COMPUTER ORGANIZATION AND ARCHITECTURE (COA)

EET 2211

4TH SEMESTER – CSE & CSIT

CHAPTER 7, LECTURE 28

TOPICS TO BE COVERED

1. External Devices
2. I/O Modules
3. Programmed I/O
4. Interrupt-Driven I/O
5. Direct Memory Access
6. Direct Cache Access

LEARNING OBJECTIVES

- Explain the use of I/O modules as part of a computer organization.
- Understand programmed I/O and discuss its relative merits.
- Present an overview of the operation of direct memory access.
- Present an overview of direct cache access.

DIRECT CACHE ACCESS (DCA)

WHY DCA?

- DMA has proved an effective means of enhancing performance of I/O with peripheral devices and network I/O traffic. However, for the dramatic increases in data rates for network I/O, DMA is not able to scale to meet the increased demand.
- This demand is coming primarily from the widespread deployment of 10-Gbps and 100-Gbps Ethernet switches to handle massive amounts of data transfer to and from database servers and other high-performance systems
- A secondary but increasingly important source of traffic comes from Wi-Fi in the gigabit range.
- Network Wi-Fi devices that handle 3.2 Gbps and 6.76 Gbps are becoming widely available and producing demand on enterprise systems.

- In this lecture, we will show how enabling the I/O function to have direct access to the cache can enhance performance, a technique known as **direct cache access (DCA).**

CONTENTS

1. We describe the way in which contemporary multicore systems use **On-chip shared cache** to enhance DMA performance. This approach involves enabling the DMA function to have **direct access to the last level cache.**

2. Next we examine **cache-related performance issues** that manifest when high-speed network traffic is processed.

3. From there, we look at several **different strategies for DCA** that are designed to enhance network protocol processing performance.

4. Finally, we describe a DCA approach implemented by Intel, referred to as **Direct Data I/O**.

DMA USING SHARED LAST-LEVEL CACHE

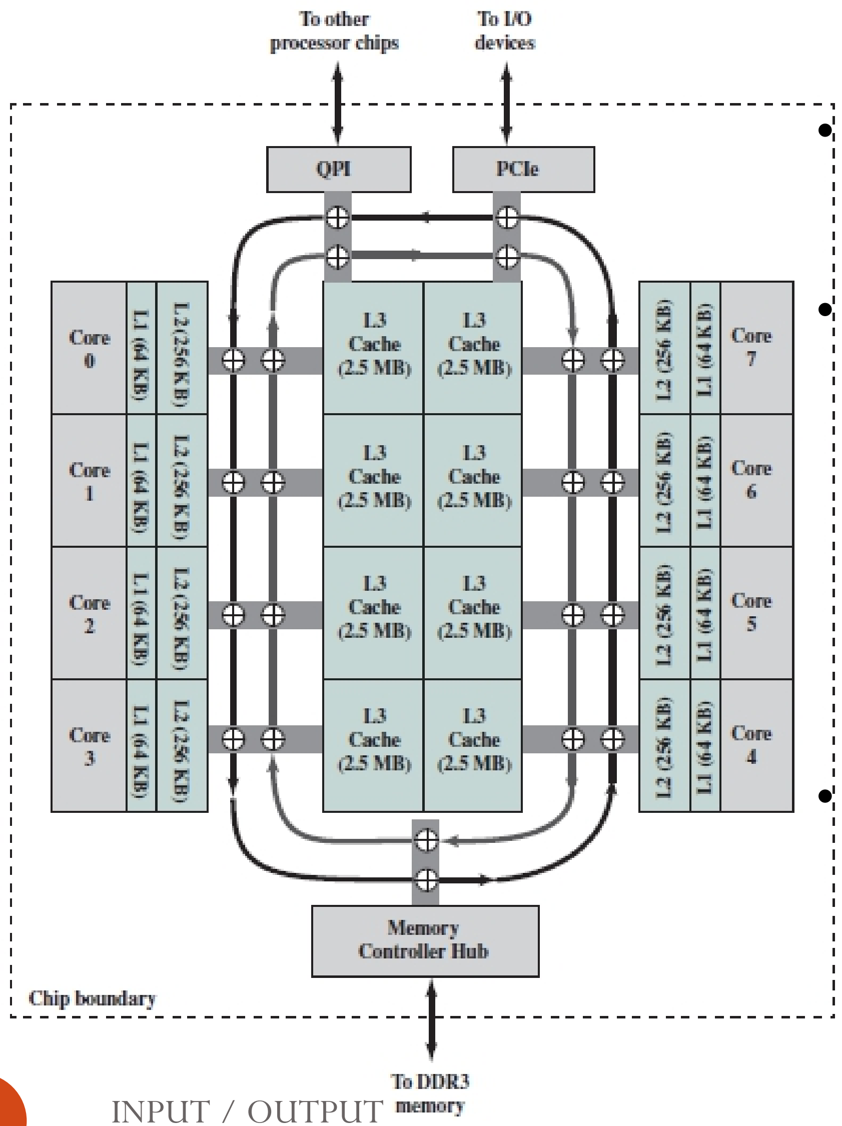
- Contemporary multicore systems include both cache dedicated to each core and an additional level of shared cache, either L2 or L3.
- With the increasing size of available last-level cache, system designers have enhanced the DMA function so that the DMA controller has access to the shared cache in a manner similar to the cores.
- To clarify the interaction of DMA and cache, it will be useful to first describe a specific system architecture.
- For this purpose, the following is an overview of the Intel Xeon system.

XEON MULTICORE PROCESSOR

- Intel Xeon is Intel's high-end, high-performance processor family, used in servers, high-performance workstations, and super computers.

- Many of the members of the Xeon family use a **ring interconnect** system

Xeon E5-2600/4600 Chip Architecture

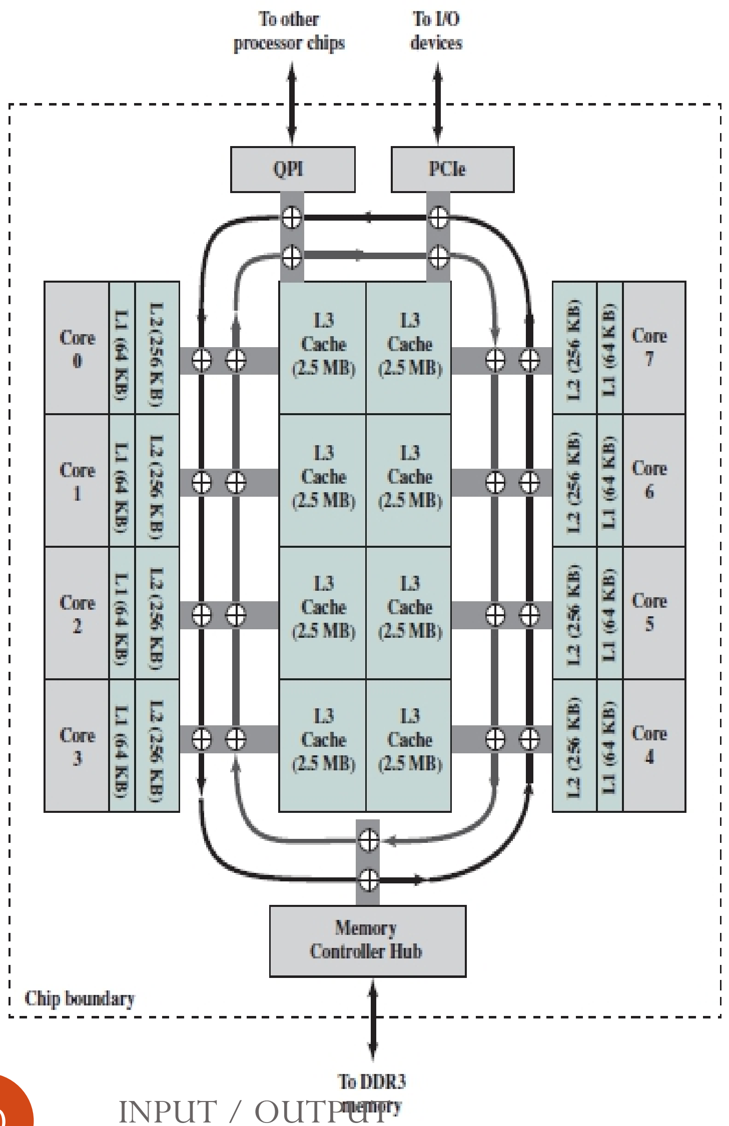


- The E5-2600/4600 can be configured with up to **eight cores** on a single chip.

- Each core has dedicated L1 and L2 caches. There is a **shared L3** cache of up to 20 MB. The L3 cache is divided into slices, one associated with each core although each core can address the entire cache. Further, each slice has its own cache pipeline, so that requests can be sent in parallel to the slices.

- The bidirectional high- speed ring interconnect links cores, last- level cache, PCIe, and **integrated memory controller (IMC).**

Working Principle



In essence, the ring operates as follows:

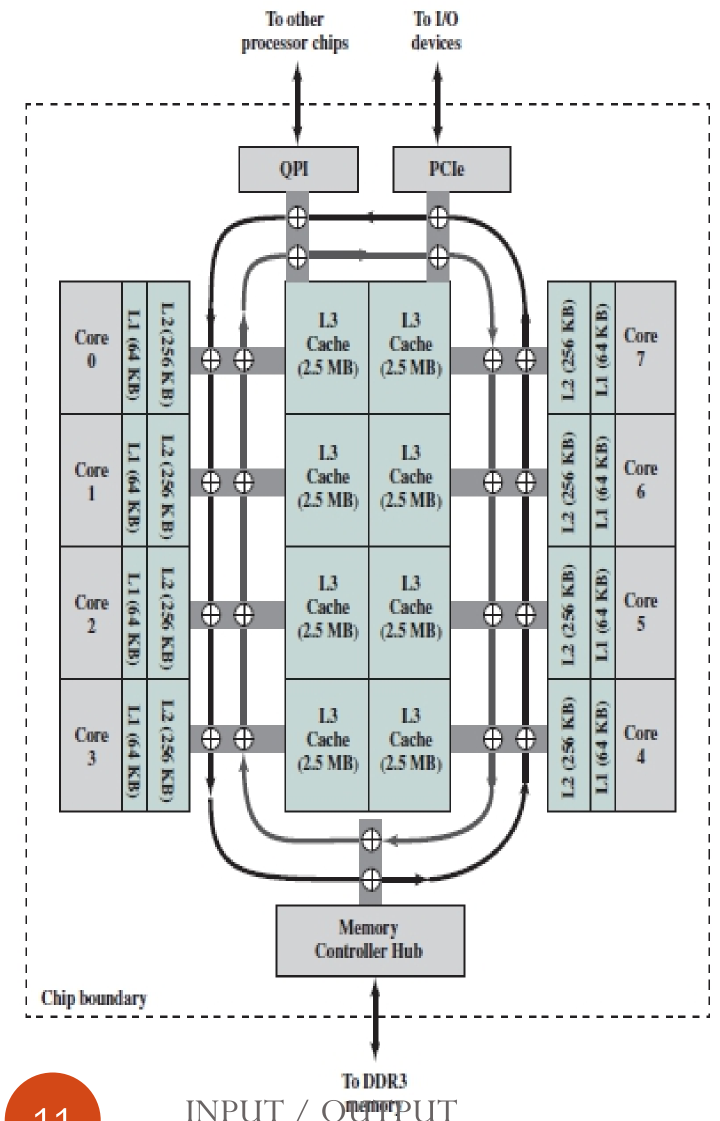
1. Each component that attaches to the bidirectional ring (QPI, PCIe, L3 cache, L2 cache) is considered a ring agent, and implements ring agent logic.

2. The ring agents cooperate via a distributed protocol to request and allocate access to the ring, in the form of time slots.

3. When an agent has data to send, it chooses the ring direction that results in the shortest path to the destination and transmits when a scheduling slot is available.

The ring architecture provides good performance and scales well for multiple cores, up to a point. For systems with a greater number of cores, multiple rings are used, with each ring supporting some of the cores.

DMA use of the cache



In traditional DMA operation, data are exchanged between main memory and an I/O device by means of the system interconnection structure, such as **a bus, ring, or QPI point-to-point matrix.** For example, if the Xeon E5-2600/4600 used a traditional DMA technique, output would proceed as follows.

An I/O driver running on a core would send an I/O command to the I/O controller (**labeled PCIe**) with the location and size of the buffer in main memory containing the data to be transferred.

The I/O controller issues a read request that is routed to the memory controller hub (MCH), which accesses the data on DDR3 memory and puts it on the system ring for delivery to the I/O controller

.

The L3 cache is not involved in this transaction and one or more off- chip memory reads are required.

Similarly, for input, data arrive from the I/O controller and is delivered over the system ring to the MCH and written out to main memory.

The MCH must also invalidate any L3 cache lines corresponding to the updated memory locations. In this case, one or more off-chip memory writes are required. Further, if an application wants to access the new data, a main memory read is required.

- With the availability of large amounts of last-level cache, a **more efficient technique** is possible, and is used by the Xeon E5-2600/4600.
- For **output**, when the I/O controller issues a read request, the MCH first checks to see if the data are in the L3 cache.
- This is likely to be the case, if an application has recently written data into the memory block to be output.
- In that case, the MCH directs data from the L3 cache to the I/O controller; **no main memory accesses** are needed.
- However, it also causes the data to be evicted from cache, that is, the act of reading by an I/O device causes data to be evicted.

- Thus, the I/O operation proceeds efficiently because it **does not require main memory access.**

- But, if an application does need that data in the future, it must be read back into the L3 cache from main memory.

- The **input operation** on the Xeon E5-2600/4600 operates as described in the previous section ; the L3 cache is not involved. Thus, the performance improvement involves only output operations.

- A final point- Although the output transfer is **directly from cache to the I/O controller**, the term *direct cache access is not used for this feature. Rather, the term* is reserved for the I/O protocol application.

Cache-Related Performance Issues

- Network traffic is transmitted in the form of a sequence of protocol blocks, called packets or protocol data units.
- The lowest, or link, level protocol is typically Ethernet, so that each arriving and departing block of data consists of an Ethernet packet containing as payload the higher-level protocol packet.
- The higher- level protocols are usually the Internet Protocol (IP), operating on top of Ethernet, and the Transmission Control Protocol (TCP), operating on top of IP. Accordingly, the Ethernet payload consists of a block of data with a TCP header and an IP header.
- For outgoing data, Ethernet packets are formed in a peripheral component, such as an I/O controller or network interface controller (NIC).
- Similarly, for incoming traffic, the I/O controller strips off the Ethernet information and delivers the TCP/ IP packet to the host CPU.

Cache-Related Performance Issues for Incoming traffic

To clarify the performance issue and to explain the benefit of DCA as a way of improving performance, let us look at the processing of protocol traffic in more detail for incoming traffic. In general terms, the following steps occur:

1. **Packet arrives:** The NIC receives an incoming Ethernet packet. The NIC processes and strips off the Ethernet control information. This includes doing an error detection calculation. The remaining TCP/IP packet is then transferred to the system's DMA module, which generally is part of the NIC. The NIC also creates a packet descriptor with information about the packet, such as its buffer location in memory.

2. **DMA:** The DMA module transfers data, including the packet descriptor, to main memory. It must also invalidate the corresponding cache lines, if any.

3. **NIC interrupts host:** After a number of packets have been transferred, the NIC issues an interrupt to the host processor.

4. **Retrieve descriptors and headers:** The core processes the interrupt, invoking an interrupt handling procedure, which reads the descriptor and header of the received packets.

5. **Cache miss occurs:** Because this is new data coming in, the cache lines corresponding to the system buffer containing the new data are invalidated. Thus, the core must stall to read the data from main memory into cache, and then to core registers.

6. **Header is processed:** The protocol software executes on the core to analyze the contents of the TCP and IP headers. This will likely include accessing a transport control block (TCB), which contains context information related to TCP. The TCB access may or may not trigger a cache miss, necessitating a main memory access.

7. **Payload transferred:** The data portion of the packet is transferred from the system buffer to the appropriate application buffer.

Cache-Related Performance Issues for OUTGOING TRAFFIC

For outgoing traffic, the following steps occur:

1. **Packet transfer requested:** When an application has a block of data to transfer to a remote system, it places the data in an application buffer and alerts the OS with some type of system call.

2. **Packet created:** The OS invokes a TCP/IP process to create the TCP/IP packet for transmission. The TCP/IP process accesses the TCB (which may involve a cache miss) and creates the appropriate headers. It also reads the data from the application buffer, and then places the completed packet (headers plus data) in a system buffer. Note that the data that is written into the system buffer also exists in the cache. The TCP/IP process also creates a packet descriptor that is placed in memory shared with the DMA module.

3. **Output operation invoked:** This uses a device driver program to signal the DMA module that output is ready for the NIC.

4. **DMA transfer:** The DMA module reads the packet descriptor, then a DMA transfer is performed from main memory or the last- level cache to the NIC. Note that DMA transfers invalidate the cache line in cache even in the case of a read (by the DMA module). If the line is modified, this causes a write back. The core does not do the invalidates. The invalidates happen when the DMA module reads the data.

5. **NIC signals completion:** After the transfer is complete, the NIC signals the driver on the core that originated the send signal.

6. **Driver frees buffer:** Once the driver receives the completion notice, it frees up the buffer space for reuse. The core must also invalidate the cache lines containing the buffer data.

DIRECT CACHE ACCESS STRATEGIES

- The simplest strategy is one that was implemented as a prototype on a number of Intel Xeon processors between 2006 and 2010.
- This form of DCA applies only to incoming network traffic.
- The DCA function in the memory controller sends a prefetch hint to the core as soon as the data are available in system memory.
- This enables the core to prefetch the data packet from the system buffer, thus avoiding cache misses and the associated waste of core cycles.

- While this simple form of DCA does provide some improvement, much more substantial gains can be realized by **avoiding the system buffer** in main memory altogether.

- For the specific function of protocol processing, note that the packet and packet descriptor information are accessed only once in the system buffer by the core.

- For **incoming packets**, the core reads the data from the buffer and transfers the packet payload to an application buffer. It has no need to access that data in the system buffer again.

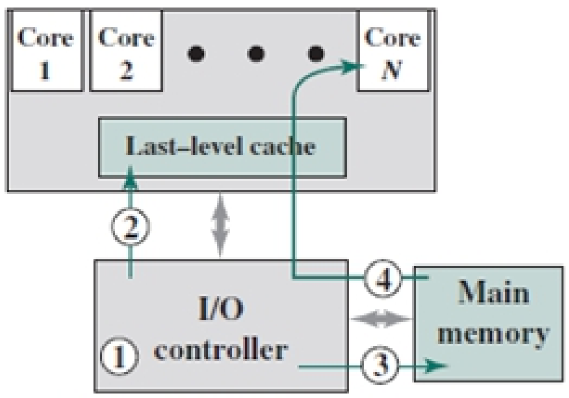
- Similarly, for **outgoing packets**, once the core has placed the data in the system buffer, it has no need to access that data again.

- Suppose, therefore, that the I/O system were equipped not only with the capability of directly accessing main memory, but also of accessing the cache, both for input and output operations. Then it would be possible to use the last-Level cache instead of the main memory to buffer packets and descriptors of incoming and outgoing packets.

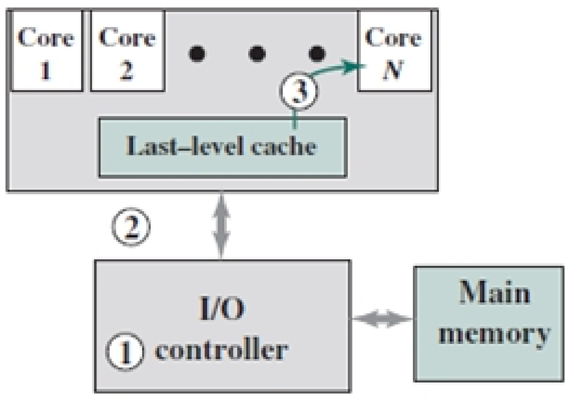
- This last approach, is a true DCA. It has also been described as **cache injection. A version of this more complete** form of DCA is implemented in Intel's Xeon processor line, referred to as **Direct Data I/O .**

DIRECT DATA I/O

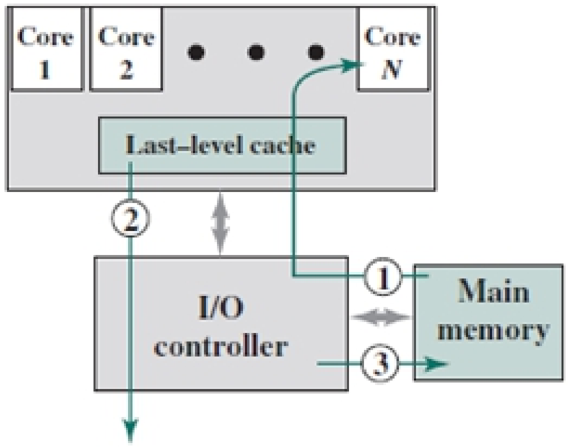
COMPARISON OF DMA AND DDIO



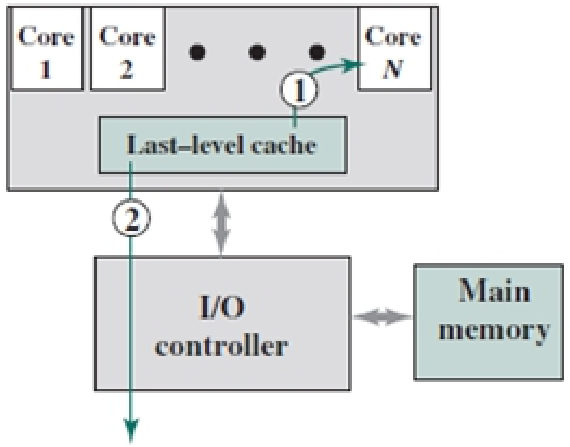
(a) Normal DMA transfer to memory



(b) DDIO transfer to cache

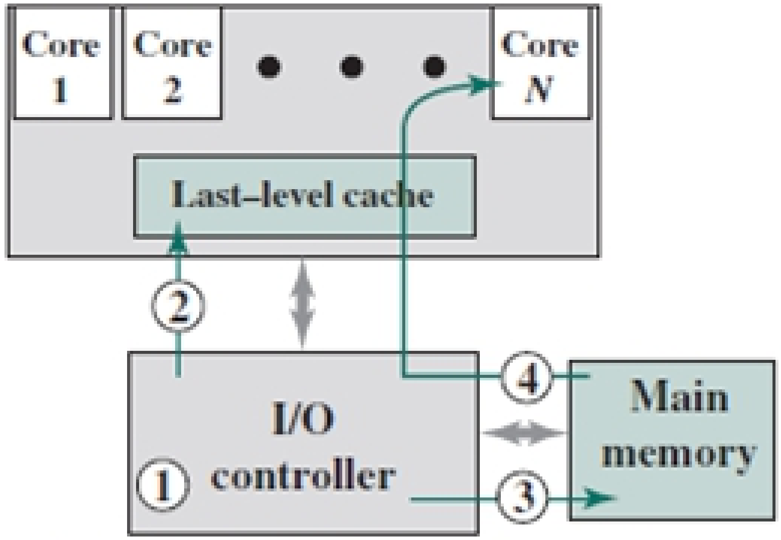


(c) Normal DMA transfer to I/O



(d) DDIO transfer to I/O

PACKET INPUT
(a) Normal DMA transfer to memory



(a) Normal DMA transfer to memory

- Intel Direct Data I/O (DDIO) is implemented on all of the Xeon E5 family of processors.
- ***packet input-*** First, we look at the case of a packet arriving at the network interface controller (NIC) from the network.
- The NIC initiates a memory write (1).
- Then the NIC invalidates the cache lines corresponding to the system buffer (2).
- Next, the DMA operation is performed, depositing the packet directly into main memory (3).
- Finally, after the appropriate core receives a DMA interrupt signal, the core can read the packet data from memory through the cache (4).

- Before discussing the processing of an incoming packet using DDIO, we need to summarize the discussion of cache write policy and introduce a new technique.
- Recall that there are two techniques for dealing with an update to a cache line:

■ ■ **Write through:** All write operations are made to main memory as well as to the cache, ensuring that main memory is always valid. Any other core–cache module can monitor traffic to main memory to maintain consistency within its own local cache.

■ ■ **Write back:** Updates are made only in the cache. When an update occurs, a dirty bit associated with the line is set. Then, when a block is replaced, it is written back to main memory if and only if the dirty bit is set.

- DDIO uses the write-back strategy in the L3 cache.
- A cache write operation may encounter a cache miss, which is dealt with by one of two strategies:

■ ■ **Write allocate:** The required line is loaded into the cache from main memory. Then, the line in the cache is updated by the write operation. This scheme is typically used with the write-back method.

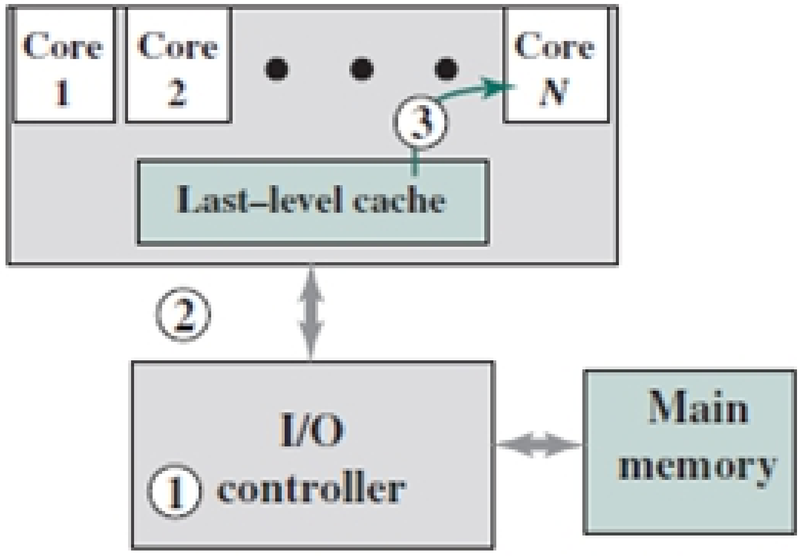
■ ■ **Non-write allocate:** The block is modified directly in main memory. No change is made to the cache. This scheme is typically used with the write- through method.

- With the above in mind, we can describe the DDIO strategy for inbound transfers initiated by the NIC.

1. If there is a **cache hit**, the cache line is updated, but not main memory; this is simply the write-back strategy for a cache hit. The Intel literature refers to this as write update.

2. If there is a **cache miss**, the write operation occurs to a line in the cache that will not be written back to main memory. Subsequent writes update the cache line, again with no reference to main memory or no future action that writes this data to main memory. The Intel documentation [INTE12] refers to this as *write allocate, which* unfortunately is not the same meaning for the term in the general cache literature.

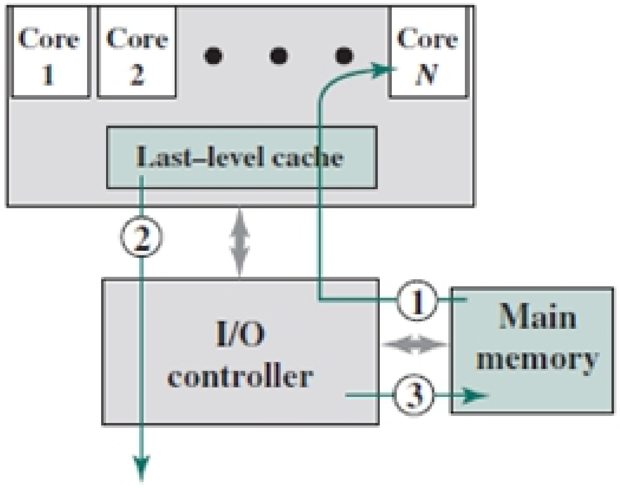
(b) DDIO transfer to cache



(b) DDIO transfer to cache

- Figure shows the operation for DDIO input.
- The NIC initiates a memory write (1).
- Then the NIC invalidates the cache lines corresponding to the system buffer and deposits the incoming data in the cache (2).
- Finally, after the appropriate core receives a DCA interrupt signal, the core can read the packet data from the cache (3).

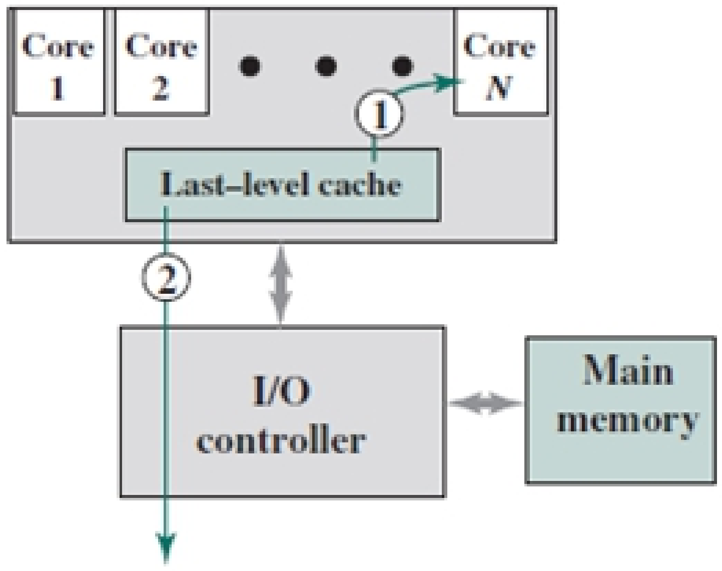
PACKET OUTPUT
(c) Normal DMA transfer to I/O



(c) Normal DMA transfer to I/O

- Figure shows the steps involved for a DMA operation for outbound packet transmission.
- The TCP/IP protocol handler executing on the core reads data in from an application buffer and writes it out to a system buffer. These data access operations result in cache misses and cause data to be read from memory and into the L3 cache (1).
- When the NIC receives notification for starting a transmit operation, it reads the data from the L3 cache and transmits it (2).
- The cache access by the NIC causes the data to be evicted from the cache and written back to main memory (3).

(d) DDIO transfer to I/O



(d) DDIO transfer to I/O

- Figure shows the steps involved for a DDIO operation for packet transmission.
- The TCP/IP protocol handler creates the packet to be transmitted and stores it in allocated space in the L3 cache (1), but not in main memory (2).
- The read operation initiated by the NIC is satisfied by data from the cache, without causing evictions to main memory.
- It should be clear from these side-by-side comparisons that DDIO is more efficient than DMA for both incoming and outgoing packets and is therefore better able to keep up with the high packet traffic rate.

REVIEW QUESTIONS

1. List three broad classifications of external, or peripheral, devices.
2. What is the International Reference Alphabet?
3. What are the major functions of an I/O module?
4. List and briefly define three techniques for performing I/O.
5. What is the difference between memory- mapped I/O and isolated I/O?
6. When a device interrupt occurs, how does the processor determine which device issued the interrupt?
7. When a DMA module takes control of a bus, and while it retains control of the bus, what does the processor do?

THANK YOU