Chapter 10: Multi-core computers

A multicore computer, also known as a chip multiprocessor, combines two or more processors (called cores) on a single piece of silicon (called a die). Typically, each core consists of all of the components of an independent processor, such as registers, ALU, pipeline hardware, and control unit, plus L1 instruction and data caches. In addition to the multiple cores, contemporary multicore chips also include L2 cache and, in some cases, L3 cache.

HARDWARE PERFORMANCE ISSUES

Microprocessor systems have experienced a steady, exponential increase in execution performance for decades. This increase is due partly to refinements in the organization of the processor on the chip, and partly to the increase in the clock frequency.

Increase in Parallelism

The organizational changes in processor design have primarily been focused on increasing instruction-level parallelism, so that more work could be done in each clock cycle.These changes include, in chronological order:

• Pipelining: Individual instructions are executed through a pipeline of stages so that while one instruction is executing in one stage of the pipeline, another instruction is executing in another stage of the pipeline.

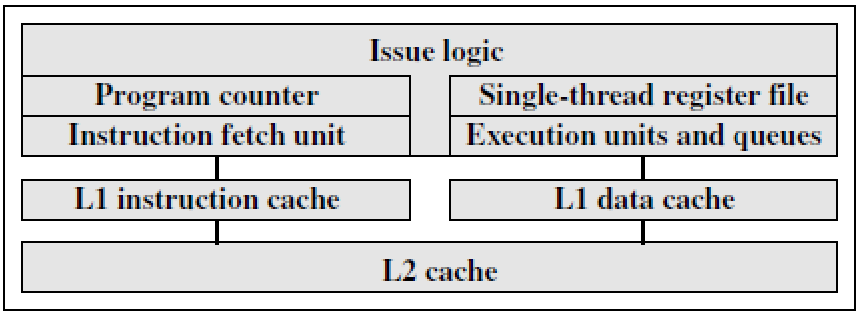
• Superscalar: Multiple pipelines are constructed by replicating execution resources.This enables parallel execution of instructions in parallel pipelines, so long as hazards are avoided.

• Simultaneous multithreading (SMT): Register banks are replicated so that multiple threads can share the use of pipeline resources.

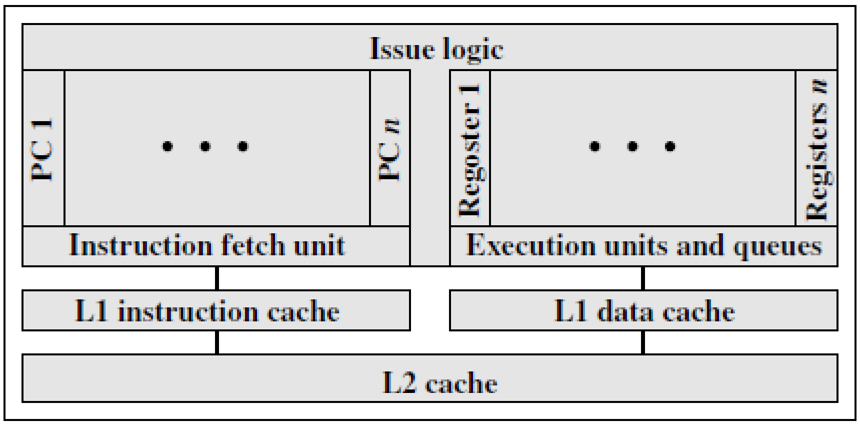
For each of these innovations, designers have over the years attempted to increase the performance of the system by adding complexity. In the case of pipelining, simple three-stage pipelines were replaced by pipelines with five stages, and then many more stages, with some implementations having over a dozen stages. There is a practical limit to how far this trend can be taken, because with more stages, there is the need for more logic, more interconnections, and more control signals.With superscalar organization, performance increases can be achieved by increasing the number of parallel pipelines. Again, there are diminishing returns as the number of pipelines increases. More logic is required to manage hazards and to stage instruction resources. Eventually, a single thread of execution reaches the point where hazards and resource dependencies prevent the full use of the multiple pipelines available. This same point of diminishing returns is reached with SMT, as the complexity of managing multiple threads over a set of pipelines limits the number of threads and number of pipelines that can be effectively utilized.

There is a related set of problems dealing with the design and fabrication of the computer chip. The increase in complexity to deal with all of the logical issues related to very long pipelines, multiple superscalar pipelines, and multiple SMT register banks means that increasing amounts of the chip area is occupied with coordinating and signal transfer logic.

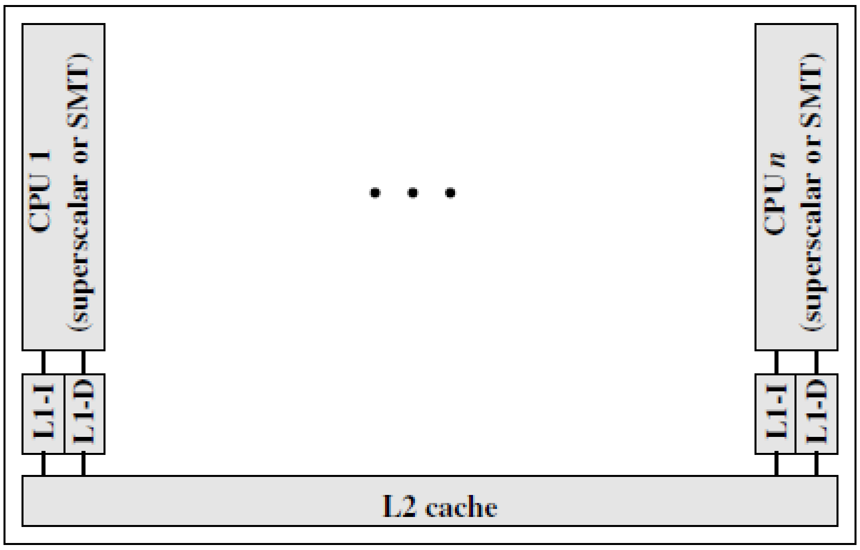
This increases the difficulty of designing, fabricating, and debugging the chips. The increasingly difficult engineering challenge related to processor logic is one of the reasons that an increasing fraction of the processor chip is devoted to the simpler memory logic.



(a) Superscalar



(b) Simultaneous multithreading

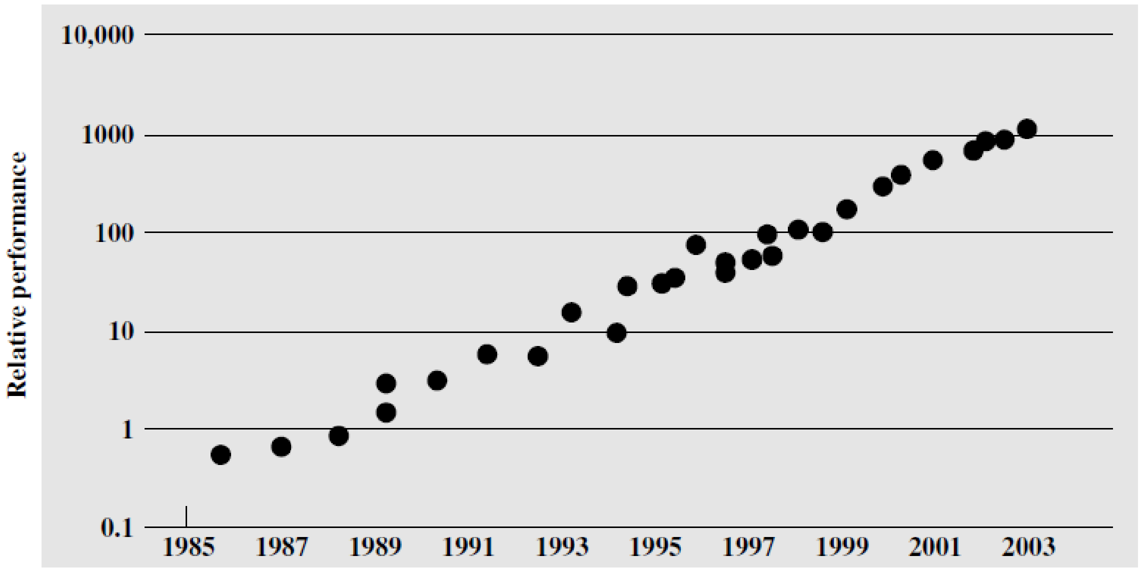


(c) Multicore

Figure 18.1 Alternative Chip Organizations

Power Consumption

To maintain the trend of higher performance as the number of transistors per chip rise, designers have resorted to more elaborate processor designs (pipelining, superscalar, SMT) and to high clock frequencies. Unfortunately, power requirements have grown exponentially as chip density and clock frequency have risen. This is shown in the lowest graph in Figure 18.2. One way to control power density is to use more of the chip area for cache memory. Memory transistors are smaller and have a power density an order of magnitude lower than that of logic (see Figure 18.3a). As Figure 18.3b, from [BORK03], shows, and percentage of the chip area devoted to memory has grown to exceed 50% as the chip transistor density has increased. Figure 18.4, from [BORK07], shows where the power consumption trend is leading. By 2015, we can expect to see microprocessor chips with about 100 billion transistors on a 300 mm2 die. Assuming about 50–60% of the chip area is devoted to memory, the chip will support cache memory of about 100 MB and leave over 1 billion transistors available for logic. How to use all those logic transistors is a key design issue. As discussed earlier in this section, there are limits to the effective use of such techniques as superscalar and SMT. In general terms, the experience of recent decades has been encapsulated in a rule of thumb known as **Pollack's rule** [POLL99], which states that performance increase is roughly proportional to square root of increase in complexity. In other words, if you double the logic in a processor core, then it delivers only 40% more performance. In principle, the use of multiple cores has the potential to provide near-linear performance improvement with the increase in the number of cores. Power considerations provide another motive for moving toward a multicore organization. Because the chip has such a huge amount of cache memory, it becomes unlikely that any one thread of execution can effectively use all that memory. Even with SMT, you are multithreading in a relatively limited fashion and cannot therefore fully exploit a gigantic cache, whereas a number of relatively independent threads or processes have a greater opportunity to take full advantage of the cache memory.



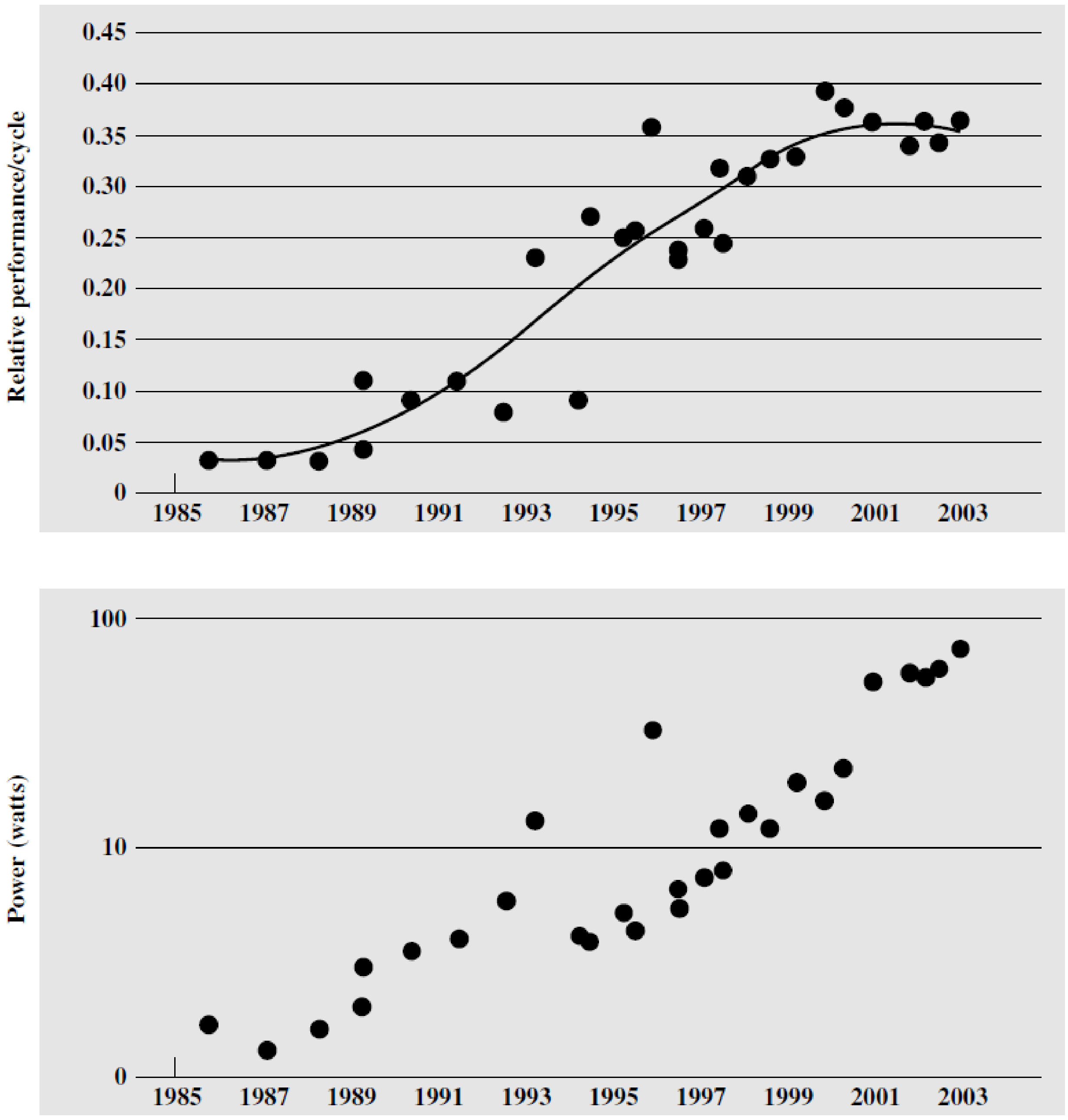


Figure 18.2 Some Intel Hardware Trends

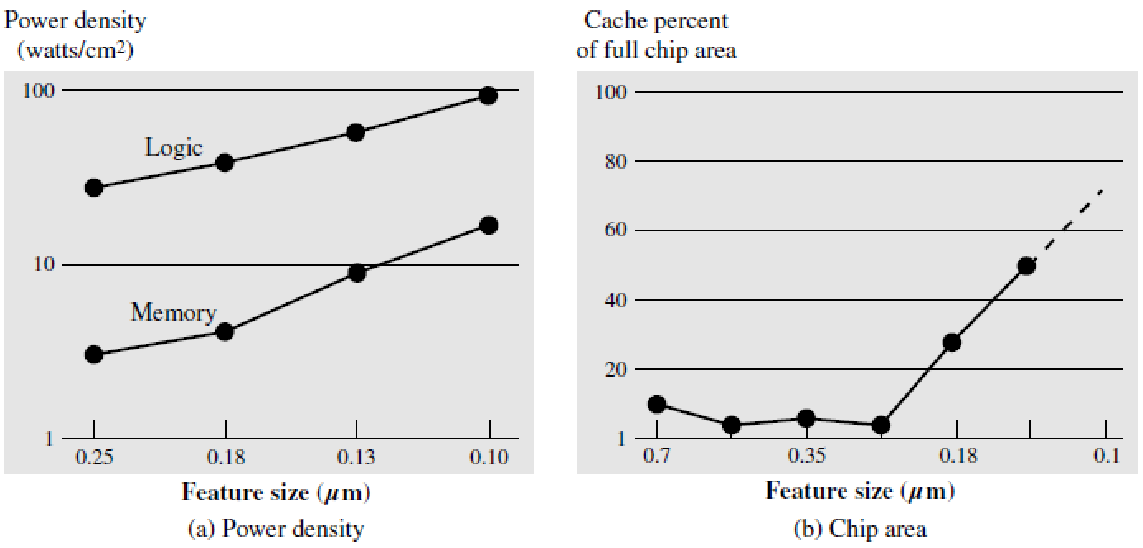


Figure 18.3 Power and Memory Considerations

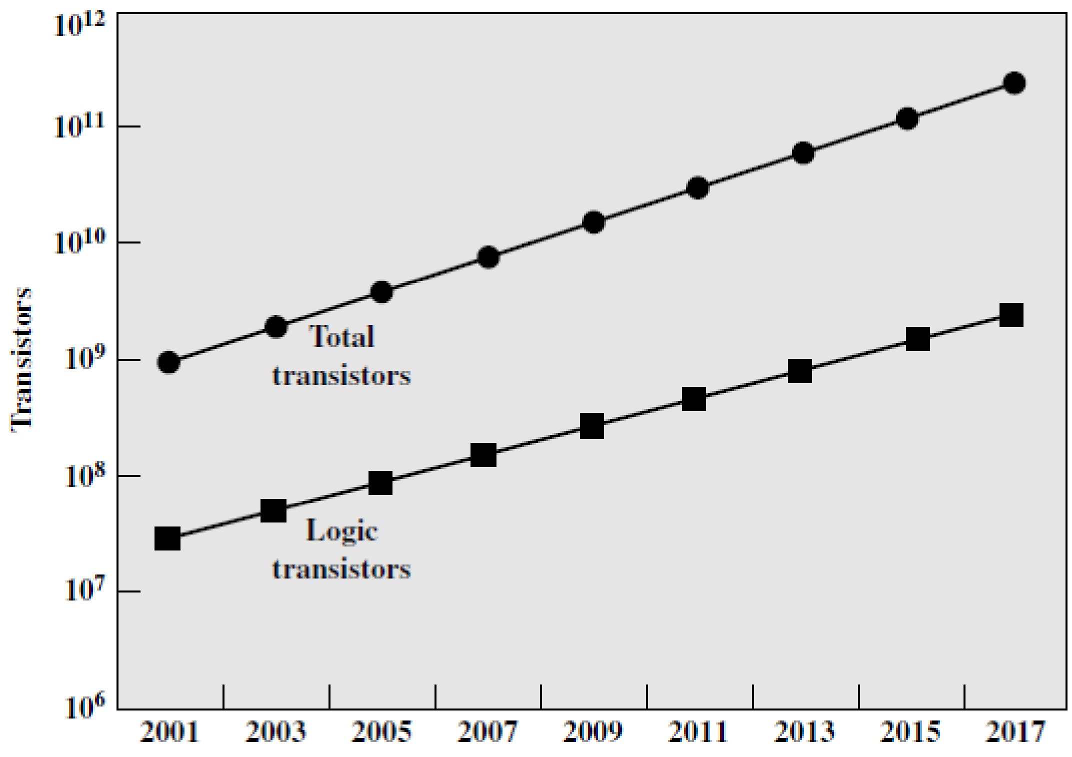


Figure 18.4 Chip Utilization of Transistors

Software performance issues

Software on Multicore

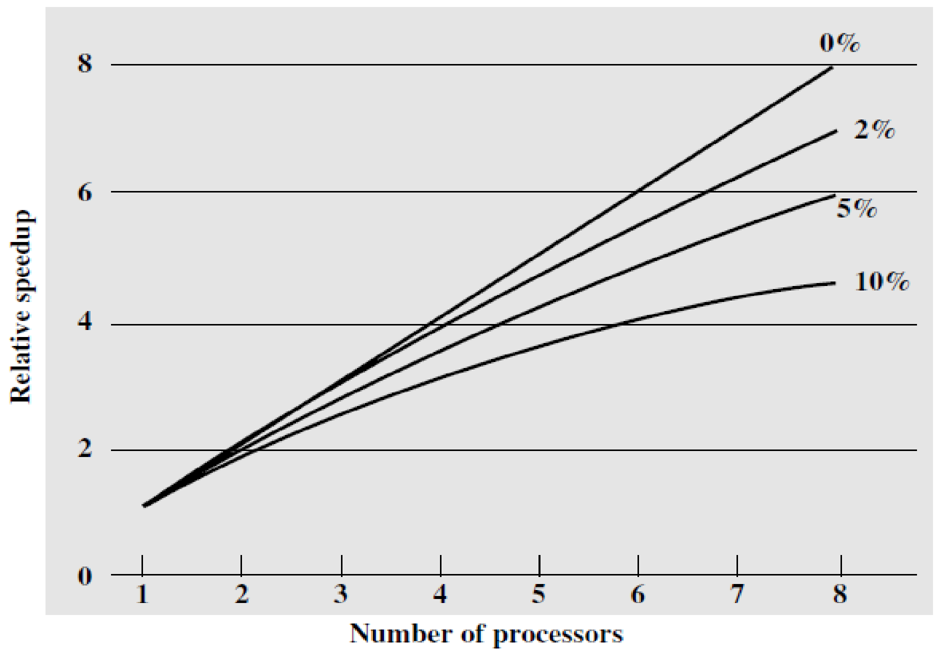
The potential performance benefits of a multicore organization depend on the ability to effectively exploit the parallel resources available to the application. Let us focus first on a single application running on a multicore system. Amdahl's law states that or speed up ratio:

$$speed\ up = \frac{\text{time to execute program on a single processor}}{\text{time to execute program on N parallel processors}}$$

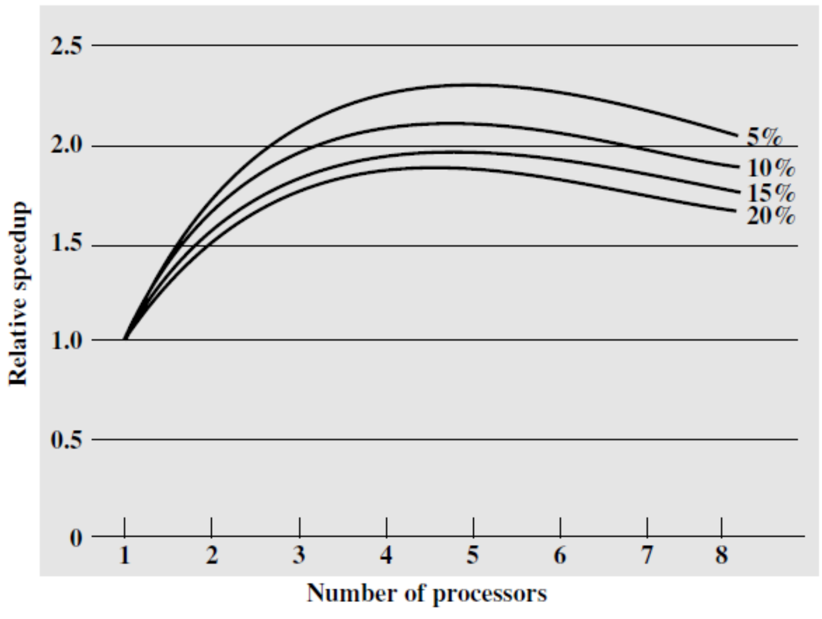
$$speed\ up = \frac{1}{(1-\mathrm{f}) + \mathrm{f}/\mathrm{n}}$$

The law assumes a program in which a fraction (1-f) of the execution time involves code that is inherently serial and a fraction f that involves code that is infinitely parallelizable with no scheduling overhead.

This law appears to make the prospect of a multicore organization attractive. But as Figure 18.5a shows, even a small amount of serial code has a noticeable impact. If only 10% of the code is inherently serial the running the program on a multicore system with 8 processors yields a performance gains of only a factor of 4.7. In addition, software typically incurs overhead as a result of communication and distribution of work to multiple processors and cache coherence overhead. This results in a curve where performance peaks than then begins to degrade because of the increased burden of the overhead of using multiple processors. Figure 18.5b, from [MCDO05], is a representative example.



(a) Speedup with 0%, 2%, 5%, and 10% sequential portions



(b) Speedup with overheads

Figure 18.5 Performance Effect of Multiple Cores

However, software engineers have been addressing this problem and there are numerous applications in which it is possible to effectively exploit a multicore system. [MCDO05] reports on a set of database applications, in which great attention was paid to reducing the serial fraction within hardware architectures, operating systems, middleware, and the database application software. Figure 18.6 shows the result. As this example shows, database management systems and database applications are one area in which multicore systems can be used effectively. Many kinds of servers can also effectively use the parallel multicore organization, because servers typically handle numerous relatively independent transactions in parallel.

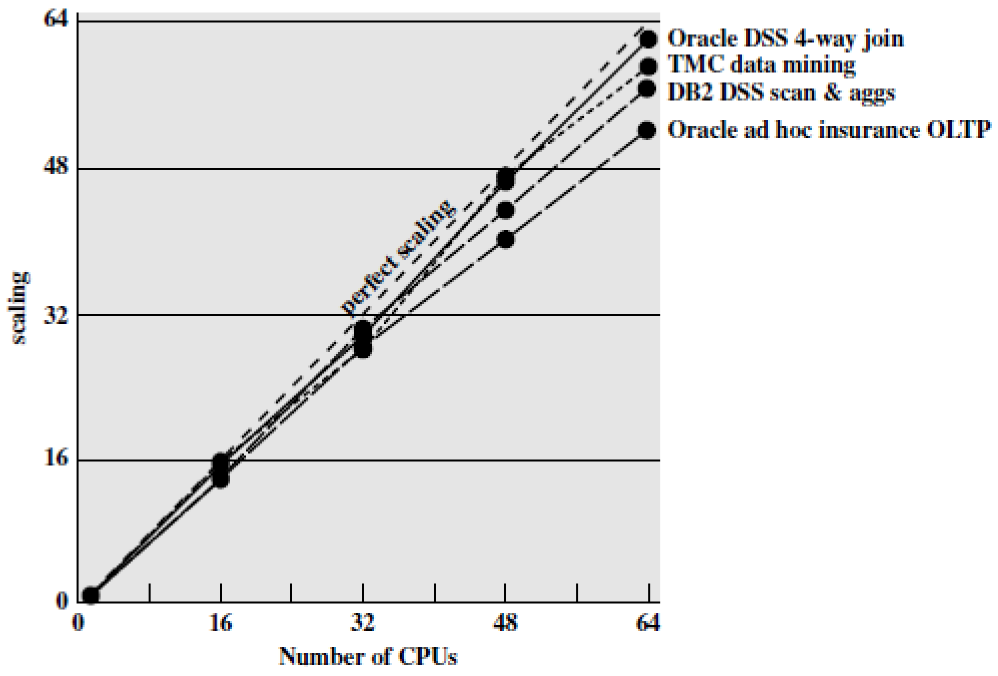


Figure 18.6 Scaling of Database Workloads on Multiple-Processor Hardware

In addition to general-purpose server software, a number of classes of applications benefit directly from the ability to scale throughput with the number of cores. Lists the following examples:

• **Multithreaded native applications:** Multithreaded applications are characterized by having a small number of highly threaded processes. Examples of threaded applications include Lotus Domino or Siebel CRM (Customer Relationship Manager).

• **Multiprocess applications:** Multiprocess applications are characterized by the presence of many single-threaded processes. Examples of multi-process applications include the Oracle database, SAP, and PeopleSoft.

• **Java applications:** Java applications embrace threading in a fundamental way. Not only does the Java language greatly facilitate multithreaded applications, but the Java Virtual Machine is a multithreaded process that provides scheduling and memory management for Java applications.

• **Multiinstance applications:** Even if an individual application does not scale to take advantage of a large number of threads, it is still possible to gain from multicore architecture by running multiple instances of the application in parallel. If multiple application instances require some degree of isolation, virtualization technology (for the hardware of the operating system) can be used to provide each of them with its own separate and secure environment.
Application Example: Valve Game Software

Multi core organization

At a top level of description, the main variables in a multicore organization are as follows:

• The number of core processors on the chip

• The number of levels of cache memory

• The amount of cache memory that is shared

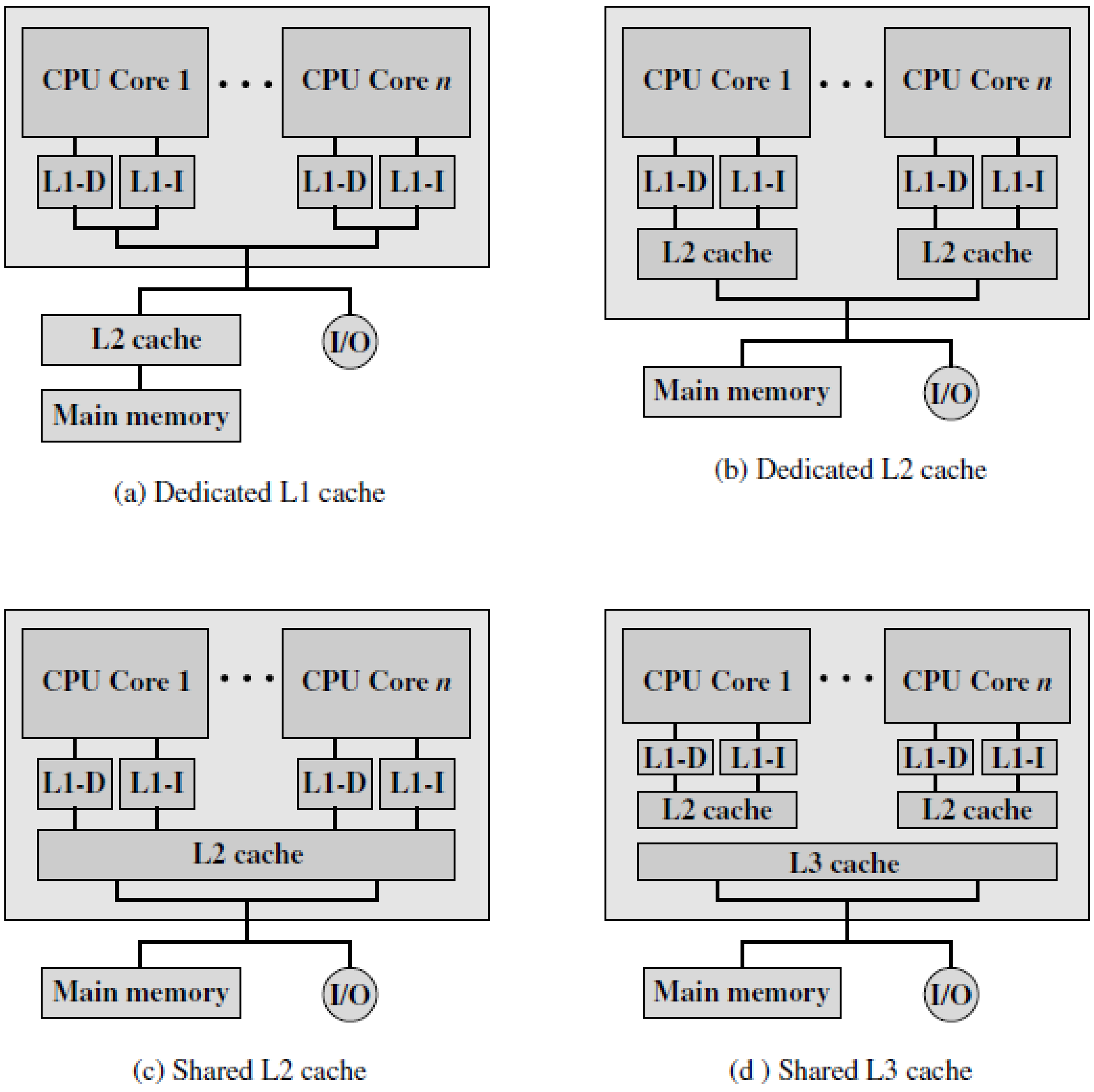


Figure 18.8 Multicore Organization Alternatives

Figure 18.8 shows four general organizations for multicore systems. Figure 18.8a is an organization found in some of the earlier multicore computer chips and is still seen in embedded chips. In this organization, the only on-chip cache is L1 cache, with each core having its own dedicated L1 cache. Almost invariably, the L1 cache is divided into instruction and data caches. An example of this organization is the ARM11 MPCore. The organization of Figure 18.8b is also one in which there is no on-chip cache sharing. In this, there is enough area available on the chip to allow for L2 cache. An example of this organization is the AMD Opteron. Figure 18.8c shows a similar allocation of chip space to memory, but with the use of a shared L2 cache. The Intel Core Duo has this organization. Finally, as the amount of cache memory available on the chip continues to grow, performance considerations dictate splitting off a separate, shared L3 cache, with dedicated L1 and L2 caches for each core processor. The Intel Core i7 is an example of this organization. The use of a shared L2 cache on the chip has several advantages over exclusive reliance on dedicated caches:

1. Constructive interference can reduce overall miss rates. That is, if a thread on one core accesses a main memory location, this brings the frame containing the referenced location into the shared cache. If a thread on another core soon thereafter accesses the same memory block, the memory locations will already be available in the shared on-chip cache.

2. A related advantage is that data shared by multiple cores is not replicated at the shared cache level.

3. With proper frame replacement algorithms, the amount of shared cache allocated to each core is dynamic, so that threads that have a less locality can employ more cache.

4. Interprocessor communication is easy to implement, via shared memory locations.

5. The use of a shared L2 cache confines the cache coherency problem to the L1 cache level, which may provide some additional performance advantage.

A potential advantage to having only dedicated L2 caches on the chip is that each core enjoys more rapid access to its private L2 cache. This is advantageous for threads that exhibit strong locality. As both the amount of memory available and the number of cores grow, the use of a shared L3 cache combined with either a shared L2 cache or dedicated per core L2 caches seems likely to provide better performance than simply a massive shared L2 cache. Another organizational design decision in a multicore system is whether the individual cores will be superscalar or will implement simultaneous multithreading (SMT). For example, the Intel Core Duo uses superscalar cores, whereas the Intel Core i7 uses SMT cores. SMT has the effect of scaling up the number of hardware level threads that the multicore system supports. Thus, a multicore system with four cores and SMT that supports four simultaneous threads in each core appears the same to the application level as a multicore system with 16 cores. As software is developed to more fully exploit parallel resources, an SMT approach appears to be more attractive than a superscalar approach.

Intel Core Duo

The Intel Core Duo, introduced in 2006, implements two x86 superscalar processors with a shared L2 cache (Figure 18.8c). The general structure of the Intel Core Duo is shown in Figure 18.9. Let us consider the key elements starting from the top of the figure. As is common in multicore systems, each core has its own dedicated **L1 cache**. In this case, each core has a 32-KB instruction cache and a 32-KB data cache. Each core has an independent **thermal control unit**. With the high transistor density of today's chips, thermal management is a fundamental capability, especially for laptop and mobile systems. The Core Duo thermal control unit is designed to manage chip heat dissipation to maximize processor performance within thermal constraints. Thermal management also improves ergonomics with a cooler system and lower fan acoustic noise. In essence, the thermal management unit monitors digital sensors for high-accuracy die temperature measurements. Each core can be defined as in independent thermal zone. The maximum temperature for each thermal zone is reported separately via dedicated registers that can be polled by software. If the temperature in a core exceeds a threshold, the thermal control unit reduces the clock rate for that core to reduce heat generation. The next key element of the Core Duo organization is the **Advanced Programmable Interrupt Controller** (APIC). The APIC performs a number of functions, including the following:

1. TheAPIC can provide interprocessor interrupts, which allow any process to interrupt any other processor or set of processors. In the case of the Core Duo, a thread in one core can generate an interrupt, which is accepted by the local APIC, routed to the APIC of the other core, and communicated as an interrupt to the other core.

2. The APIC accepts I/O interrupts and routes these to the appropriate core.

3. Each APIC includes a timer, which can be set by the OS to generate an interrupt to the local core.

The **power management logic** is responsible for reducing power consumption when possible, thus increasing battery life for mobile platforms, such as laptops. In essence, the power management logic monitors thermal conditions and CPU activity and adjusts voltage levels and power consumption appropriately. It includes an advanced power-gating capability that allows for an ultra fine-grained logic control that turns on individual processor logic subsystems only if and when they are needed. Additionally, many buses and arrays are split so that data required in some modes of operation can be put in a low power state when not needed. The Core Duo chip includes a shared 2-MB **L2 cache**. The cache logic allows for a dynamic allocation of cache space based on current core needs, so that one core can be assigned up to 100% of the L2 cache. The L2 cache includes logic to support the MESI protocol for the attached L1 caches. The key point to consider is when a cache write is done at the L1 level. A cache line gets the M state when a processor writes to it; if the line is not in E or M-state prior to writing it, the cache sends a Read-For-Ownership (RFO) request that ensures that the line exists in the L1 cache and is in the I state in the other L1 cache. The Intel Core Duo extends this protocol to take into account the case when there is multiple Core Duo chips organized as a symmetric multiprocessor (SMP) system. The L2 cache controller allow the system to distinguish between a situation in which data are shared by the two local cores, but not with the rest of the world, and a situation in which the data are shared by one or more caches on the die as well as by an agent on the external bus (can be another processor).When a core issues an RFO, if the line is shared only by the other cache within the local die, we can resolve the RFO internally very fast, without going to the external bus at all. Only if the line is shared with another agent on the external bus do we need to issue the RFO externally.

The **bus interface** connects to the external bus, known as the Front Side Bus, which connects to main memory, I/O controllers, and other processor chips.

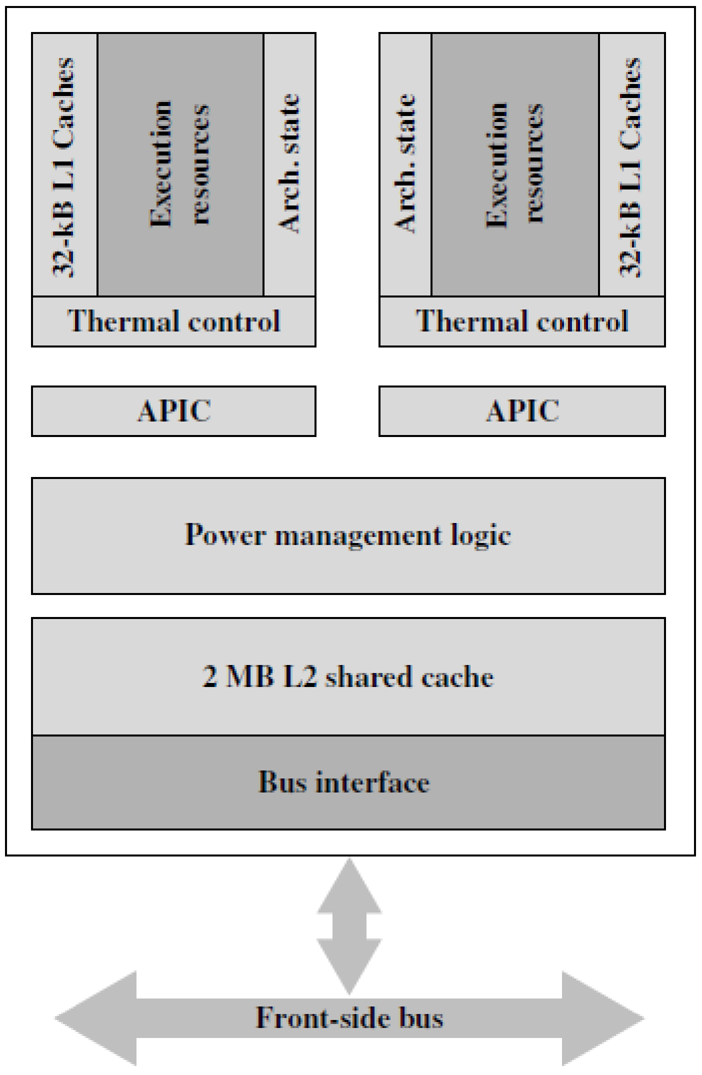


Figure 18.9 Intel Core Duo Block Diagram

Intel Core i7

The Intel Core i7, introduced in November of 2008, implements four x86 SMT processors, each with a dedicated L2 cache, and with a shared L3 cache (Figure 18.8d). The general structure of the Intel Core i7 is shown in Figure 18.10. Each core has its own **dedicated L2 cache** and the four cores share an 8-MB **L3 cache**. One mechanism Intel uses to make its caches more effective is prefetching, in which the hardware examines memory access patterns and attempts to fill the caches speculatively with data that's likely to be requested soon. It is interesting to compare the performance of this three-level on chip cache organization with a comparable two level organization from Intel. Table 18.1 shows the cache access latency, in terms of clock cycles for two Intel multicore systems running at the same clock frequency. The Core 2 Quad has a shared L2 cache, similar to the Core Duo. The Core i7 improves on L2 cache performance with the use of the dedicated L2 caches, and provides a relatively high-speed access to the L3 cache. The Core i7 chip supports two forms of external communications to other chips. The **DDR3 memory controller** brings the memory controller for the DDR main memory2 onto the chip. The interface supports three channels that are 8 bytes wide for a total bus width of 192 bits, for an aggregate data rate of up to 32 GB/s. With the memory controller on the chip, the Front Side Bus is eliminated. The **Quick Path Interconnect** (QPI) is a cache-coherent, point-to-point link based electrical interconnect specification for Intel processors and chipsets. It enables high-speed communications among connected processor chips. The QPI link operates at 6.4 GT/s (transfers per second). At 16 bits per transfer, that adds up to 12.8 GB/s, and since QPI links involve dedicated bidirectional pairs, the total bandwidth is 25.6 GB/s.

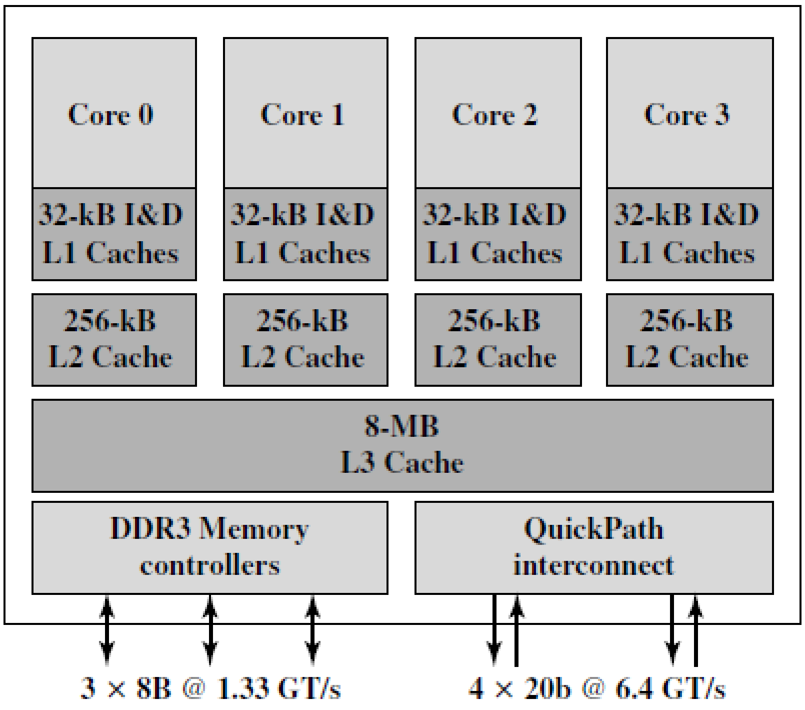


Figure 18.10 Intel Core i7 Block Diagram

The ARM11 MPCore is a multicore product based on the ARM11 processor family. The ARM11 MPCore can be configured with up to four processors, each with its own L1 instruction and data caches, per chip. Table 18.1 lists the configurable options for the system, including the default values. Figure 18.11 presents a block diagram of the ARM11 MPCore. The key elements of the system are as follows:

- **Distributed interrupt controller (DIC):** Handles interrupt detection and interrupt prioritization. The DIC distributes interrupts to individual processors.

- **Timer:** Each CPU has its own private timer that can generate interrupts.

- **Watchdog:** Issues warning alerts in the event of software failures. If the watchdog is enabled, it is set to a predetermined value and counts down to 0. It is periodically reset. If the watchdog value reaches zero, an alert is issued.

- **CPU interface:** Handles interrupt acknowledgement, interrupt masking, and interrupt completion acknowledgement

• **CPU:** A single ARM11 processor. Individual CPUs are referred to as **MP11 CPUs**.

• **Vector floating-point (VFP) unit:** A coprocessor that implements floating point operations in hardware.

• **L1 cache:** Each CPU has its own dedicated L1 data cache and L1 instruction cache.

• **Snoop control unit (SCU):** Responsible for maintaining coherency among L1

data caches.

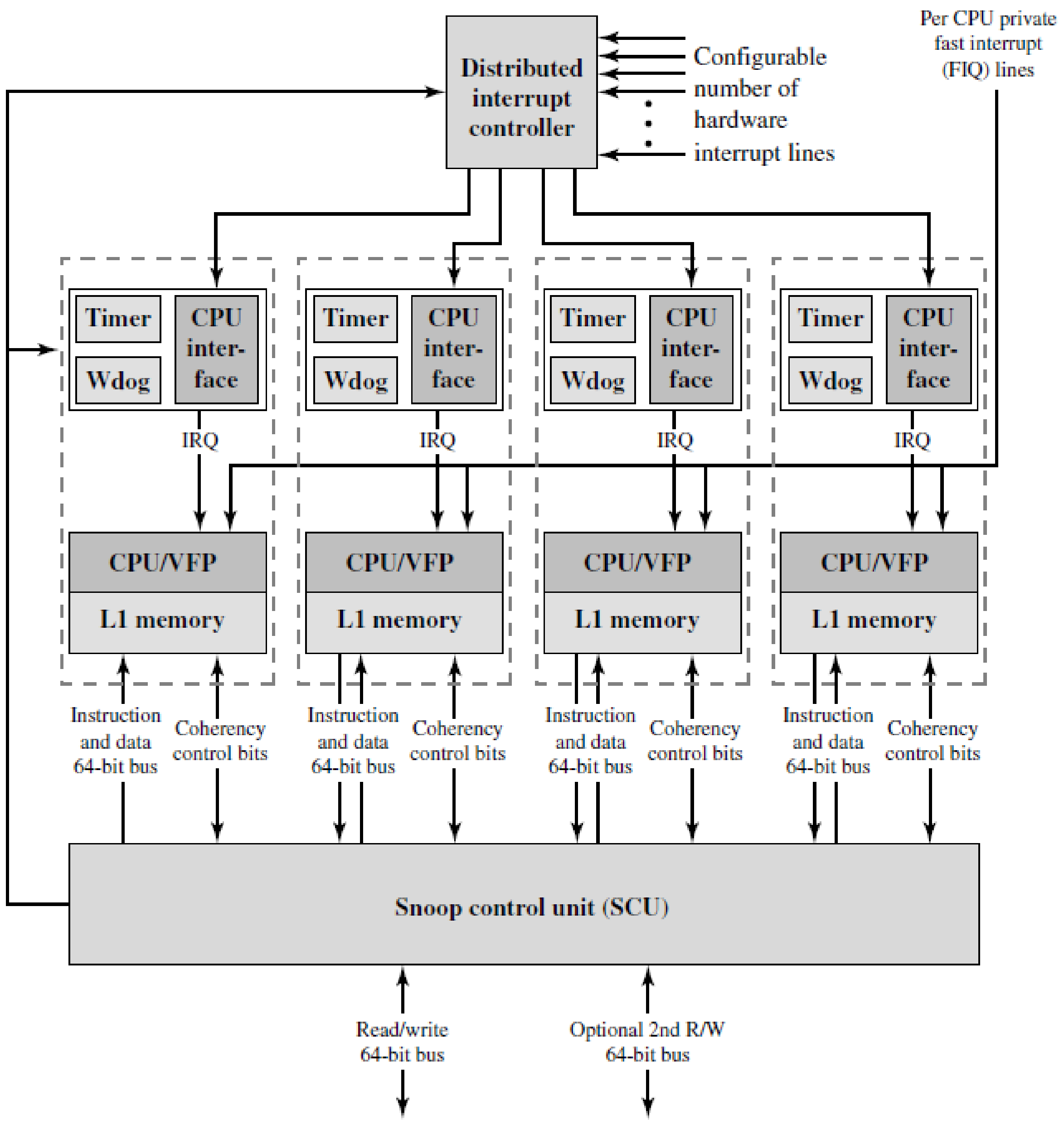

Distributed interrupt controller
Configurable number of hardware interrupt lines
Per CPU private fast interrupt (FIQ) lines
Timer
Wdog
CPU interface
IRQ
CPU/VFP
L1 memory
Instruction and data 64-bit bus
Coherency control bits
Snoop control unit (SCU)
Read/write 64-bit bus
Optional 2nd R/W 64-bit bus

Figure 18.11 ARM11 MPCore Processor Block Diagram