On the Design and Implementation of Low-power MESI Protocol

Tri Caohuu, Xiaochen Zhang

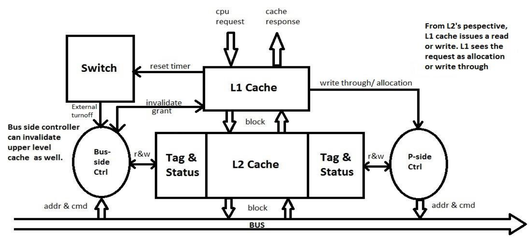
Department of Electrical Engineering, San Jose State university, San Jose, California 95192.

Introduction

This project investigates the design and implementation of parallel computer architecture using techniques to save the leakage current in the multi-core system. The method applied in this project enhances the MESI cache coherence protocol by modifying its state transition to allow cache lines to be turned off with invalidation requests and external signals. The system simulates the function of a two-level cache, in which L1 cache is a direct mapped write through cache and L2 is a two-way associated write back cache. Functional and performance tests on this cache system show that the power consumption of L2 cache is substantially reduced at the cost of slightly larger size of the design and more compulsory cache misses.

Methodology

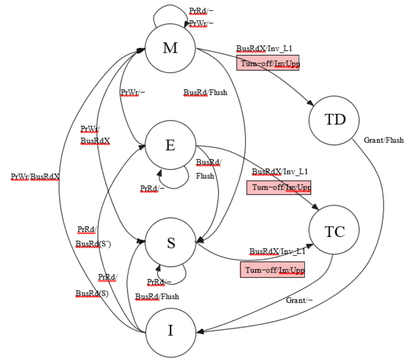
A two-level private cache is implemented. L1 cache is a direct-mapped write through cache with 64 entries of 16-byte cache line. No dirty bit is needed in this cache structure, because the cache writes through the data to the low level memory on every write hit to make sure value in L2 cache block is always up-to-date. The L2 cache is a two-way associated write back cache with 1k entries of the same size block and Least Recently Used replacement policy. Two ways of turning off cache blocks have been implemented (BusRdx turning off and external turning off).



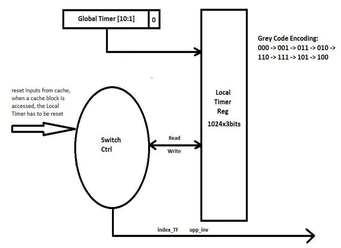
MESI (Illinois) protocol is the most popular protocol for multiprocessor system to maintain cache coherence. The classic MESI protocol works as a state machine with four transition states, Modified state, Exclusive state, Shared state, and Invalid state. In this modified version of MESI protocol, two more transient states are added into state transfer diagram. The state Transient Dirty means the data of the cache block temporally in this state is different from value in the memory. If a cache block is currently in Modified state, and a BusRdx request or an external turn off signal is seen by the bus snooping unit, then the state of this cache block will be altered to the Transient Dirty state by the bus-side cache controller, and the state machine will generate an invalidation signal as an output

Methodology

to disable the corresponding block in L1 cache if this cache block exists. Another state called Transient Clean means the data stored in this cache line is the same as the copy in main memory. If a cache block is in Exclusive state or Shared state, and a BusRdx request or an external turn off signal intends to invalidate this block, the state will go to the Transient Clean state and propagate the invalidation to L1 cache in a similar way. The only difference between Transient Dirty and Transient Clean is that when the block is invalidated by grant signal, the block in Transient Dirty state will be flushed to the bus, but the block in Transient Clean state will not.



In order to achieve power saving, a switching unit is added to this system to enable external turn off. This switching unit is a RAM with a timer that can trace the access history of each cache block. There is an 11-bit global timer that increment at the positive edge of the clock, and a local timers RAM with 1024 entries (one for each L2 cache block). Every local timer entry is a 3-bit register that will be incremented in a Grey Code fashion when the global timer is pointing to it. Every local timer entry will be reset to its initial value by any hit in L1 cache. Whenever the local timer change its value to 100, it will generate a external



signal to turn off a L2 cache block according to the global timer. This mechanism will work like the BusRdx request to invalidate L2 cache block and the corresponding L1 cache block if it's necessary.

Results Analysis

The testbench is constructed by using SystemVerilog random function to enhance the flexibility and coverage of test cases. Every variable is randomized with constrains. The private and shared requests rate was set by randomizing the time interval between two consecutive requests. Write request percentage is set by comparing random number with a threshold. Cache reference address is constrained in a scope based on previous cache reference to exploit spatial locality.

```
addr_CPU = $urandom_range (0, 8192);
repeat (1024) begin
   addr_CPU = $urandom_range (addr_CPU - X, addr_CPU + X);
end
```

The first test case is the random address test. In this test case, 90 percent of all requests are private reads or writes, 10 percent of them are shared reads and writes. The read/write ratio is 7:3. The processor sends a private request every 5.5 clock cycles averagely, and bus sends a shared request every 47 clock cycles averagely.

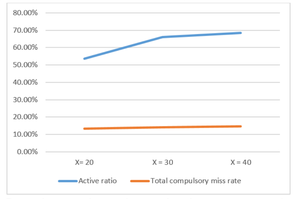
Metrics	Case 1	Case 2
Private requests	8192	16384
Shared requests	973	1912
Execution cycles	45020	90306
Cycles/ Private Request	5.50	5.51
Cycles/Shared Request	46.27	47.23
L2 on cycles	35281558	77253321
L2 total cycles	46100480	92473344
Active ratio	76.53%	83.54%
Extra compulsory miss	215(1239)	758(1782)
Extra compulsory miss rate	2.6%	4.6%

The active ratio is calculated by the following formula:

$$\text{The Active Ratio} = \frac{\sum (\textit{the Active Time of each cache entry})}{\#\text{ of Cache Entries} * \text{Execution Cycles}}$$

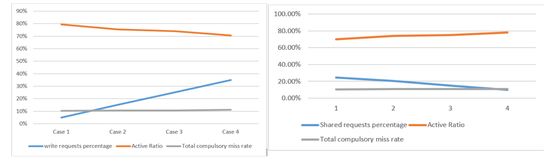
The second test case exploits spatial locality. In this test case the cache performance is determined by three variables. The first one is the locality factor (the scope of consecutive cache reference address), another one is the percentage of shared requests, and the last one is the percentage of write requests.

Metrics	Case 1	Case 2	Case 3
Locality factor	X= 20	X = 30	X = 40
Private requests	8192	8192	8192
Shared requests	949	947	950
Execution cycles	45198	44932	45017
L2 on cycles	24815487	30358464	31547066
Active ratio	53.62%	65.98%	68.44%
Total compulsory miss	1089	1152	1203
Total compulsory miss rate	13.3%	14.1%	14.7%



If the locality factor X is set as a greater number, requests will be able to cover more cache entries. The off time will decrease if X increases. On the other hand, the compulsory misses will increase if the X increases.

If the more shared requests have been issued from the bus, more shared write will occur with no changes on the read/write ratio. In this case, more BusRdx requests will turn off more cache lines and the compulsory misses of L2 cache will increase. If more write requests have been issued, more BusRdx requests will be snooped from the bus, and more L2 cache blocks will be turned off and higher compulsory miss rate is expected.



Conclusion

Power analysis for random memory reference case shows 16 percent of L2 cache power consumption can be saved at the cost of 2.6 to 4.6 percent more compulsory misses. The performance test case that exploits spatial locality suggests 20 to 30 percent of power can be saved. The performance loss caused by extra misses will be less than 5 percent. Besides, synthesis result suggests 1.6 percent of more logic gates and 0.7 percent of more power are needed to complete the functions of the two-level coherent cache system with modified MESI protocol. The overall power saving is still considerable.

Key References

[1] Culler, D. E., J. P. Singh, A. Gupta, "Parallel Computer Architecture: A Hardware/Software Approach," Morgan Kaufmann Publisher Inc., 1999

[2] M. Monchiero, R. Canal, A. Gonzalez, "Using Coherence Information and Decay Techniques to Optimize L2 Cache Leakage in CMPs," International Conference on Parallel Processing, ICCP' 09, pp.1-8, 2009

[3] S. Kaxiras, Z. Hu, M. Martonosi, "Cache Decay: Exploiting Generational Behavior to Reduce Cache Leakage Power,"Proceedings of 28th Annual International Symposium on Computer architecture. New York, NY, USA: ACM Press, Pages 240-251, 0-7695-1162-7/01/2001

Acknowledgements

I wish to thank Professor Tri Caohuu for helping me with learning Parallel Computer System and the design of coherent cache. I wish to thank Professor Morris Jones for helping me with the design environment and tracer driven testing.