-allocate write-through fifo

Tracefile_2's Cache Parameters: (a) 4 16 8 write-allocate write-back lru

(b) 8 32 16 write-allocate write-back lru

Tracefile_3's Cache Parameters: (a) 8 2 4 no-write-allocate write-through fifo

(b) 16 2 4 no-write-allocate write-through fifo

	Total Load	Total Store	Load Hit	Load Miss	Store Hit	Store Miss	Total Cycles
Tracefile_1(a)	534	466	40	494	38	428	**140722**
Tracefile_1(b)	534	466	82	452	59	407	**134359**
Tracefile_2(a)	499	501	254	245	245	256	**156775**
Tracefile_2(b)	499	501	403	96	393	108	**82804**
Tracefile_3(a)	513	487	107	406	108	379	**90327**
Tracefile_3(b)	513	487	157	356	167	320	**85336**

- *Hypothesis for Optimal Hit Rate:*

(1) **As the number of ways increases beyond 4/8 then there is no further increment hit ratio**

Reasoning: The number of ways increases the associativity level increases so hit rate increases but we result a slower cache. But we will see that beyond increases 4 or 8 the Hit ratio saturates.-off.

Testing & Experimentation:

Tracefile_1's Cache Parameters: (a) 64 2 8 write-allocate write-back lru

(b) 64 4 8 write-allocate write-back lru

(c) 64 8 8 write-allocate write-back lru

(d) 64 16 8 write-allocate write-back lru

(e) 64 32 8 write-allocate write-back lru

(f) 64 64 8 write-allocate write-back lru

	Total Load	Total Store	Load Hit	Load Miss	Store Hit	Store Miss	Hit Ratio
Tracefile_1(a)	507	493	276	231	290	203	**0.566**
Tracefile_1(b)	507	493	356	151	368	125	**0.724**
Tracefile_1(c)	507	493	377	130	403	90	**0.780**
Tracefile_1(d)	507	493	377	130	403	90	**0.780**
Tracefile_1(e)	507	493	377	130	403	90	**0.780**
Tracefile_1(f)	507	493	377	130	403	90	**0.780**

(2) **Write Allocate performs better than No Write Allocate in terms of Hit rate**

Reasoning: Write allocate though takes a few cycles more than no write allocate but more importantly it stores the updated data in cache unlike no write allocate. Updating cache helps for further future access which aligns with fact/proposition of temporal/time locality of access of data in a cache. But no write allocate suffers future access so write allocate is going to have higher hit rate.

Testing & Experimentation:

Tracefile_1's Cache Parameters: (a) 8 2 16 write-allocate write-through fifo

(b) 8 2 16 no-write-allocate write-through fifo

Tracefile_2's Cache Parameters: (a) 32 2 16 write-allocate write-through fifo

(b) 32 2 16 no-write-allocate write-through fifo

Tracefile_3's Cache Parameters: (a) 32 8 4 write-allocate write-through fifo

(b) 32 8 4 no-write-allocate write-through fifo

	Total Load	Total Store	Load Hit	Load Miss	Store Hit	Store Miss	Hit Ratio
Tracefile_1(a)	513	487	112	401	99	388	**0.211**
Tracefile_1(b)	513	487	108	405	76	411	**0.184**
Tracefile_2(a)	484	516	233	251	241	275	**0.474**
Tracefile_2(b)	484	516	217	267	211	305	**0.428**
Tracefile_3(a)	478	522	395	83	414	108	**0.809**
Tracefile_3(b)	478	522	324	154	339	183	**0.663**

(3) **LRU in general performs better than FIFO Eviction Strategy**

Reasoning: LRU is based on temporal/time access locality while FIFO is based on main memory access history. LRU represents general programs written for e.g. Global Variables may be declared very early in program (hence accessed from memory very early) but can be called multiple times in later code. In such cases certainly LRU outperforms FIFO.

Testing & Experimentation:

Tracefile_1's Cache Parameters: (a) 256 2 32 write-allocate write-back lru

(b) 256 2 32 write-allocate write-back fifo

Tracefile_2's Cache Parameters: (a) 16 8 32 no-write-allocate write-through lru

(b) 16 8 32 no-write-allocate write-through fifo

Tracefile_3's Cache Parameters: (a) 2 16 4 write-allocate write-through lru

(b) 2 16 4 write-allocate write-through fifo

	Total Load	Total Store	Load Hit	Load Miss	Store Hit	Store Miss	Hit Ratio
Tracefile_1(a)	495	505	392	103	399	106	**0.791**
Tracefile_1(b)	495	505	393	102	397	108	**0.790**
Tracefile_2(a)	460	540	279	181	338	202	**0.617**
Tracefile_2(b)	460	540	272	188	341	199	**0.613**
Tracefile_3(a)	534	466	188	346	167	299	**0.355**
Tracefile_3(b)	534	466	185	349	160	306	**0.345**

Conclusion:

From all the discussions that we made above here is sequential application of results:

1) Write allocate is preferred over No-Write allocate

2) Since write allocate write policy is used so for faster system write back along with it.

3)LRU has slight edge over FIFO in terms of hit ratio

4)4-way set associative caches are optimal

So, in terms of Fastest and Highest hit ratio cache is a ***4-Way set associative cache using LRU as replacement policy, Write Back as write hit policy and write allocate as write miss policy***.

(**Practical Implementation Considerations:** In case one also wants trade-off with lower hardware (hence lower cost of production) one might consider FIFO over LRU, FIFO can be easily implemented on hardware system by setting up a counter. In case exception in cache halts in between then all the progress till the point of exception is stored in main memory in case of write through policy but is lost in write back. In practical if considered system require such safety, we might need to switch to slower yet safe write through)