Parallelism:
The Real Y2K Crisis

Darek Mihocka
August 14, 2008

The Free Ride

- For decades, Moore's Law allowed CPU vendors to rely on steady clock speed increases:
  - late 1970's: 1 MHz (6502)
    mid 1980's: 8 MHz (68000, 80286)
    1990: 33 MHz (386, 486)
    1994: 90 MHz (Intel Pentium)
    2000: 1000 MHz (AMD Athlon, Intel Pentium III)
    2001: 3000 MHz (Intel Pentium 4)

No Limits?

- In the mid 1990's incredible increase in clock speed due to introduction of out-of-order (OOO) cores such as Intel "P6" (Pentium Pro, Pentium II, Pentium III, Core 2) and AMD Athlon.

- Hardware took some of the optimization burden away from the programmer by allow you to write "good enough" code. Hardware re-ordered instructions for you.

- OOO cores allow instructions to "pass" each other, allowing pipeline bubbles such as memory cache miss latency to not hold up later instructions.

Reality Sets In

- In 2001, Intel predicted that by 2005 the Pentium 4 would be running at 10 GHz:
  - late 1970's: 1 MHz (6502)
    mid 1980's: 8 MHz (68000, 80286)
    1990: 33 MHz (386, 486)
    1994: 90 MHz (Intel Pentium)
    2000: 1000 MHz (AMD Athlon, Intel Pentium III)
    2001: 3000 MHz (Intel Pentium 4)
    2005: 10000 MHz (predicted)
    2007: 3200 MHz (Intel Core 2 actual)
- Oops! What happened!

The Clock Speed Brick Wall

- CPU core clock speed increases outpaced memory speeds, requiring larger on-chip L1 and L2 caches, and deep pipelines to absorb the memory latency bubbles.
- Faster clock generates more heat, requires more cooling – bigger louder fans!
- Faster clock speed brings up other issues - physical layout of the core, gate propagation delays, and loss due to capacitance.
- Intel Pentium 4 and AMD Athlon peaked out at close to 150W power consumption! Laptop computers literally started catching on fire.

Wasteful Computing

- 1990's PCs and Macintosh computers did not need cooling fans on the CPU, just a passive heat sink. Typical laptop: 20 watts, desktop: 90 watts.
- Today, 1 billion Windows PCs potentially wasting 100W of power each.
- Worse, clock speed is not even an indication of overall performance. An Intel Pentium 4 on average Windows code retires about 1 instructions every 2 clock cycles - 3.0 GHz Pentium 4 delivers 1500 MIPS throughput. “Older” Pentium III and AMD Athlon retire 1 instruction per clock cycle.
- Not surprising then that early reviews of Pentium 4 pointed out that both the "older" Pentium III and Athlon ran cirlces around the "new" Pentium 4.
- The release of the Pentium 4 and the design approaches of the hardware industry in late 2000 was the REAL Y2K crisis.

Improving IPC

- Clock speeds hit a brick wall at about 3.0 GHz. Further performance gains need to come from techniques that increase IPC (instructions-per-cycle) throughput.

- Some approaches so far:
  - wider pipelines to improve instruction level parallelism: Intel Core 2 for example can retire 4 instructions per cycle (but usually starved at decoder stage).
  - Larger caches - 256K to 512K to 1M to 2M to 4M to 48M!?!?!?
  - More emphasis on SIMD (SSE) - execute identical arithmetic or FP operations in parallel
  - 64-bit registers - perform larger calculations using fewer instructions.

- Result: last generation of Pentium 4 in 2005 had much better IPC throughput but still suffered from power consumption issues.

- First in the notebook market (the Pentium M release in 2003) and then the desktop (Core Duo, Core 2 in 2006), Intel quietly went back to using the 1990's "P6" core which achieves higher IPC at lower clock speeds.

- As if the Pentium 4 never happened!

Multi-Core Is Here

- In 2005, Intel rebooted, announcing push for multi-core computing, trading peak performance of a single thread (which consumes a lot of power) for best aggregate performance of multiple cores.
- This is not a new concept. Supercomputers did this decades ago.
- Parallel computing comes to the PC desktop! Multi-core computing *is* parallel computing.

Parallelism Is Difficult

- Parallel computing passes the performance burden back to the programmer.
- Donald Knuth quoted in April 2008 said of multi-core computing: "hardware designers have run out of ideas... trying to pass the blame for the future demise of Moore's Law to the software writers".
- Most algorithms are single threaded. Trees, hash tables, linked lists, etc.
- Most programmers write single threaded code, e.g. performing synchronous file I/O (e.g. ReadFile or fwrite()).
- Object oriented languages perform hidden memory allocations and garbage collection "under-the-hood", resulting in unpredictable execution speeds, erratic pauses, and serial execution.
- Popularity of virtual machines creates more serialization - Java and .NET virtual machines rely on jitting, which tends to cause performance bottlenecks exactly at the time that you need it (application launch, loading new classes, etc).

Chaos

- The past 3 years have been a confusing time of hype about multi-core processors, hype about parallelism, and reliance on bogus SPEC benchmarks that are “embarrassingly parallel”, e.g. SpecJbb 2000.
- Ultimately, neither the Microsofts of the world nor the Intels of this world have made much progress. Developers still don’t even use critical sections correctly!
- Consumers are realizing that for the things they usual do, a 600 MHz single core 32-bit CPU is just as useful as the latest whiz bang 4-core 3.0 GHz 64-bit CPU.
- Recent popularity of ultra-mobile notebooks - ASUS EEE, OLPC XO, Acer Aspire - all based on 1990's AMD and Intel CPU cores!
- **How to make progress!?!?!?!?!**

Dynamic Translation

- IPC throughput still has a long way to go. Most current CPU cores can ideally retire at least 3 instructions per clock cycles, yet historic throughput is 0.5 to 1.0 IPC.
- Trend is to return to in-order cores (Xbox 360, Intel Atom) - fewer gates means smaller die and lower power consumption, but lower throughput than an OOO core at the same clock speed.
- Burden is once again very much on the programmer and the code generator (C/C++ compiler or Java/.NET jitter) to give the CPU as well optimized code as possible.
- I am of the personal opinion that we cannot rely on programmers or static compilers to fix the problem - code that is "ideal" today may not work on next year's CPU architecture.
- Legacy code is always doomed to become inefficient over time. Is the solution really to just keep rewriting it over and over again?
- One solution: use virtualization to dynamically recompile code on-the-fly. This is what Transmeta did almost 10 years ago and what needs to happen today in mainstream computing.
- Take the “write once run anywhere” approach of Java, but apply it to *all* code.