Unit 2: Hardware Background

Martha A. Kim

October 10, 2012

System background and terminology

Here, we briefly review the high-level operation of a computer system, defining terminology to be used down the road. For more background on these topics, see the related reading [1].

[1] David A. Patterson and John L. Hennessy. *Computer Organization and Design: The Hardware / Software Interface*. Morgan-Kauffman, fourth edition, 2009

Hardware

The hardware portion of a system comprises all physical components, including the central processing unit, memory, motherboard, hard drive, graphics chip, network interface cards, USB and other ports, mouse, keyboard, display and more. The many varied types of hardware components combine to perform just three essential functions: inputting and outputting data (i.e., I/O), processing data, and storing data.

Software

Software portion consists of the many programs which direct the actions of the hardware. Software is often classified into *application software* and *system software*. Application software includes any application a computer user might invoke, such as word processor, web browser, databases, email, image editors, and so on. The system software provides the interface between the system hardware and the application software. The operating system is the most well-known piece of system software. The OS manages the computer's hardware resources, making them available to applications via an *application programming interface* or API. However, system software also includes other tools, such as the network stack, compilers, assemblers, and loaders.

All applications consist of one or more processes. A *process* is simply an executing program. From the operating system's perspective, a process consists of an address space, executable code, a unique identifier, environment variables, a priority class, security permissions, and atleast one thread. A *thread* is a list of instructions that can be scheduled (again by the OS) for execution on a processor. Applications (and processes) may be either *single-threaded* (i.e., *sequential*) or *multi-threaded*. Each thread consists of a list of *instructions* which are the basic commands understood by the processor. The instructions in a thread are executed *serially*, one after the other. As we will see in a moment, the processor may, in fact, reorder the instructions slightly, or execute multiple instructions in parallel, but to the world outside the processor, the instructions in a thread will always appear to have been executed in the order they are listed.

Program Execution

Programmers typically write programs in a *high-level language* such as C, C++, Java, or X10. They then invoke a compiler to compile the high-level program description to a particular *instruction set architecture (or ISA)*. The ISA defines the set of instructions that a processor can execute. In defining how software and hardware communicate, an ISA forms the interface between hardware and software. This is a pivotal role in a computer system, and, as we will see later, changes to this interface can have significant repercussions. The instruction-level program representation is then encoded into a binary format. This last step is called program *assembly* and is accomplished using an *assembler*. The result of assembly is a program *binary* or *executable*. The executable is stored, typically on a hard disk, until it is time to execute it. At that point the operating system loads the program into memory memory (this step is known as *loading*) and points the processor to the first instruction.

Processor Architectures and Microarchitectures

The processor, or central processing unit (CPU), is is responsible for executing a program's instructions, thereby affecting the behavior of the program. The processor reads each instruction from memory and carries out the specified operation. This is the most basic operational loop of a processor: FETCH instruction, EXECUTE it, repeat with the next instruction. A processor's *architecture* consists of aspects of a processor's operation which are exposed to software via the ISA. Architectural resources often include instructions, registers, and memory addressing modes. For more detail on this area please refer to the related text [2].

[2] David A. Patterson and John L. Hennessy. *Computer Organization and Design: The Hardware / Software Interface*. Morgan-Kauffman, fourth edition, 2009

By contrast, a processor's *microarchitecture* describes a particular implementation of an architecture. For example, Intel and AMD each design, build, and sell different processors, yet they both support the X86 ISA. The differences between Intel and AMD chips lie in their microarchitectures. With the ISA providing compatability, each corporation's microarchitectural optimizations are applied below the ISA and are thus functionally transparent to the software. Though they do not affect a program's function, they affect almost every other aspect of a system, including its performance, power consumption, and reliability.

ISAs in the Wild

By far the most common ISA for desktop computers is Intel's X86 [3] which is supported by the vast majority of the desktop market including all of Intel's and AMD's general purpose processors. Other common ISAs include IBM's PowerPC [4] which was the the core of Apple's computers for many years, and Sun's UltraSPARC in the server market. Broadening our view to embedded processors we find ARM's XScale ISA, implemented by ARM in iPhones [5], and by Intel in the more and recent BlackBerries [6].

[3] Intel Corporation. Intel 64 and IA-32 architectures software developer's manuals, b. URL `http://www.intel.com/products/processor/manuals`

[4] IBM Corporation. Power architecture offerings, a. URL `http://www-03.ibm.com/technology/power/powerpc.html`

[5] Anand Lal Shimpi. Apple's iphone dissected: We did it, so you don't have to. URL `http://www.anandtech.com/printarticle.aspx?i=3026`

[6] ZDNet. Blackberry 9000 will have faster processor than current iphone. URL `http://blogs.zdnet.com/blackberry/?p=472`

Technology Trends and Moore's Law

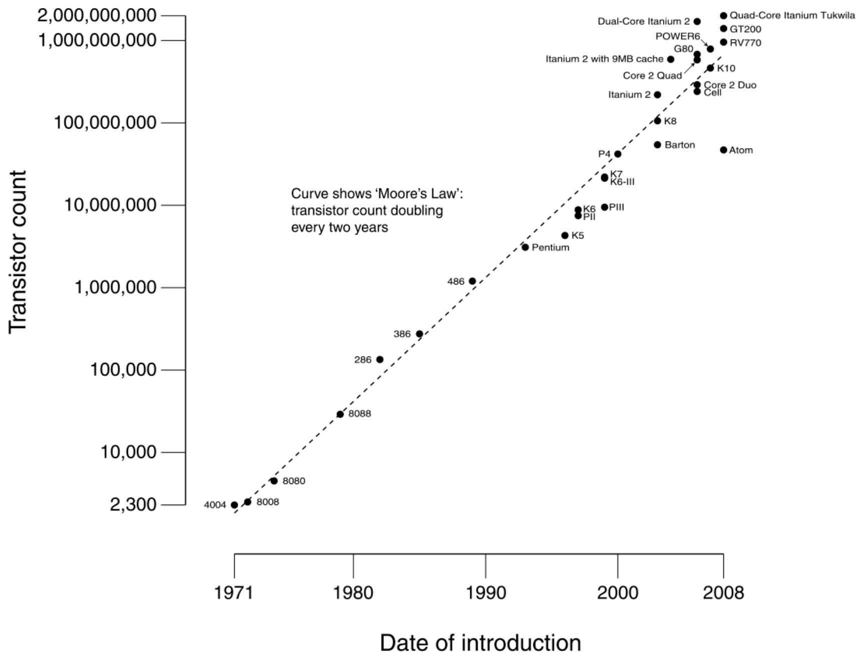

Figure 1: Rising transistor counts have tracked Gordon Moore's observation that they doubled every two years.

Strictly speaking, Moore's Law is not a law at all. Instead, this "law" describes the empirical trend that transistors have shrunk at an exponential rate for nearly half a century. The pattern was first observed by Gordon Moore, co-founder of Intel, in 1965 [7]. Specifically, Moore observed that the number of transistors that could be produced at a fixed cost (largely determined by silicon area) was increasing exponentially, doubling roughly every two years. Moore's law is often mis-stated to say that processor speed doubles every two years. However, the law speaks only to the size of the transistors as

[7] Gordon E Moore. *Cramming More Components onto Integrated Circuits.* 1965. URL `ftp://download.intel.com/research/silicon/moorespaper.pdf`

indicated in Figure 1.

Moore's has driven many correlated *technology* trends, including:

- Smaller transistors can switch more rapidly between logic 1 and logic 0, thereby allowing faster clocking of the logic (think bigger Hz). The steady increases in clock speed is known as *clock scaling*.
- Smaller transistors are more densely packed resulting in greater power consumption (and head production) per unit area. This is what is known as *power density*.
- Storage density has also scaled, increasing memory capacities at nearly the raw transistor growth rate.
- Chips have shrunk. A fixed design will, every two years, require roughly half the amount of silicon it used to.
- Smaller transistors are more prone to manufacturing defects and to accidental bit flips during operation (the latter are often known as *soft errors* or *transient faults*).
- Due to the complexity of designing a billion-transistor chip, design teams and design times have grown. Processors are often in the pipe for five years (or more) before they are released.

Some Words on Power Consumption

The power consumption of a circuit is spent in two parts:

First, there is the switching, or dynamic, power. This is the power required to switch the transistors open and closed while carrying out a computation. A single cycle requires one charge and discharge of a capacitor, making the energy of this operation $E_{switch} = C \times V_{dd}^2$. Let f be the frequency at which the circuit operates. The switching power simply the energy of the switching operation times the frequency with which it occurs: $P_{switching} = E_{switch} \times f = C \times V_{dd}^2 \times f$. What this means is that slower circuits (i.e., those operating at lower frequencies) use less power, but not less energy to perform the same computation. By contrast, the energy is independent of operating frequency, and this often serves as a quality measure of a logic family.

The second component of power consumption is the leakage, or static, power. When a circuit is powered on, even if it is not actively switching, the transistors leak power from their gates to their sources. This leakage current (I) gives rise to additional power consumption $P_{leakage} = V_{dd} \times I$.

Thus, the total power consumption of a circuit can be modeled as follows: $P = P_{switching} + P_{leakage} = C \times V_{dd}^2 \times f + V_{dd} \times I$.

A Processor Retrospective

1978-1986: Riding Moore's coattails

Prior to the mid 1980s, the growth in processor performance, as shown in Figure 2 (Source: [8]) was primarily technology driven. Transistors shrunk, clock speeds increased, enabling the resulting processor circuits to carry out computation faster and faster.

[8] David A. Patterson and John L. Hennessy. *Computer Organization and Design: The Hardware / Software Interface*. Morgan-Kauffman, fourth edition, 2009

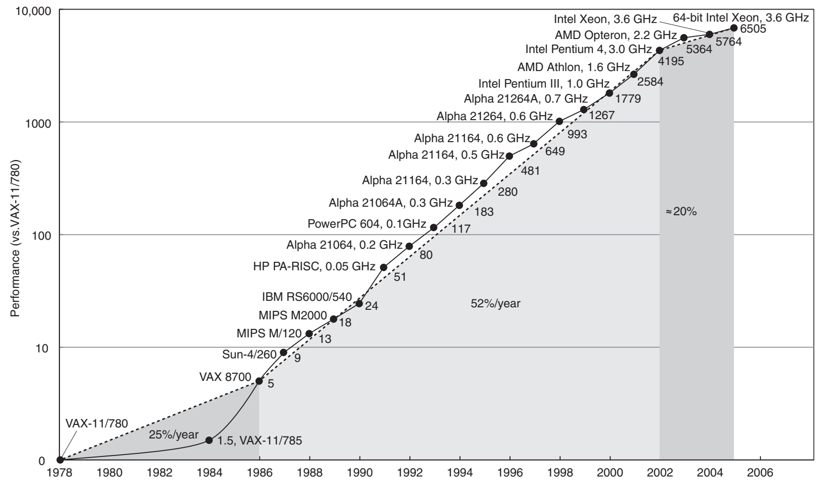

Figure 2: Historical single-threaded processor performance.

1986-2002: What Andy gave, Bill took

As Figure 2 indicates, single processor performance improved at a steady rate of 52% per year for 16 years [9]. These performance improvements were the result of both technology trends and microarchitectural improvements. On the technological side, clock scaling was in full force. On the microarchitectural side, processor designers were spending the abundance of transistors exploiting instruction-level parallelism (ILP) extraction techniques such as superscalar processing, out-out-of-order execution, and deeper pipelines [10].

[9] John L. Hennessy and David A. Patterson. *Computer Architecture: A Quantitative Approach*. Morgan-Kauffman, fourth edition, 2008

[10] John L. Hennessy and David A. Patterson. *Computer Architecture: A Quantitative Approach*. Morgan-Kauffman, fourth edition, 2008

The performance gains shown in Figure 2 are attributable largely to faster clocks and microarchitectural enhancements. Because these improvements all occurred *below* the ISA, software and software developers got a free ride. They were able to reap the performance gains without changing their codes or their programming model. Moreover, procrastination by the software community was rewarded. With sequential performance improving at approximately 2x every 1.5 years, waiting 1.5 years, making absolutely no change to the soft-

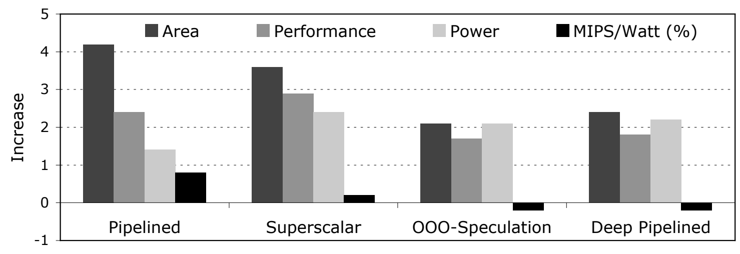

Figure 3: Area, performance, power, and efficiency effects of common ILP-extraction techniques. Source: Shekhar Borkar, Intel.

ware would still be rewarded with improved performance. These "dividends of Moore's Law" [11] were often spent adding additional features and enhancements to programs.

[11] James R. Larus. Spending mooreâĂŹs dividend. *Communications of the ACM*, May 2008. URL Freelyavailabletechnicalreport: http://research.microsoft.com/pubs/70581/tr-2008-69.pdf

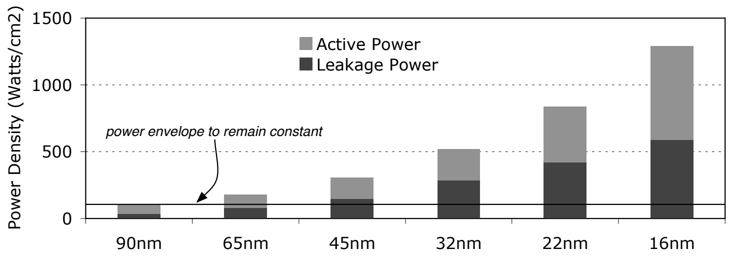

Figure 4: Power consumption for current and future process nodes. Source: Shekhar Borkar, Intel.

So, what brought this heyday to a close? Power. The power consumption of these chips, and the attendant heat production, had reached the an untenable limit. During this same period, while raw performance was improving dramatically, power efficiency was diminishing. Figure 3 shows the performance benefits and area and power costs of the principle ILP-extraction techniques applied during this era. While these techniques improved performance, they did little to help (and occasionally hurt) power efficiency.

Lower power efficiency implies that increased power consumption is necessary to meet a fixed performance goal. Cooling concerns, however, impose a strict limit on how much power a chip may consume. This is what is known as a "power envelope". This limit has stayed, and is expected to remain, relatively fixed, resulting in the situation shown in Figure 4 where the projected power requirements of processors far outstrip our ability to cool them. Something had to change.

2002-present: After the power wall

As a result of this power crunch, the processor industry has bet their future on *multicores*. Also sometimes called *chip multiprocessors (or CMPs)*, multicores contain multiple processors per chip. In 2004 Intel

Manufacturer Year	IBM 2004	AMD 2005	Sun 2005	Intel 2006
Processors / chip	2	2	8	2
Threads / processor	2	1	4	2
Threads / chip	4	2	32	4

Table 1: The first CMPs hit the market.

cancelled two processor designs that they had been working on. At the time Intel President, Paul Otellini stated "We are dedicating all our future product development to multicore designs, ... This is a sea change in computing." Intel was not alone. Other major processor manufacturers too began heading in the direction of multicore, as shown in Table 1.

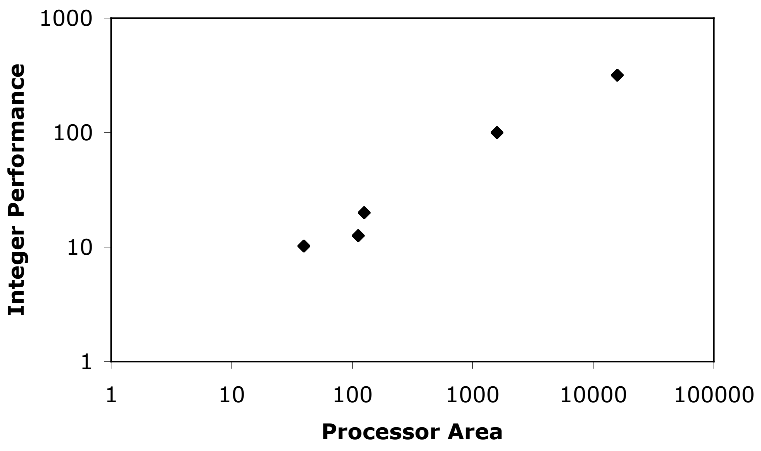

Figure 5: mpirical data confirming Pollack's rule, that processor performance grows with the square root of area.

To appreciate the reasons behind the industry-wide shift, one needs to appreciate Pollack's Rule. Pollack's rule states that processor performance improves with the square root of the area budget. I.e., doubling the size of a microprocessor will make it 40% faster. Like Moore's law, this is based on empirical observations, such as the performance v. area data plotted for some leading microprocessors in Figure 5 (Source: [12]). This rule of thumb describes the diminishing returns and power inefficiencies seen in the years leading up to this shift.

[12] Shekhar Borkar. Thousand core chips — a technology perspective. In *Proceedings of the Design Automation Conference*, 2007. URL `http://videos.dac.com/44th/papers/42_1.pdf`

Because Moore's law was still very much in effect, the challenge was how to tranlate those transistors into performance per watt. Here is why multicores have such potential in this space. Figure 6 (left) shows a basic processor. This design has unit area, power and performance. Soon, technology scaling (i.e., Moore's law) will make it possibe to manufacture the very same chip in one fourth the area. This, new chip, Figure 6 (center), will have 1/4th the area, 1/4th the power consumption (using the approximation that power is proportional to area), and one half the performance (per Pollack's law, the performance is proportional to the square root of the area). If instead we use that area to build a quad-core, Figure 6 (right), we will have produced a design with the same area and power consumption as the older chip, but with double the performance. Or more precisely, double the perfomance potential. Realizing this potential requires that the software actually use both cores.

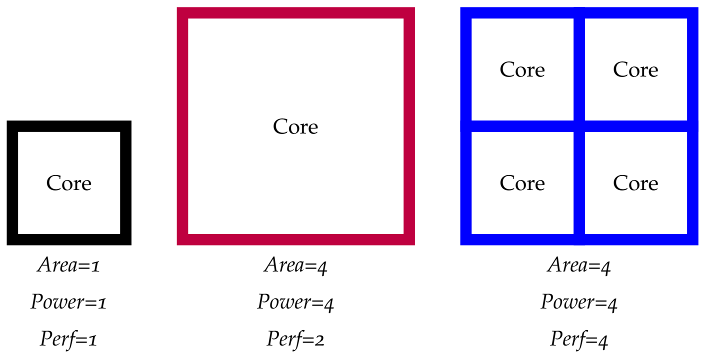

Figure 6: Spending Moore's law on multiple cores. Assume today's core (black) has unit power, area and performance. Quadrupling the transistor budget offers two options: scaling a single core (purple) or replicating the base core (blue). Because power scales with area, but performance with the square root of area, the multicore option (blue) offers a more power efficient option.

Why a sea change?

To continue to see software performance inprovements at the historical rates, software must now change. Instead of single processor performance, what is now scaling is the *number* of processors. Existing sequential codes will no longer automatically run faster, and newly-written parallel applications must be able to harness increasing numbers of parallel cores.

This constitutes an enormous software shakeup from below. To reap performance, one must parallelize as much of an application's execution as possible. A recent article by Marty and Hill [13] explores the relationship between performance and the amount of parallelism that is exposed. The boundary between hardware and software is significantly less crisp than it once was. What is crystal clear is that software must be significantly more hardware-conscious than it has been in the past, and that hardware is newly dependent on software to provide gains in performance.

[13] Mark D. Hill and Michael R. Marty. Amdahl's law in the multicore era. *IEEE Computer*, July 2008. URL `http://www.cs.wisc.edu/multifacet/papers/ieeecomputer08_amdahl_multicore.pdf`

How far will multicores scale?

In theory, multicore scaling can continue as long as Moore's law continues to hold. Take the simple RISC II core from 1983. This was a 32-bit, 5 stage pipeline design with 40,760 transistors, which required 60 mm^2 of silicon in a 300nm process, and could be clocked at 3MHz. This processor requires just 0.02 mm^2 in today's 65nm process. Including a floating point unit, instruction and data caches, one could still fit 2000 such cores on a 60 mm^2 chip today. However, it is not likely to continue unchecked as analytical studies indicate that there will be diminishing returns beyond a certain point [14].

[14] Mark D. Hill and Michael R. Marty. Amdahl's law in the multicore era. *IEEE Computer*, July 2008. URL `http://www.cs.wisc.edu/multifacet/papers/ieeecomputer08_amdahl_multicore.pdf`

References

Shekhar Borkar. Thousand core chips — a technology perspective. In *Proceedings of the Design Automation Conference*, 2007. URL `http://videos.dac.com/44th/papers/42_1.pdf`.

IBM Corporation. Power architecture offerings, a. URL `http://www-03.ibm.com/technology/power/powerpc.html`.

Intel Corporation. Intel 64 and IA-32 architectures software developer's manuals, b. URL `http://www.intel.com/products/processor/manuals`.

John L. Hennessy and David A. Patterson. *Computer Architecture: A Quantitative Approach*. Morgan-Kauffman, fourth edition, 2008.

Mark D. Hill and Michael R. Marty. Amdahl's law in the multicore era. *IEEE Computer*, July 2008. URL `http://www.cs.wisc.edu/multifacet/papers/ieeecomputer08_amdahl_multicore.pdf`.

James R. Larus. Spending mooreâĂŹs dividend. *Communications of the ACM*, May 2008. URL `Freelyavailabletechnicalreport:http://research.microsoft.com/pubs/70581/tr-2008-69.pdf`.

Gordon E Moore. *Cramming More Components onto Integrated Circuits*. 1965. URL `ftp://download.intel.com/research/silicon/moorespaper.pdf`.

David A. Patterson and John L. Hennessy. *Computer Organization and Design: The Hardware / Software Interface*. Morgan-Kauffman, fourth edition, 2009.

Anand Lal Shimpi. Apple's iphone dissected: We did it, so you don't have to. URL `http://www.anandtech.com/printarticle.aspx?i=3026`.

ZDNet. Blackberry 9000 will have faster processor than current iphone. URL http://blogs.zdnet.com/blackberry/?p=472.