INDIAN INSTITUTE OF TECHNOLOGY PALAKKAD

CS5119- Advanced Computer Architecture Lab

Subham Ball(152202017)

December 15, 2022

LAB 6

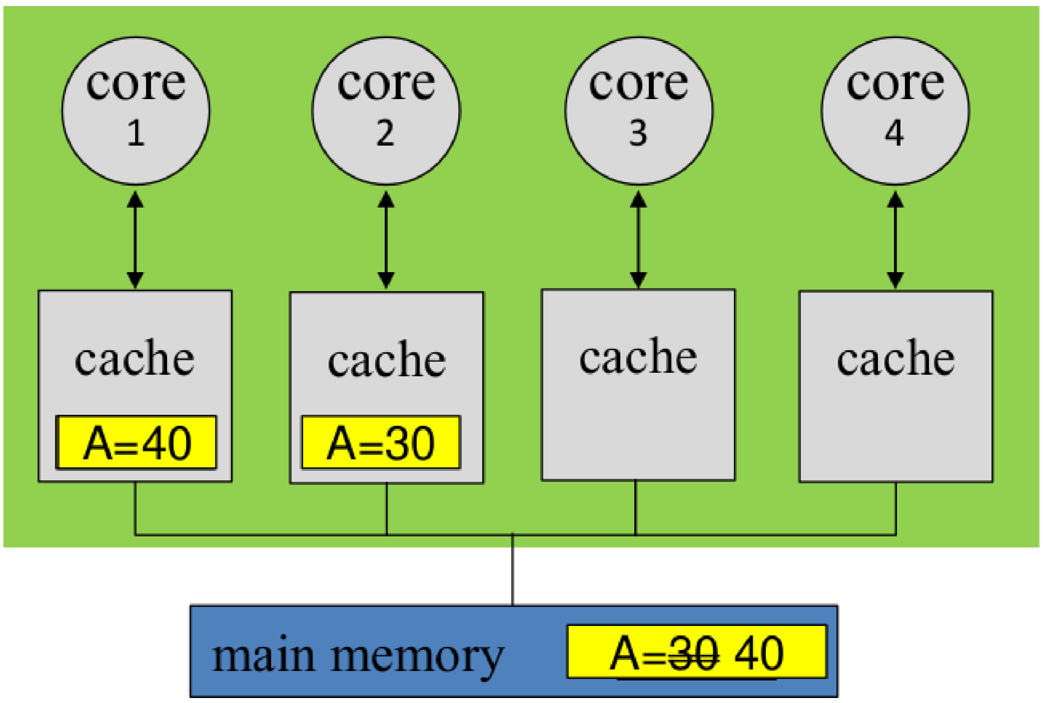
1 INTRODUCTION

In this experiment, we will explore all cache memory system features and execute memory-intensive programmes on both in-order and out-of-order CPUs using different cache sizes (128KB, 256KB, 512KB, and 1024KB).

2 CACHE COHERENCE

In a multiprocessor system, data inconsistency may occur among adjacent levels or within the same level of the memory hierarchy. In a shared memory multiprocessor with a separate cache memory for each processor, it is possible to have many copies of any one instruction operand: one copy in the main memory and one in each cache memory. When one copy of an operand is changed, the other copies of the operand must be changed also. There are many Cache Coherence Protocols in the multiprocessor system.

1. MSI protocol (Modified, Shared, Invalid)
2. MOSI protocol (Modified, Owned, Shared, Invalid)
3. MESI protocol (Modified, Exclusive, Shared, Invalid)
4. MOESI protocol (Modified, Owned, Exclusive, Shared, Invalid)



1. Modified(M): Value in cache is dirty, need to update on main memory.
2. exclusive(E): value present in cache is clean, which means the same value as main memory.
3. shared(s): cache holds most recent data copy and it is shared all cache and memory.
4. owned(O): it means the current cache hold the block, and the owner of the block. that is having all rights on that particular block.
5. Invalid(I): state that the current cache block is invalid, needs to fetch from another cache or memory.

3 Cache Coherence mechanism

1. snooping based protocol
   a) write update write through
   b) write update write back
   c) write invalidate write through
   d) write invalidate write back
2. Directory based protocol

4 Creating SimObjects in the memory system

In this chapter, we'll build a basic memory object that resides between the CPU and the memory bus.

1. Port interface : There are two types of ports in gem5:
   a) master ports
   b) slave ports

   Whenever you implement a memory object, you will implement at least one of these types of ports. To do this, you create a new class that inherits from either MasterPort or SlavePort for master and slave ports, respectively. Master ports send requests (and receive response), and slave ports receive requests (and send responses).

2. Packets : In gem5, Packets are sent across ports. A Packet is made up of a MemReq which is the memory request object. The MemReq holds information about the original request that initiated the packet such as the requestor, the address, and the type of request (read, write, etc.).

3. Connections : These ports are linked using the assignment operator. Vector ports are used by objects with an infinite number of ports, such as buses. After all objects are instantiated, the python code connects memory ports in C++.

4. Request : The initial request sent by a CPU or I/O device is encapsulated in a request object. Virtual address is one of the request object attributes. If the request was sent directly to a physical address, this field may be invalid. Physical address, Data size, Time the request was generated, The ID of the CPU/thread that caused this request.

5. Access Type: there are three type
   a) Timing : When a timing request is successfully sent in the future, the device that sent the request will either get the answer or a NACK if the request could not be completed.
   b) Atomic : Atomic accesses are a faster than detailed access. They are used for fast forwarding and warming up caches.
   c) Functional: Functional accesses are used to load binaries, examine and change variables in the simulated system, and connect a remote debugger to the simulator.

6. Timing Flow control : Timing requests imitate a genuine memory system, and flow control is required to handle them because they are not instantaneous. When a timing packet is sent using sendTiming(), it may or may not be acknowledged, as indicated by the return value true or false. If false, the object should not send any further packets until it gets a recvRetry() function. The true/false return is designed for local flow control, whereas nacking is intended for global flow control. A answer cannot be nacked in any situation.

7. Memory intensive application : In my perspective, the ML2 BW is the best since it contains more load and store operations that demand memory every time an instruction or code is executed.

   a) in-order Cpu :

Parameter	CacheSize((**128KB**))	CacheSize((**256KB**))	CacheSize((**512KB**))	CacheSize((**1024KB**))
simSeconds	0.005014	0.004672	0.003921	0.002098
MissRate	0.202276	0.185380	0.149196	0.062551

   As we note from the table the performance is growing by raising the cache size. Because of the larger cache sizes, there is no need to go to memory every time to get data.

   b) Out-of-order CPU :

Parameter	CacheSize((**128KB**))	CacheSize((**256KB**))	CacheSize((**512KB**))	CacheSize((**1024KB**))
simSeconds	0.001287	0.001168	0.000914	0.000335
MissRate	0.788342	0.755005	0.682230	0.498712

   Out-of-order CPU performance rises when compared to in-order CPU performance since there are so many load and store instructions. When we execute in-order, stalls arise due to data dependencies, but stalls are removed in out-of-order, therefore cpu cycles also reduce while boosting performance. If we look at the trend of out-of-order CPU execution, we can observe that performance is gradually improving as cache sizes are increased.