Software-Hardware Evolution and birth of Multicore Processors

K.R. Chowdhary, Professor
Dept. of Computer Science and Engineering,
JNV University, Jodhpur.
Email: kr.chowdhary@acm.org

Abstract

This paper presents a brief journey to the evolution of computer hardware and software, and underlines that shift to multicore technology is a natural part of the evolution, and highlights the various laws governing the advancement of computer industry. Looking to these, it appears that the HW-SW industry trend can be represented by a mathematical model, for which future developments are predictable. Finally, the paper establishes that the future of computer industry lies in more thrust in software to exploit parallelism available in software to utilize the heterogeneity in multicore processors.

Keywords: Multicore, Moore’s law, Amdahl’s law, Myhrvold’s law, SW-HW heterogeneity.

1 Introduction

The computer performance has been driven by largely decreasing the size of the chips while increasing the number of transistors they contain. In accordance with Moore’s law this has caused the chipmakers to rise and prices to drop. This ongoing trend has driven the computer industry for years [1].

However, the transistors cannot shrink forever. As transistors components grow thinner, chip manufacturers find it difficult to cap power usage and heat generation. Resulting to this, manufacturers have started building chips with multiple cores, each separately cooled, instead of one increasingly powerful core. The multiple core chips do not necessarily run as fast as the highest performing single-core models, but they improve over all performance by handling more work in parallel.

The past progress was inefficient in terms of transistors and and power (such as multiple instruction issue, deep pipeline, out-of-order execution, speculative execution, and prefetching), but that increased performance while preserving the sequential programming model.

Multicore chips are biggest change in PC programming since 32-bit 386 architecture was introduced in 1990s. The multicore are a way to extend the Moore’s law so that the user gets more performance out of a piece of silicon. The multicore chips have been introduced by AMD, IBM, Intel and Sun micro in servers, desktops, and laptops.

1.1 Driving multicore

Current transistor technology limits the ability to continue making single processor cores more powerful. For example, as transistor gets smaller, the gate which switches the electric current due to electrons - off and on, gets thinner and less able to block the flow of electrons (figure-1). Thus small transistors tend to use electricity all the time, even when they are not switching. This wastes and dissipates the power. Also increasing clock speeds causes transistors to switch faster and thus consume more power and generate more heat.

As per Moore’s law, the clock rates would have exceeded 10 GHz in 2008, and 15 GHz in 2010, if there were no switch over to multicore by chip manufacturers in 2005. Due to this shifting to multicore multicore, the clock rate has been restrained below 3 GHz in 2010, which is less than that of single core processors in 2004, helping substantially in
reduction of power dissipation and heating.

1.2 Inside multicore

Consider a dual core chip running multiple applications against a single core. Each core in multicore chip includes everything a multiprocessor has, except level-2 cache memory hierarchy, which is outside the multicore.

In a multicore system, compiler handles the scheduling of instructions in a program, while operating system (OS) controls the overall assignment of tasks. The OS or multithreaded application parcels out the work to multicores. Generally, when a multicore processor has completed a task, one core takes the completed data from the other cores and assembles the final result.

1.3 Applications

To take advantage of multicore chips, vendors must redesign applications. So that processor can run them as multiple threads. The programmers must find good places to breakup the applications, divide the work into roughly equal pieces that can run at the same time.

The vendors must redesign applications so that they can recognize each core’s speed and memory access capabilities as well as how fast cores can communicate with one another.

The typical multicore chips are: AMD, IBM: (Power p5, p6, p7), Intel: Xeon, Sun: Niagara processor - 8 core (has shared II level cache).

The multicore processors will find a natural home in servers, but would not be very useful in desktops until vendors develop considerable more multithreaded applications[2].

Table 1: Performance and clock rate increase in Intel processors

| Year | Processor  | Transistors | Clock f. |
|------|------------|-------------|----------|
| 1978 | 8086       | $29 \times 10^3$ | 5.00Mhz  |
| 2006 | Intel core2d | $291 \times 10^6$ | 2.93GHz  |

If Moore’s law continues to apply, the number of cores in a chip will keep on increasing. The server applications primarily focus on throughput per cost and power. The multicore targeted for these applications use large number of small low power cores. However, desktop users are interested in the performance of a single application at a time. The multicore for desktop users are more likely to have smaller number of larger, higher power cores with better single thread performance. Thus, general solution is heterogeneous chip microprocessor with both high and low complexity cores[3].

1.4 Advantages

There is a speed gain when multiple tasks are run on multicore, compared to on the single-core CPU. Because cores are on the same die, they can share architectural components such as memory elements and memory management. They thus have fewer components and lower costs than systems running multiple chips. Also, signaling between cores can be faster and use less electricity than multichip systems.

2 Hardware Innovations and Moore’s Law

In 1965, Gorden Moore, a co-founder of Intel, postulated that number of transistors that could be fabricated on a semiconductor chip would double every year. Amazingly, this forecast still holds. Each next generation of transistor is smaller and switches at faster speed, allowing clock speed and computer performance to increase at a similar rate (figure 2, table 1). The figure shows that number of transistors increase at the rate predicted by Moore’s law, but clock frequency grew at slightly a different rate[5].
Faster processors governed by Moore’s law enabled software vendors to add new features and functionality to software that in turn demanded larger developer teams. The challenges of constructing increasingly complex software increased demand for high level languages and program libraries. Their higher level of abstraction contributed to slower code and, in conjunction with larger and more complex programs, drove demand for faster processors and closed the cycle, as indicated in figure 3.

Now the era of steady growth of single processor performance is over, and there is a transition from sequential to parallel computation (multicore). The sequential computing era ended when the practical limits on power dissipation stopped the continual increase in clock speed, and lack of exploitation in instruction level parallelism diminished the values of complex architectures.

However, the Moore’s law is still applicable, as semiconductor technology still doubles the number of transistors on chip every two years[6]. This doubling of transistors is now used to increase the number of independent processors on chip, instead of attempting to enhance the capability of individual processor. Now the challenge is - how to design software to exhibit the capability of multicore architecture. In other words, will parallelism continue the cycle of software innovation?

3 Software Innovations and Myhrvold’s Law

A common belief among software developers is that software grows at least at the same rate as the platform on which it runs. Nathan Myhrvolds, former chief technology officer at Microsoft, memorably captured this wisdom with his four laws about software[7].

1. Software is like a gas, which has tendency to expand to fill the capacity of any computer.
   - Windows NT: LOC (lines of code) doubling time 866 days (@ 33.90 percent per year).
   - Browser: LOC doubling time 216 days, growth rate 221 percent per year.

2. Software grows until it comes to limitation by Moore’s law.
   The initial growth is quick, like gas expanding (as in browser), eventually limited by hardware (as in NT), which ultimately brings any processor to its knees, just before the new model is out.

3. Software growth makes Moore’s law possible.
   That is why chips get faster at the same price, not cheaper. This will continue as long as there is opportunity for new software.

4. Impossible to have enough new algorithms, new applications, and new notion of what is cool.

In fact, increase in size of SW has not only overloaded the processor and memory, but have many positive effects also. Some of these are
1. **Increased functionality**
   - Improved security
   - Printer and IO drivers for graphics
   - Printer resolution and color depth (24 bits)
   - Improved software engineering practices such as layering of SW architecture and modularizing system to improve development
   - The data manipulated by computers also evolved from simple ASCII to larger structured objects (word and excel documents), to compressed documents like jpeg, xml (computation and space saving)
   - Growing use of videos

2. **Programming changes**
   Over the last 30 years, programming languages have evolved from Assembly language and C code to high level languages. Major step was C++, which brought object oriented mechanism. C++ also introduced abstraction, like classes and template, and made possible rich libraries. These features give expensive run time implementations, modularity in development, and information hiding. These practices enabled the creation of ever larger and more complex software.

   Safe and managed languages such as C♯ and Java further increased the level of programming by introducing garbage collection, richer class libraries (.net, java classes), just-in-time compilation, etc. All these features provide powerful abstractions for developing software but consume lot of processor resources and memory.

   The rich features of language requires a run time system to maintain a large amount of meta-data on every method and class at run time, even if the required features are not invoked. The table 2 shows the code for compiling “Hello World” in Windows Vista, coded using Visual studio. The table 3 shows the execution time of “Hello World”.

| Language | Size in bytes |
|----------|---------------|
| C        | 5874          |
| C++      | 8762          |

| Mechanism         | Timer (280 nsec.) |
|-------------------|-------------------|
| C++, Console      | 1760              |
| C++, Windows      | 36375             |
| C♯, console       | 2628              |
| C♯, windows       | 80348             |

3. **Decreased Programming Focus**

Abundant machine resources have allowed programmers to become easy-going about performance and less aware about resource consumption by their code. More important is change in the mindset of the developer. Consequently, most codes run at near machine capacity, and any further increase in code size / complexity will retard the performance.

4. **Mitigating Amdahls’s Law**

The Amdahl’s law states that speed of a particular application is limited by the fraction of the application that is serial (cannot be parallelized). During the serial portion of execution, the chip’s power budget is applied towards using a single large core to allow the serial portion to execute as quickly as possible. During the parallel portions, the chip’s power budget is used more efficiently by running the parallel portions on large number of small-area and power efficient cores. Thus, executing serial portions of an application on a fast but relatively inefficient core and executing parallel portions of an algorithm on many small cores can maximize the ratio of performance to power dissipation.

**Illustrative Example:** Let for 10 percent of time a program gets no speedup on a 100-core computer. To run this sequential piece twice as fast, assume a single fat core would need 10 times as many resources as a thin core due to larger caches, a vector
unit, and other features. Applying Amdahl’s law, the speedups can be computed as follows[8]:

i True serial: 1 sec.

ii For 100 thin cores:
Required time = 0.1 + 0.9/100 = 0.109 sec.
Relative speedup = 1/0.109 = 9.17 times

iii For 90 thin cores, and one thick core:
Required time = 0.1/2 + 0.9/90 = 0.06 sec.
Relative speedup = 1/0.06 = 16.66 times

Since, the implementation cost of one thick core is taken as equal to 10 thin cores, the total cost in (ii) and (iii) is going to be the same.

**Requirement of Heterogeneity in Software:**
To take full advantage of heterogeneous Chip Multiprocessors (CMPs), the system software must use the execution characteristics of each application to predict its future processing needs and then schedule it to a core that matches those needs if one is available. The predictions can minimize the loss to the whole system rather than to the single application.

Effective schedulers can be implemented, even with the current commercial operating systems, and open OS like Linux.

To achieve the best performance it will require to compile the program for heterogeneous CMPs slightly differently. That is, for statically schedulable and dynamically schedulable cores. Programming or compiling parallel applications might require more awareness of heterogeneity. Application developers assume that cores provide equal performance, but heterogeneity breaks this assumption. In fact, many of the future areas of research remain for heterogeneous CMPs.[4]

5 Multicore and its future

With clock speeds stalling out, and computational power increasing due to doubling of number of cores per processor, serial computing is now dead, and parallel computing revolution is now upon us. However, writing parallel applications is a significant challenge. For parallelism to succeed it must ultimately produce better performance relative to speed, efficiency, and reliability. However, most programmors are not only ill equipped to produce proper parallel programs, they also lack the tools, and environments for producing such programs.[9]

Dealing with those issues require a suite of tools and environments that provide users and developers with convenient mechanisms for managing different resources in multicore environments. The resources are memory, cache, and computing units, compilers that allow sequential programs to automatically take advantages of multicore systems, strategies that allow users to analyze performance issues in multicore systems, and environment to control and eliminate bugs in parallel threaded programs.

Developing parallel algorithms are considerable challenge, but many problems, such as video processing, natural language interactions, speech recognition, linear and nonlinear optimization, machine learning, are some of the areas, which are computationally intensive, and multicore are the solutions for these. However, Amdahl’s law limit the scope until sequential component itself is rescheduled.

An alternative use for multicore processor is to redesign a sequential application into a loosely coupled or asynchronous system in which computations run on separate processors. For example, it is natural to separate monitoring features from program logic.

It is fact that, for many applications, most functionality is likely to remain sequential. For software developers to find the resources to add or change features, it may be necessary to eliminate old features or reduce their resource consumption. An important consequence of multicore is that sequential performance tuning and code-restructuring tools are likely to be increasingly important.

Another challenge is modifications and updating of parallel code. Parallelism will also force major change in software development. Moore’s dividend enabled a shift to higher level languages and libraries, the pressure driving this trend will not change, because increasing abstraction improves security reliability, and program productivity. In multicore, parallel garbage collection does not increase pause time of threads, hence real-time applications are easier to implement with real-time constraints.

Another approach that sacrifices the performance for development productivity is, to hide the
underlying parallel implementation.

6 Tools for Multicore programming

Cetus [10] [11] is an open source source-to-source C compiler written in java and maintained at Purdue University. The Cetus has features of automatic parallelization of C code. The following part shows the transformation of C code by Cetus.

#Input to Cetus

```c
int temp;
int main(void) {
    int i,j, c, a[100];
    c = 2;
    for (i=0; i<100; i++){
        a[i] = c*a[i] + (a[i] -1);
    }
}
```

#Output from Cetus

```c
int temp;
int main(void) {
    int i,j, c, a[100];
    c = 2;
    for (i=0; i<100; i++){
        a[i] = ((c*a[i]) + (a[i] -1));
    }
}
```

Cetus performs symbolic transformations for efficient source code generation and the one which can be parallelized easily. Some examples are:

- $1 + 2^a + 4 - a \Rightarrow 5 + a$ (folding)
- $a^b + c \Rightarrow a^b + a^c$ (distribution)
- $(a^2)/(8^c) \Rightarrow a / (4^c)$ (division)

7 Multicore programming languages

A parallel programming language must have key properties. First, expressing parallelism should be simple. Second, should be able to combine number of parallel programming models. Third, the languages must exploit parallel resources efficiently.

Cilk, an extension of C provides two key words for expressing parallelism. A `spawn` transforms sequential (blocking) function call into an asynchronous (non blocking) call. A `sync` blocks a function’s execution until all its spawned children are completed.

```c
Cilk int fib (n) {
    if ( n < 2 ) return n;
    else {
        int x, y;
        x = spawn fib (n -1);
        y = spawn fib (n -2);
        sync;
        return ( x + y);
    }
}
```

Because the recursive calls to `fib` are spawned, Cilk’s runtime system can execute them in parallel. Because the expression $(x + y)$ depends on the results of both calls, the `sync` ensures that both have completed before the addition begins.

The Cilk’s run time system must efficiently map logically independent calls onto computational cores.

8 Conclusion

The multicore processors will change the software profoundly as previous hardware evolutions such as the shift from vacuum tubes to transistors or transistors to ICs. Parallelism will drive software in new direction which is computational intensive.

References

[1] David Geer, “Chip makers turn to Multicore Processors”, *Computer*, May 2005, pp. 11-14.

[2] Terraya A., Tenhunen H. and Wolf W., “Multiprocessor Systems on Chips”, *Computer*, June 2005, pp. 36-40.

[3] Rakesh Kumar, et al., “Heterogeneous chip multiprocessors”, *Computer*, Nov. 2005, pp. 32-38.
[4] Stroze L. and Brooks D., “Energy and Area-Efficient Architectures through Application Clustering and Architectural Heterogeneity”, *ACM Transaction son Architecture and Code Optimization*, Vol. 6, No. 1, March 2009, pp. 4.1-4.31.

[5] James Larus, “Spending Moore’s dividend”, *Communications of ACM*, Vol. 52, May 2009, pp.62-69.

[6] Hachman M., “Intel’s Gelsinger Predicts IntelInside Every Thing”, *PC Magazine*, July 3, 2008.

[7] research.microsoft.com/acm97/nm/tsl026.htm.

[8] Krste Asanovic et. al., “A view of parallel Computing Landscape”, *Communications of ACM*, Oct. 2009, vol. 52, No. 10, pp. 56-67.

[9] W-Chun and pavan Balaji, “Tools and Environments for Multicore and many core Architectures”, *IEEE Computer*, Dec. 2009, pp. 26-27.

[10] Chirag Dave, et. al., “CETUS: A source-to-source compiler infrastructure for multicores”, *IEEE Computer*, Dec. 2009, pp. 36-42.

[11] cetus.ecn.purdue.edu.