The future of computers - Part 1: Multicore and the Memory Wall

Russell Fish - November 17, 2011

Editor's note: In this first of three features, processor expert Russell Fish provides his take on the future of computers, and multicore processing in particular. Contentious and opinionated, Fish pulls no punches – Intel fans should not read this – but provides a good introduction to the problems and hyperbole surrounding multicore, before offering his take on the way forward.

After nearly 40 years wandering in the silicon wilderness searching for the promised land of CPU performance and power, computer deity, Berkeley's Dr. David Patterson handed down his famous "Three Walls." They were not etched in stone, but they may as well have been. These three immovable impediments defined the end times of increased computing performance. They would prevent computer users from ever reaching the land of milk and honey and 10 GHz Pentiums. There may be a hole in the Walls, but for now we know them as:

"Power Wall + Memory Wall + ILP Wall = Brick Wall"

- The Power Wall means faster computers get really hot.
- The Memory Wall means 1000 pins on a CPU package is way too many.
- ILP Wall means a deeper instruction pipeline really means digging a deeper power hole. (ILP stands for instruction level parallelism.)

Taken together, they mean that computers will stop getting faster. Furthermore, if an engineer optimizes one wall he aggravates the other two. That is exactly what Intel did.

Intel's Tejas hits the walls - hard
Inte engineers went pedal to the metal straight into the Power Wall, backed up, gunned the gas, and went hard into the Memory Wall.
The industry was stunned when Intel cancelled not one but two premier processor designs in May of 2004. Intel's Tejas CPU\textsuperscript{2}, Sanskrit for fire, dissipated a stupendous 150 watts at 2.8 GHz, more than Hasbro's Easy Bake Oven.\textsuperscript{3}

The Tejas had been projected to run 7 GHz. It never did. When microprocessors get too hot, they quit working and sometimes blow up.\textsuperscript{4}

So, Intel quickly changed direction, slowed down their processors, and announced dual-core/multicore. Craig Barrett, Intel's CEO at the time, used a Q&A session at the 2005 Intel Developer Forum to explain the shift:

\textit{Question: “How should a consumer relate to the importance of dual core?”}

\textit{Answer: Fair question. I would only tell the average consumer... it's the way that the industry is going to continue to follow Moore's Law going forward - to increase the processing power in an exponential fashion over time... Dual core is really important because that's how it's happening. Multicore is tomorrow... Those are the magic ingredients that the average consumer will never see, but they will experience [them] through the performance of the machine.”}\textsuperscript{5}

When an engineer, or anyone else for that matter, explains how to "increase the processing power in an exponential fashion" with "magic ingredients," it is wise to double check his math.

Multicore means that two or more complete microprocessors are built on the same chip attached to a shared memory bus. If one microprocessor is good, two must be twice as good, and four must be......even better!

Barrett's statement was of course marketing hyperbole, an attempt to rally developers, customers, and stockholders behind the badly stumbling technology icon.

In small amounts multi-core can have an effect. Two or even four cores can improve performance. However, doubling the number of cores does not double the performance. But the news is particularly alarming for popular data intensive cloud computing applications such as managing unstructured data.

\textbf{Sandia Labs}\textsuperscript{6} is an 8,000 person US government lab that dates back to the Manhattan Project. Besides nuclear weapons work, Sandia hosts several supercomputers and is a major computer research center. Sandia Labs performed an analysis of multi-core microprocessors running just such
Sandia checked Barrett's math and found it lacking in correctness. They reported as the number of cores increased, the processor power increased at substantially less than linear improvement and then **decreased** at an exponential rate. Sandia explained the decrease thusly, "The problem is the lack of memory bandwidth as well as contention between processors over the memory bus available to each processor."

### Multicore meet Memory Wall

![Graph showing performance vs. number of nodes](image)

INTEL acknowledged the problem as "a critical concern" and recognized the criticality with the following understatement, "Engineers at Sandia National Laboratories, in New Mexico, have simulated future high-performance computers containing the 8-core, 16-core, and 32-core microprocessors that chip makers say are the future of the industry. The results are distressing."

### Breakfast at Barrett's

To help understand the problem, imagine the "Make Breakfast" application:

1. Start with a single cook in a small kitchen to scramble eggs.
2. Add another to fry bacon in parallel with the first doing the eggs, and making breakfast goes a bit faster.
3. Add another to brown, butter, and slice the toast, and you might gain a bit more speed.
4. Yet another person can set the table, pour the juice, and serve.
5. It is still not fast enough so you add a dozen more cooks to get that "magic", "exponential" increase in performance.

There are problems however.
The first four cooks all need to use the refrigerator. In our culinary world each would twiddle his thumbs waiting his turn. This is the physical world equivalent of "contention between processors of the memory bus."

Now how much faster do you think breakfast will be ready when those additional dozen cooks are added? As Borat would say, "Not so much."

Allocating multicores to do useful work is similarly challenging to allocating cooks. Just how many cores can usefully accelerate Microsoft Word, Excel, or PowerPoint? The answer is, not many. Legacy PC applications do not factor nicely into many pieces.

**No need to be embarrassed**

Fortunately multicore is really about enabling the future rather than accelerating the past. Many of the really interesting and commercially valuable future computing opportunities are of type known as "embarrassingly parallel". This means that the problems may be divided into many independent pieces and worked on separately.

Imagine the 1,000 page New York City phone book. Your task is to find a specific number in those thousand pages. The phone book is ordered by name, but the phone numbers are essentially random. You would begin a linear search at page 1 and continue until you found the desired number, probably several days later.

On the other hand, if you have a big Facebook community, you can give a page to each of your 1,000 friends and ask all of them to search at the same time. You will find the number approximately 1,000 times faster than if you searched by yourself.

The phone number search is an embarrassingly parallel problem. It can be divided into many independent tasks which can execute in parallel, and the results of all the tasks can be combined to produce a result. This is also the description of one of the most important applications in the massively parallel computer world. It is called MapReduce, and it runs Google's million server network search engine.

Intel has similarly described their future view of embarrassingly parallel problems as "Recognition, Mining, and Synthesis." In other words, future applications will manipulate and manage patterns of information. The pattern might be a sentence of text, a face in a crowd, or a phrase from a spoken speech. The datasets containing the patterns are immense; terabytes, petabytes, and eventually exabytes.

These enormous datasets factor nicely across embarrassingly parallel many CPU systems, and that
was Intel's plan with multicore until it ran into the Memory Wall. Faced with the wall they have staved off total destruction with a series of architectural tricks.

**Cache for clunkers**

One attempt to mitigate the Memory Wall was computer cache. Cache is a small dedicated local memory that sits between a CPU core and main memory. Cache takes advantage of the fact that sometimes both instructions and data may be reused, and if they are already present in the local memory, the CPU core need not get it from main memory.

Back to the kitchen example. Each cook has a plate containing the breakfast ingredients retrieved from the refrigerator. Instead of going back to the refrigerator to get two more slices of bread each time the toaster dings done, the toast cook reaches for his stack of bread on his ingredient plate. The ingredient plate will eliminate some contention for the refrigerator, but the refrigerator still has one door, and that door is a choke point.

If the kitchen is cooking for 2 people, the ingredient plates will be small. As the kitchen scales up to serve a larger family, the ingredient plates will get bigger. This is similar to increasing a computer's cache size. Despite the ingredient plates, the refrigerator will be getting a lot of use loading up those plates.

When the kitchen decides to expand to become a commercial restaurant, management may decide to invest in a dual-door refrigerator. However there is no dual-door refrigerator for CPUs.

Caches are of limited use for data intensive applications such as MapReduce unless the entire dataset can fit in the cache thereby duplicating the main memory. Caches already occupy over half the silicon area of some CPUs and consume much of the power.

**Rambus Redux**

Since caches are nearing their practical limits in size, the obvious question from computer users is, "Why can't you just increase the memory bus bandwidth?"

There are three ways to increase memory bus bandwidth:

1. Increase memory transfer speed.
2. Increase memory transfer size.
3. Move data closer to CPUs.

Memory transfer speed is limited by the Power Wall. The faster the data is pushed on the memory bus, the more heat is generated. Several attempts have been made to increase speed other than just pushing harder and getting hotter. About a decade ago, Intel designed some processors using a high-speed technology called Rambus. The technology was fast but more expensive than other technologies.

More recently, Intel has revisited the memory transfer problem with a similar but updated technology they call Hybrid Memory Cube (HMC). Few details have been publicly released, but it appears that HMC may suffer the similar cost problem to Rambus.
Memory transfer size has increased over the years. The first microprocessors only fetched a single instruction from memory at a time. To increase memory transfer size Intel and most other computer makers licensed a technology that fetched multiple instructions.\textsuperscript{14}

Once again the Power Wall limits the ultimate width of the memory transfer size. Each memory bus pin causes the CPU to consume power as it is charged and discharged. Increasing the bus width to mitigate the Memory Wall therefore aggravates the Power Wall.

**Move data closer to CPUs**

The Memory Wall could be substantially eliminated if data was stored adjacent to the CPUs. One way to do this is to increase the size of cache memory so it can act as main memory. Access to cache is up to 100x faster than access to main memory and the Memory Wall would collapse like the Walls of Jericho. How hard could it be?

It turns out to be pretty hard. Intel recently announced their new Itanium, "\textit{A 32nm 3.1 Billion Transistor 12-Wide-Issue Itanium Processor for Mission Critical Servers}".\textsuperscript{15} This monster chip is nearly one inch on a side and only has 54 Mbytes of cache, reportedly twice the size of any cache of any other microprocessor ever made. Intel has not announced pricing for this Goliath, but we can be pretty certain it will be more expensive than their current top of the line Itanium, the 9350.

The Itanium 9350 has 24 Mbytes of cache, four cores, and has a recommend customer price of only \textdollar{3,838}.\textsuperscript{16}

**Your tax dollars at work**

Other smart people have worked on the problem besides Intel. Merging main memory and CPU was first proposed in 1989\textsuperscript{17}, but the first chips were not built until about a decade later. Most of these efforts were funded by the Defense Advanced Research Projects Agency (DARPA). They included \textit{Exacube PIM}\textsuperscript{18}, \textit{Gilgamesh}\textsuperscript{19}, \textit{Cyclops}\textsuperscript{20}, and the best known of all, David Patterson's \textit{iRAM}\textsuperscript{21} That would be the same Patterson as "Patterson's Walls".

All attempted to include main memory and CPUs on a single chip. All essentially succeeded in proving that the Memory Wall could be breached, but none made the slightest ripple in the commercial world. As product with commercial prospects they failed for the same reason an Itanium with cache the size of main memory would fail. The cost would have bankrupted King Solomon's Mines.
The early patent clearly explained that the key to the CPU was, “integration on the memory chip die”. Instead the DARPA efforts all attempted memory integration on the logic chip die.

The following graph illustrates the problem:

![Graph showing cost comparison between microprocessor transistors and memory transistors](http://www.venraytechnology.com)

### Why doesn't everybody do it?

The obvious question is why everybody doesn’t build CPUs integrated on memory die. The general idea is to take a commodity DRAM and slice it up and insert the CPUs. Sounds simple enough. (You can see an animation of the process here: [http://www.venraytechnology.com](http://www.venraytechnology.com)). In general there were two impediments, a small bump and a large hump. Our group headquartered in Dallas but with able assistance from Europe and China attacked the bump and the hump.

The small bump was the lack of device libraries. The easiest way to design a CPU is to ask a foundry such as TSMC, UMC, SMIC, or Global Foundries for their library of cells for the particular process you want to build. The cells are predefined logical elements that can be as simple as an inverter and as complex as a multiplier or a memory array. The existence of these logic libraries significantly reduces the time necessary to complete a design. The libraries are nicely integrated into common design tools.

There really isn't much DRAM foundry business. You cannot call up Nanya, Promos, or Samsung and ask them to send over their logic libraries. They have no such thing.

Creating and characterizing a logic library is not difficult, but it was time consuming. We ultimately created just over 60 digital cells and 14 analog cells and supercells. The CPUs were built from these cells.

The hump on the other hand was a much bigger problem. DRAM processes are designed for low cost and low leakage. To achieve the low cost, DRAMs only use three layers of metal compared to 10 or 12 layers for CPU processes. The metal layers enable connections between the logic gates that constitute the CPUs.

The solution was to invent a CPU architecture that was sophisticated enough to perform well on data intensive tasks but compact enough to route with three layers of metal and not blow up the die size of the parent DRAM. The resulting 32-bit microprocessor cores less cache are implemented in 22,000 transistors including multiplier and barrel shifter. The virtual memory controller adds another 3,500 transistors. The architecture could be called a modified RISC.
The wiring was simplified by creating each of the 32-bits individually and then connecting the bits together with control lines. (The technique is called bit-slicing.) As an added advantage, bit-slicing greatly eases moving the microprocessor from one DRAM design to another.

We relied heavily on Sandia Labs and their MapReduce-MPI\textsuperscript{24} for fine tuning the caches and tweaking a few instructions.

MapReduce is one of the most commercially popular massively parallel applications. It divides huge databases across many microprocessors in order to perform rapid searches in parallel. MapReduce was first proposed by Google to coordinate the million servers that constitute its search engine, probably the largest cloud computing installation in existence.\textsuperscript{25} MapReduce-MPI is written in C++ and tuned for Sandia's supercomputers, such as Red Storm.\textsuperscript{26} It is therefore much faster than Hadoop\textsuperscript{27}, the popular MapReduce implementation written in Java.

For purposes of benchmarking a massively parallel system sixteen TOMI Borealis chips were configured to fit on a 4-inch circuit board about the size of an ordinary memory DIMM. The board included 128 cores, 2 Gbytes of DRAM, network controller, and a switching power supply. It is shown below with the heat spreader removed.
We assumed four rows of 32 of these DIMMs arranged on a single 19-in. motherboard with provisions for forced air cooling. This was compared against a full 19-in. rack of Intel Xeon processors.

The single card solution based from CPUs built on commodity DRAMs outran and entire rack of Intel Xeon E5620 multicores running Sandia Labs MapReduce-mpi on a 256-Gbyte dataset.

In the next article we will investigate the Power Wall more thoroughly.

Go to Part 2: Future of computers - Part 2: The Power Wall

About the author:
Russell Fish's three-decade career dates from the birth of the microprocessor. One or more of his designs are licensed into most computers, cell phones, and video games manufactured today.

Russell and Chuck Moore created the Sh-Boom Processor which was included in the IEEE's "25 Microchips That Shook The World". He has a BSEE from Georgia Tech and an MSEE from Arizona State.

Footnotes:
1. http://www.slidefinder.net/future_computer_architecture_david_patterson/6912680


