1 Introduction

For the past 20 years we have lived through the information revolution, powered by the explosive growth of semiconductor integration and of the internet. The exponential performance improvement of semiconductor devices was predicted by Moore's law as early as the 1960s. There are several formulations of Moore's law. One of them is directed at the computing power of microprocessors. Moore's law predicts that the computing power of microprocessors will double every 18–24 months at constant cost so that their cost-effectiveness (the ratio between performance and cost) will grow at an exponential rate. It has been observed that the computing power of entire systems also grows at the same pace. This law has endured the test of time and still remains valid today. This law will be tested repeatedly, both now and in the future, as many people see today strong evidence that the "end of the ride" is near, mostly because the miniaturization of CMOS technology is fast reaching its limit, the so-called *CMOS endpoint*.

Besides semiconductor technology, improved chip designs have also fueled the phenomenal performance growth of microprocessors over the years. Historically, with each new process generation, the logic switching speed and the amount of on-chip logic have both increased dramatically. Faster switching speeds lead to higher clock rates. Aggressive chip designs also contribute to higher clock rates by improving the design of circuits or by pipelining the steps in the execution of an instruction. With deeper pipelines, the function performed in each pipeline stage takes fewer gate delays. More importantly, the dramatic increase in the amount of on-chip resources over the years gives the chip architect new opportunities to deploy various techniques to improve throughput, such as exploiting parallelism at all levels of the hardware/software stack. How best to use the ever-increasing wealth of resources provided by technology falls into the realm of computer architecture.

Computer architecture is a relatively young engineering discipline. The academic and research field of computer architecture started in the early 1970s with the birth of the very successful International Conference on Parallel Processing (ICPP) and International Symposium on Computer Architecture (ISCA). Obviously parallel processing was already a major focus of computer architecture at that time. Actually in the 1980s and at the beginning of the 1990s parallel processing and parallel computer architecture were very popular topics among researchers in the field. Academic researchers were promoting scalable parallel systems with millions of slow, cheap processing elements. Then as now, the demise of systems based on a single central processing unit was seen as inevitable and fast approaching. Eventually, industry decided otherwise, and towards the middle of the 1990s parallel systems were eclipsed by the so-called "killer-micro." The years that followed saw an explosion in the speed and

capabilities of microprocessors built with a single CPU. With the unrelenting success of Moore's law, designers can exploit rapidly increasing transistor densities and clock frequencies. The increased transistor count was in the past utilized to design complex single out-of-order processors capable of processing hundreds of instructions in any given cycle. Rather than dealing with the complexity of programming parallel systems, industry embraced complex out-of-order processors with ever-increasing clock speeds because they provided the path of least resistance to fulfill the ever-growing expectations of computer users. In the commercial arena, multiprocessors were merely seen as extensions to uniprocessor systems, offering a range of machines with various cost/performance ratios.

This situation rapidly changed in the early years of the twenty-first century. Technological trends shifted in favor of processors made of multiple CPUs or cores. Issues such as power, complexity, and the growing performance gap between processors and main memory have restored an acute interest in parallel processing and parallel architectures, both in industry and in academia. Nowadays the consensus in the computer architecture community is that all future microarchitectures will have to adopt some form of parallel execution. Generically, this emerging form of microarchitecture is referred to as chip multiprocessors (or CMPs), and is one of the major focal points of this book.

Conceiving the design of a microprocessor, a part of a microprocessor, or an entire computer system is the role of the computer architect. Although Moore's law applies to any device or system, and although many techniques covered in this book are applicable to other types of microchips such as ASICs, this book specifically focuses on instruction processing systems and microprocessors in which the chip or system is designed to execute an instruction set as effectively as possible.

1.1 WHAT IS COMPUTER ARCHITECTURE?

Computer architecture is an engineering or applied science discipline whose focus is the design of better computers, given technology constraints and software demands. In the past, computer architecture was synonymous with the design of instruction sets. However, over time, the term has evolved to encompass the hardware organization of a computer, and the design of a microprocessor or of an entire system down to the hardware component level. In this book we adopt by default the modern definition of "computer architecture" to mean the "hardware organization and design of computers." Whenever we refer to the instruction set we will explicitly use the term "instruction set architecture" or ISA. The design of instruction sets is, at this point of history, quite settled, and only a few instruction sets are still supported by industry. Although there may be additions to current ISAs from time to time, it is extremely unlikely that new instruction sets will again be created from scratch because the cost of developing a brand new instruction set and its implementations is astronomical. In this book, we cover ISAs rather cursorily since our primary target is not ISAs but rather parallel computer architectures that implement an ISA fast and correctly, within cost and technological constraints.

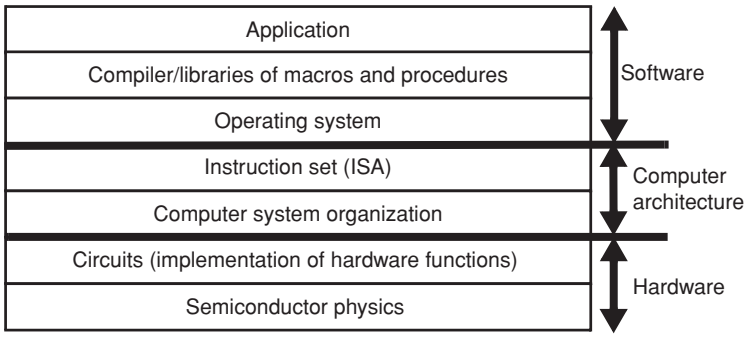

Figure 1.1. Layered view of computer systems.

The design of computer systems is very complex and involves competence in many different engineering and scientific fields. The only way to manage this complexity is to split the design in several layers, so that engineers and scientists in different fields can each focus their competence into a particular layer. Figure 1.1 illustrates the layered view of modern computer systems. Each layer relies on the layer(s) below it. An expert in a particular application field writes application programs in a high-level language such as Fortran, C++, or Java using calls to libraries for complex and common functions at the user level and to the operating system for system functions such as I/O (input/output) and memory management. The compiler compiles the source code to a level that is understandable by the machine (i.e., object or machine code) and the operating system (through operating system calls). Compiler designers just have to focus on parsing high-level language statements, on optimizing the code, and on translating it into assembly or machine code. Object code is linked with software libraries implementing a set of common software functions. The operating system extends the functionality of the hardware by handling complex functions in software and orchestrates the sharing of machine resources among multiple users in a way that is efficient, safe, and transparent to each user. This is the domain of kernel developers. Underneath these complex software layers lies the instruction set architecture, or ISA.

The ISA is a particularly important interface. It separates software from hardware, computer scientists from computer/electrical engineers. The implementation of the ISA is independent of all the software layers above it. The goal of the computer architect is to design a hardware device to implement the instruction set as efficiently as possible, given technological constraints. The computer architect designs at the boundary between hardware and software and must be knowledgeable in both. The computer architect must understand compilers and operating systems and at the same time must be aware of technological constraints and circuit design techniques.

System functions may be implemented in hardware or in software. For example, some types of exceptions, such as translation lookaside buffer misses in virtual memory systems, may be implemented in hardware or in kernel software. Some components of cache coherence may also be implemented in hardware or in software. Using software to implement system functions is a flexible approach to simplifying hardware. On the other hand, software implementations of system functions are usually slower than hardware implementations. Once the hardware

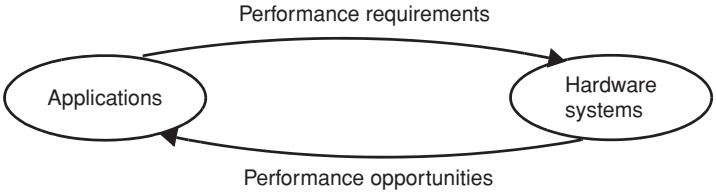

Figure 1.2. Synergy between application growth and hardware performance.

architecture has been specified, its actual implementation is left to circuit engineers, although iterations are possible. Finally the hardware substrate is conceived and developed by process and manufacturing engineers and by material scientists.

By separating hardware layers from software layers, the ISA has historically played a critical role in the dramatic success of the computer industry since its inception. In the 1950s and early 1960s, every new computer was designed with a different instruction set. In fact, the instruction set was the defining hallmark of every computer design. The downside of this strategy was that software was not portable from one machine to the next. At that time compilers did not exist, and all programs were written in assembly code. In 1964 IBM transformed itself into the behemoth computer company we know it to be today by introducing its System/360 ISA. From then on, IBM guaranteed that all its future computers would be capable of running all software written for System/360 because they would support all System/360 instructions forever. This guarantee called *backward compatibility* ensured that all binary codes written or compiled for the IBM System/360 ISA would run on any IBM/360 system forever and software would never again become obsolete. The IBM 360 instruction set might expand in the future – and it did – but it would never drop instructions nor change the semantic or side effects of any instruction. This strategy has endured the test of time, even if most programs today are written in high-level languages and compiled into binaries, because the source code of binaries may be lost and, moreover, software vendors often deliver object code only. Over the years, as it expanded, System/360 was renamed System/370, then System/390, and today is known as System z.

Because instruction sets do not change much over time, the function of the computer architect is to build the best hardware architecture to meet the ever-growing demands of software systems. Figure 1.2 illustrates the synergy between growing software requirements and hardware performance. Users always want more from the hardware (e.g., processing speed, amount of memory, or I/O bandwidth) as their applications grow. On the other hand, as hardware evolves, it exposes new opportunities to software developers, who rapidly take advantage of them. This synergy has worked wonders for Intel and Microsoft over the years.

We are at an important juncture in this self-perpetuating cycle. The current evolution of microarchitectures dictates that software must become more parallel in order to take advantage of new hardware opportunities offered by multi-core microprocessors. Today's technology dictates that the path to higher performance must be through chip multiprocessors (CMPs). The development of effective parallel software is probably the biggest challenge facing future computing systems today, even more so than all the technological challenges. Software must adapt to take advantage of multiprocessor architectures. Parallel programming and the

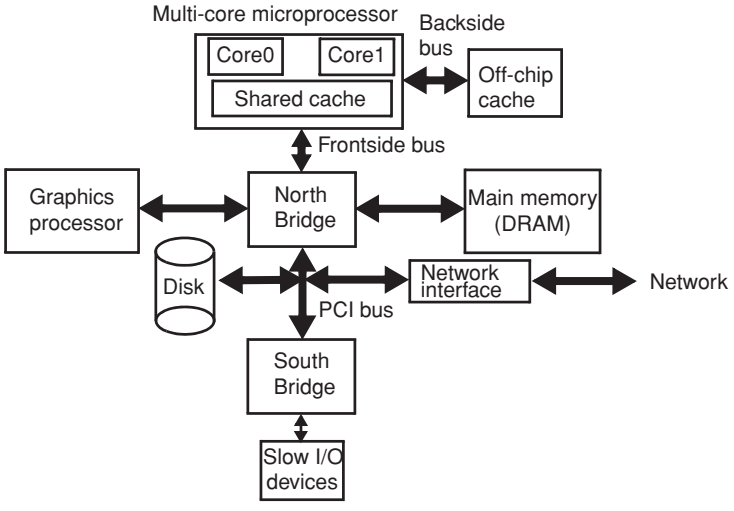

Figure 1.3. Basic PC architecture.

compiling of sequential code into parallel code have been attempted time and again in the past. Unless software can harness and unleash the power of multi-core, multi-threaded systems, the information revolution may come to an end.

Because of technological constraints, hardware cannot sustain the exponential growth of single-threaded performance at the rate envisioned by Moore's law. Future microprocessors will have multiple cores running multiple threads in parallel. In future, single-thread performance will, on average, grow at a more modest pace, and Moore's law as applied to computing power will be met by running more and more threads in parallel in every processor node.

1.2 COMPONENTS OF A PARALLEL ARCHITECTURE

The architecture of a basic personal computer (PC), one of the simplest parallel computers, is shown in Figure 1.3. The North Bridge chip acts as a system bus connecting a (multi-core) processor, main memory, and I/O (input/output) devices. The PCI (Peripheral Component Interconnect) bus is the I/O bus connecting high-speed I/O interfaces to disk, network, and slow I/O devices (such as keyboard, printer, and mouse) to the North Bridge. The South Bridge acts as a bus for low-bandwidth peripheral devices such as printers or keyboards.

A generic high-end parallel architecture is shown in Figure 1.4. Several processor nodes are connected through an interconnection network, which enables the nodes to transmit data between themselves. Each node has a (possibly multi-core) processor (P), a share of the main memory (M), and a cache hierarchy (C). The processor nodes are connected to the global interconnection – a bus or a point-to-point network – through a network interface (NI). Another important component of a computer system is I/O; I/O devices (such as disks) are often connected to an I/O bus, which is interfaced to the memory in each processor node through the interconnect. Processor, memory hierarchy, and interconnection are critical components of a parallel system.

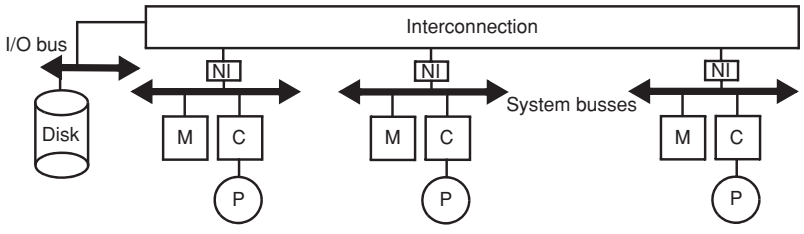

Figure 1.4. Generic multiprocessor system with distributed memory.

1.2.1 Processors

First, in this era of chip multiprocessors and multi-threaded cores, a few basic definitions are in order.

A *program* (sometimes referred to as *code*, *code fragment*, or *code segment*) is a static set of statements written by the programmer to perform the computational steps of an algorithm. A *process* or *thread* is an abstraction which embeds the execution of these computational steps. In some sense, to use a culinary analogy, a program is to a process what a recipe is to cooking. At times the words process and thread are used interchangeably, but usually the management of threads is lighter (has less overhead) than the management of processes. In this book, we will mostly use the word thread.

Threads run on cores or CPUs (central processing units). A core or CPU is a hardware entity capable of sequencing and executing the instructions of a thread. Some cores are *multi-threaded* and can execute more than one thread at the same time. In this case, each thread runs in a *hardware thread context* in the core. Microprocessors or processors are made of one or multiple cores. A multi-core microprocessor is also sometimes called a chip multiprocessor or CMP. A multiprocessor is a set of processors connected together to execute a common workload.

Nowadays, processors are mass-produced, off-the-shelf microprocessors comprising one or several cores and several levels of caches. Moreover, various system functions, such as memory controllers, external cache directory, and network interfaces, may be migrated on-chip in order to facilitate the integration of entire systems with a minimum number of chips.

Several factors affect core performance. The major factor is the clock frequency. Because cores are pipelined, the clock frequency dictates the rate at which instructions are fetched and executed. In the past the performance of microprocessors was mostly dictated by their clock rates. The possible clock rate of a processor is determined by three main factors:

- The technology node. With every new process generation, the switching speed of every transistor increases by 41%, as a direct result of process shrinkage. The impact of this factor on the clock rate will be blunted in future by wire delays because the speed of signal transmission on wires does not scale like transistor switching speed.
- The pipeline depth. With deeper pipelines (i.e., more pipeline stages) the number of gate delays per stage decreases because the function implemented in each stage is less complex. Historically the number of gate delays per pipeline stage has dropped by roughly 25% in

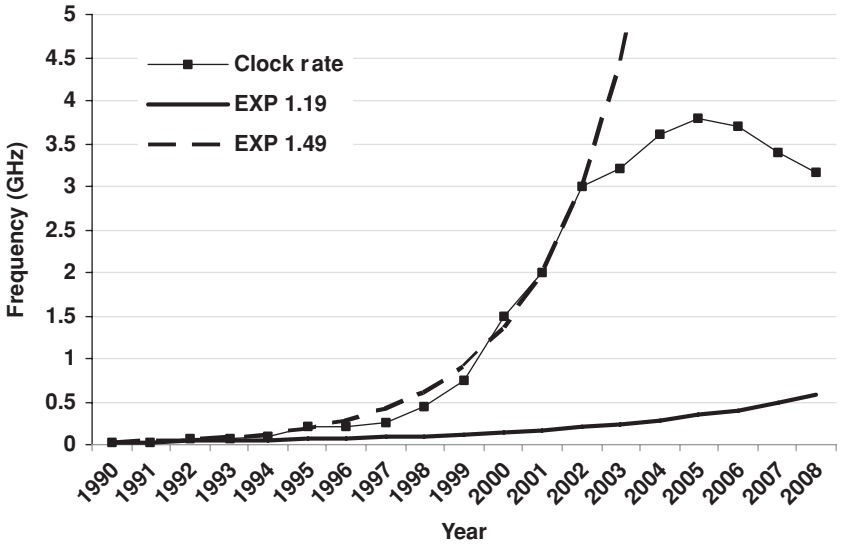

Figure 1.5. Highest clock rate of Intel processors from 1990 to 2008.

every process generation. From now on it will be difficult to increase pipeline depth because it is difficult to implement useful stage functions in fewer than ten gate delays.

- Circuit design. Better circuits are designed to improve the delay of gates and their interconnection.

Figure 1.5 displays the highest clock rate of Intel processors since 1990. The curve for the clock rate is compared to two exponentials, one increasing by 19% per year (doubling every 48 months) and one increasing by 49% per year (doubling every 21 months). The 19% curve shows frequency increases resulting solely from technology scaling (41% per generation every two years). This would be the rate of frequency improvement if the same hardware had been mapped to each new technology over time. From 1990 to 2002, the clock rate grew at a much more rapid rate, doubling in less than two years (the 49% curve). After 2002, clock rate increases started to taper off, and the rates peaked in 2005. Before 2003, clock rates of 10 GHz seemed to be around the corner. At that time some were predicting 10 GHz before 2010. Actually, if the clock rate had stayed on the 49% curve, it would have been more than 30 GHz in 2008! In November 2004 Intel canceled its announced 4 GHz Pentium 4 processor, which had been marred by delays, and changed tack to multi-core microarchitectures. This announcement was perceived as a major turning point in the microprocessor industry at large, a tectonic shift away from muscled uniprocessor pipelined designs to multi-core microarchitectures.

Architecture played a critical role in the large frequency gains observed between 1990 and 2002. These frequency gains were to a large extent due to the advent of very deep pipelines in the Pentium III and Pentium 4 microarchitectures. To sustain pipelines with 10 to 20 stages, vast amounts of parallelism had to be extracted from the instruction stream. Architectural innovations covered in this book, such as branch prediction, register renaming, re-order buffer, lock-up free caches, and memory disambiguation, were key to efficient out-of-order, speculative execution,

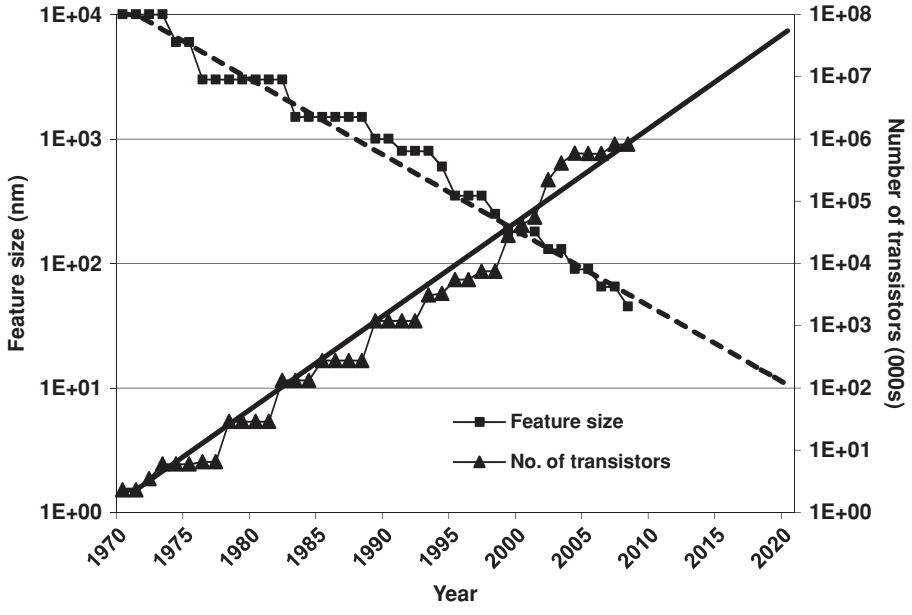

Figure 1.6. Feature size scaling in Intel microprocessors.

and to the exploration of massive amounts of instruction-level parallelism (ILP). Without these innovations, it would have been futile to pipeline the processor deeper.

There is a strong argument that the clock rate gains of the past cannot be sustained in the future, for three reasons. First it will be difficult to build useful pipelines with fewer than ten levels of logic in every stage, a limit we have already reached. Second, wire delays, not transistor switching speeds, will dominate the clock cycle in future technologies. Third, circuits clocked at higher rates consume more power, and we have reached the limits of power consumption in single-chip microprocessors. Figure 1.5 empirically validates this argument: since 2002, the clock rate improvements of microprocessors have mostly stalled.

The contributions of computer architecture go beyond simply sustaining clock rate improvements. Instruction throughput can also be improved by better memory system designs, by improving the efficiency of all parts of the processor, by fetching and decoding multiple instructions per clock, by running multiple threads on the same core (a technique called core multi-threading), or even by running threads on multiple cores at the same time. Besides higher frequencies, each new process generation offers a bounty of new resources (transistors and pins) which can be exploited by the computer architect to improve performance further. An obvious and simple way to exploit this growing real estate is to add more cache space on chip. However, this real estate can also be utilized for other purposes and offers the computer architect new opportunities, a sandbox in which to play so to speak.

Figure 1.6 shows the evolution of feature sizes in Intel technologies from 1971 to 2008 extrapolated to 2020. For the past 20 years a new process generation has occurred every two years, and the feature size has shrunk at the rate of 15% per year, i.e., it is reduced by 30% every generation or halved every five years. Figure 1.6 also shows the maximum number of

Table 1.1 **Cost and size of memories in a basic PC (2008)**

Memory	Size	Marginal cost	Cost per MB	Access time
L2 cache (on chip)	1 MB	$20/MB	$20	5 ns
Main memory	1 GB	$50/GB	5c	200 ns
Disk	500 GB	$100/500 GB	0.02c	5 ms

transistors in Intel microprocessor chips in each year from 1971. This number factors in the increase in transistor density and in die area. The figure shows that the amount of on-chip real estate has doubled every two years; in 2008, one billion transistors was reached. If the trend continues, we will have 100 billion transistors on a chip by 2020. However, let's remember that trends only last until they end, and can only be established in the past, as the frequency trends of the past demonstrate.

Finding ways to exploit 100 billion transistors in the best way possible is one of the biggest challenges of the computer architecture research field in the next ten years. The most probable and promising direction is to implement multiprocessors on a chip, possibly large-scale ones, with hundreds or even thousands of cores.

1.2.2 Memory

The memory system comprises caches, main (primary) memory, and disk (secondary) memory. Any data or instruction directly accessible by the processor must be present in main memory. Perennial problems in computer systems are the speed gaps between main memory (access times in the 100 nanosecond range) and processor (clocked at several gigahertz), and between disk (access time in milliseconds) and processor.

The design of a memory system is dictated by its cost and by physical constraints. Physical constraints are of two types. First, a computer system needs a very large non-volatile memory to store permanent files. Most significant semiconductor memories such as main memory and caches are volatile and their content is lost on power down. This functionality is commonly fulfilled by hard disk drives (HDDs). Although more costly, solid-state disks (SSDs) such as flash memories are often deployed as well in systems. Second, the access time of any type of memory increases with its size. This will be particularly true in future technologies, because access times to semiconductor memories are dominated by wire delays. With larger memories, address decoding, address line (row) propagation, and bit line (column) propagation all take more time. The cost and size of memories at different levels for a basic PC in 2008 are listed in Table 1.1.

The goal of a memory hierarchy is to give the illusion to the processor of a monolithic memory system that has an average memory access time similar to the processor cycle time and, at the same time, has the size of the disk space and a cost per bit close to that of disk memory.

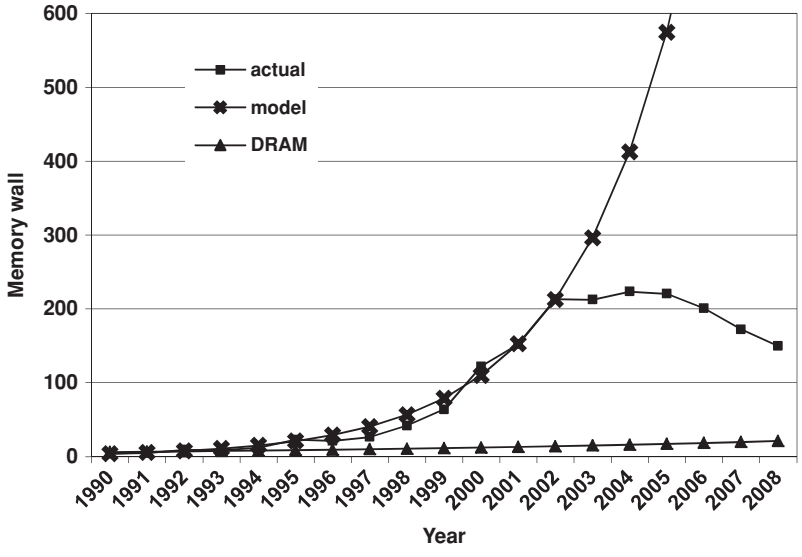

Figure 1.7. Memory wall (DRAM access time/CPU clock time).

Main memory

The speed gap between main memory (built with DRAMs) is large enough that it can affect processor performance. For example, if the processor is clocked at 1 GHz and the main memory access time is 100 ns, more than 100 instructions could be executed while the processor is waiting on an access. A processor, however complex it is and however fast it is clocked, cannot execute instructions faster than the memory system can deliver instructions and data to it.

Historically, the gap between processor cycle time and main memory access time has been growing at an alarming rate, a trend called the *memory wall*. Between higher clock rates and computer architecture innovations, microprocessor speed has historically increased by more than 50% per year. On the other hand, DRAM performance has increased at the much lower rate of about 7% per year. Note that the access time to DRAM includes not only the access time of the DRAM chips themselves, but also delays through the memory bus and controllers.

Figure 1.7 illustrates the memory wall over time. Here the memory wall is defined as the ratio of main memory access time and processor cycle time. In 1990, the Intel i486 was clocked at 25 MHz and access to DRAM was of the order of 150 ns, a factor of 4. Thus the "height" of the memory wall was 4. If processor performance had kept improving at the rate of 49% every year from 1990 on, then the height of the memory wall would have surged by a staggering factor of 400, to 1600 by 2008. However, this obviously was not the case. Rather, processor clock rates peaked while DRAM speed kept improving at a modest pace. Because of this, the actual performance gap between memory and processors is only a factor of 40 larger in 2008 than it was in 1990. Figure 1.7 shows that at around 2002 the memory wall departed from its historical trends to peak and has even dropped since 2003.

Historically, the lackluster performance of DRAM memories has been offset by a cache hierarchy between the processor and main memory and by mechanisms to tolerate large cache