1 Introductiom

For this study we chose to examine several system organizations,including
(a) a baseline system with a single cache
(b) a system with a two-level cache, and
(c) a two-level system with separate local instruction and data caches.

We refer to the processor-local cache as the L1 cache and the second-level cache as the L2 cache. At a high level, the components of this system operates in the following way;
l. The instruction fetch unit contains an instruction register and a prefetch buffer into which the next instruction is read while the current instruction is executing. When the prefetch buffer is empty, the instruction fetch unit sends an address to the cache and waits for the instruction to return. This provides a simple model of instruction prefetch.
2. The instruction execution unit fetches operands and stores results. The instruction execution unit waits until an instruction is ready in the instruction register, begins fetching operands from the cache, executes the instruction,and then stores results.
3. The L1 cache receives addresses from the prefetch and returns instructions either from the cache or from the next level of the memory hierarchy. The cache also receives addresses from the execution unit and reads or writes operands, again from the cache or from the next level of the hierarchy. The handling of writes varies with different write algorithms. If separate Ll instruction and data caches are present, they respond to the instruction fetch and instruction execution units, respectively.
4. The L2 cache receives addresses from the Ll cache (or caches) and reads or writes operands from its storage or from the primary memory system. The handling of writes varies with different write algorithms.
5. The bus is a half-duplex datapath connecting the caches to the memory system. Devices on the bus must arbitrate for bus ownership before commands or data can be sent.
6. The primary memory consists of a number of interleaved memories. Simulation parameters include the interleaving factor, access time, and cycle time of main memory.

2 Methodology

Implemented a two-level (L1 and L2) cache simulator in C++ with round robin eviction policy The cache simulator takes several parameters describing the cache (block size, associativity, etc) along with a memory access trace file for an input program.

Cache Design

[1]Read Miss: (i)on a read miss, the cache issues a read request for the data

from the lower level of the cache. Once the data is returned, it is placed in an empty way, if one exists, or data in one of the ways is evicted to create room for the new data.

(ii) The ways of the cache are numbered from 0,1,2..W-1 for a W-way cache. If an empty way exists, data is placed in lowest numbered empty way.

(iii) Eviction is performed based on a round-robin policy. Each way has a counter that is initialized to 0, counts up to W-1 and loops back to zero. The current value of the counter indicates the way from wich data is to be evicted. The counter is incremented by 1 after an eviction.

[2]Write Hit: both the L1 and L2 caches are write-back caches.

[3]Write Miss: both the L1 and L2 caches are write no-allocate caches. On a write miss, the write request is forwarded to the lower level of the cache.

[4]Non-Inclusive: the L1 and L2 caches are non-inclusive.

Configuration File (config.txt):

The parameters of the L1 and L2 caches are specified in a configuration file. The format of the configuration file is as follows.

[1]Block size: Specifies the block size for the cache in bytes. This should always be a non-negative power of 2 (i.e., 1, 2, 4, 8, etc).

[2]Associativity: Specifies the associativity of the cache. A value of "1" implies a direct-mapped cache, while a "0" value implies fully-associative. Should always be a non-negative power of 2.

[3]Cache size: Specifies the total size of the cache data array in KB.

An example config file is: a 16KB direct-mapped L1 cache with 8 byte blocks, and a 32KB 4-way set associative L2 cache with 16 byte blocks.

Trace File (trace.txt):

Simulator needs to take as input a trace file that is used to compute the output statistics. The trace file specifies all the data memory accesses that occur in the sample program. Each line in the trace file specifies a new memory reference. Each line in the trace cache therefore have the following two fields:

[1]Access Type: A single character indicating whether the access is a read ('R') or a write ('W').

[2]Address: A 32-bit integer (in unsigned hexidecimal format) specifying the

memory address that is being accessed. Fields on the same line are separated by a single space.

The code reads the trace file one line at a time in order. After each access, code emulates the impact of the access on the cache hierarchy.