Improving the future by examining the past

Charles P. Thacker

June 2010

Collaborators at PARC, DEC SRC, and Microsoft

- Butler Lampson (Bcc500, MAXC, Alto, Dorado, Firefly, AN1, AN2, Tablet PC)
- Ed McCreight (Alto)
- Mike Schroeder (Firefly, AN1, AN2)
- Roy Levin (Firefly)
- Andrew Birrell (Firefly, Beehive)
- And dozens more...

Thesis

- Computer architecture has hit a wall.
- This will require us to build future systems in new ways.
- New systems will require changes in the way we program them.
- In designing future systems, we should examine some of the decisions made in the past. Some choices we made may be less relevant, and paths not taken might be more appropriate today.

Forty years of progress

Item	Alto, 1972	My home PC, 2010	Factor
CPU clock rate	6 MHz	2.8 GHz (x4)	1900
Memory size	128 KB	6 GB	48000
Memory access time	850 ns	50 ns	17
Display pixels	606 x 808 x 1	1920 x 1200 x 32	150
Network	3 Mb Ethernet	1 Gb Ethernet	300
Disk capacity	2.5 MB	700 GB	280000

Exponential drivers for computing

- Rotating magnetic storage
  - Today, we can store every bit of information we receive during a lifetime on a few disks.
- Glass fibers
  - No two points on earth are more than 50ms apart.
  - Bandwidth is no longer scarce.
- Semiconductors and Moore’s Law
  - Often misunderstood

The nature of the wall, and some review

- $P = C * V^2 * f$
  - Relates power, capacitive load, operating voltage, and frequency
- If we scale C and V by k: $P' = C/k * (V/k)^2 * f'$
- $= (C * V^2 * f') / k^3$
  - k is the “feature size”. 90nm -> 45 nm is k = 2.
  - New chip is 1/k2 in size, 1/k3 the power at the same frequency.
- If f’ = k * f, the power per unit area is *unchanged*.
- Semiconductor makers have used scaling in different ways.

Scaling for Memories and CPUs

- RAM manufacturers used scaling to improve *density*.
  - Lower voltage, somewhat higher frequency and power, but *lots* more bits.
- CPU manufacturers took another path: higher clock rates, more complex designs.
  - Result: CPUs got *very* hot (the power wall).
  - Memories got bigger, but not much faster (the memory wall).
- Future trend: Rather than a single fast CPU, we will see many simpler CPUs on each chip.
  - Moore’s law is *not* at an end.
- This has profound effects on software.

Example 1: Virtual Memory

- Introduced in the Manchester Atlas (’60s).
  - 16 KW core memory (small, fast).
  - 1 MW magnetic drum (large, slow).
- Simple idea: Make the core memory *appear* to be as large as the drum.
  - Divide core into 32 512-word pages.
  - Use an associative memory (TLB) to map virtual (drum) addresses to real (core) addresses.
  - Also used a “learning program” to decide which pages to transfer back to drum.
- Used in most computers today.
  - Except supercomputers and embedded systems.

VM continued

- As system speed and size increased, problems arose:
  - Memories grew faster than page size. => larger tables.
  - Associative memories are expensive and slow. => cached page tables.
  - Multiprogramming. => more tables.
- Approaches: paged tables, segments, paged segments.
- Sometimes paging has unanticipated consequences.

Problems VM still solves today

- Protection
  - This will be a problem as long as we program in languages with pointers.
- Relocation
  - Segments are simpler.
- Fragmentation
  - Compacting garbage collectors?
- Note: Size of real memory is no longer a problem.
- VM was invented in a time of scarcity. Is it still a good idea?

Example 2: Memory coherence

- Until the ’70 coherence wasn’t an issue, but caches changed this.
  - With uniprocessors, we only needed to worry about I/O coherence.
  - With early multiprocessors, bus-based snooping protocols were adequate and simple.
- With the larger-scale multiprocessors of the ‘80s, more complex protocols were needed.
  - Busses no longer worked. Point-to-point links were needed.
  - Protocol complexity skyrocketed, requiring formal methods to ensure correctness.

Coherence vs. Message-passing

- Why did we choose coherent memory?
  - After all, message-passing and shared memory are *duals*.
  - Messages are *much* simpler.
- Possibility 1: Programming is easier.
- Possibility 2: We could make it work adequately at small scale, and didn't foresee the complexity that larger scales would introduce.
- Would we make the same choice today?

Example 3: Threads and locks

"I am convinced more than ever that this type of work is very difficult, and that every effort to do it with other than the best people is doomed to either failure or moderate success at enormous cost".
Edsger Dijkstra, 1968

- Lock-based programming is still hard.
  - Coarse grained locking is easy, but inefficient.
  - Fine-grained locked is difficult to reason about and get right. Bugs are frequently Heisenbugs.
  - Lock-based abstractions don't compose.
  - Even with "best practice" guidance by experts, programmers struggle.
- In the era of many-core systems, programs can't be written by only "the best people".

Alternatives to locks: Transactional memory?

- Lots of designs, mostly for STM.
- TM has problems of its own:
  - I/O
  - Coarse/fine grain arguments.
  - Transactions are *speculation*.
    - And if a transaction aborts work and energy are wasted.
- Main problem: We don’t have much experience in using TM.
  - Need some implementations
  - Need benchmarks
  - Need real systems
- Pure message passing is another contender.

Example 4: Complex CPUs

- ‘50s and ‘60s: The age of experiment
  - Decimal CPUs, String processors, Single-language CPUs (Lisp, Fortran, Algol), Stack machines.
- ‘70s and ‘80s: Consolidation and religious warfare:
  - RISC (IBM, Stanford, Berkeley) vs CISC (x86).
  - Lots of papers about the advantages of each, which gradually died out as each camp adopted techniques from the other.

Complex CPUs

- ’90s – today: Elaboration and the quest for ILP.
  - Multi-level cache hierarchies
  - Multithreading
  - Speculative execution *everywhere*.
  - Incremental advances in performance at the cost of *lots* of complexity, and *lots* of power.
  - The problem: Most sequential programs don’t exhibit very much ILP.

Possible directions for CPUs

- Intel Single-chip Cloud Computer
  - Lots of simple cores
  - No coherence
  - Highly efficient inter-core network
  - Efficient message passing
- Low power cores (Intel, AMD)
  - Recognition that for many applications, today’s systems are fast enough.
  - I don’t need to spend 100 watts to edit a presentation.

Example 5: Interrupts

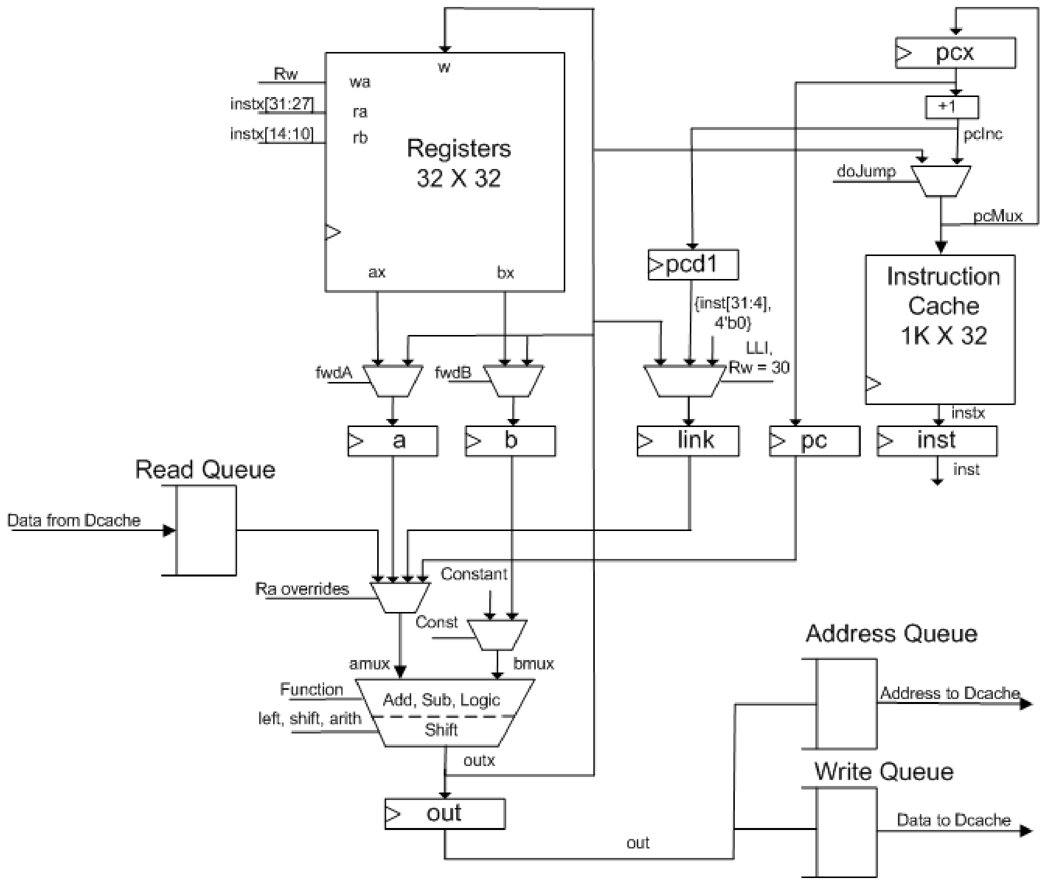
"..an interrupt system to fall in love with
is certainly an inspiring feature"
Edsger Dijkstra, 1968

- Do we need hardware interrupts?
  - In a world with many cores on each chip, their value seems less than when an idle CPU represented substantial waste.
  - They increase hardware and software complexity substantially.
- It *is* possible to build systems without interrupts.
  - The BBN Pluribus (1972). ARPAnet IMP V2.

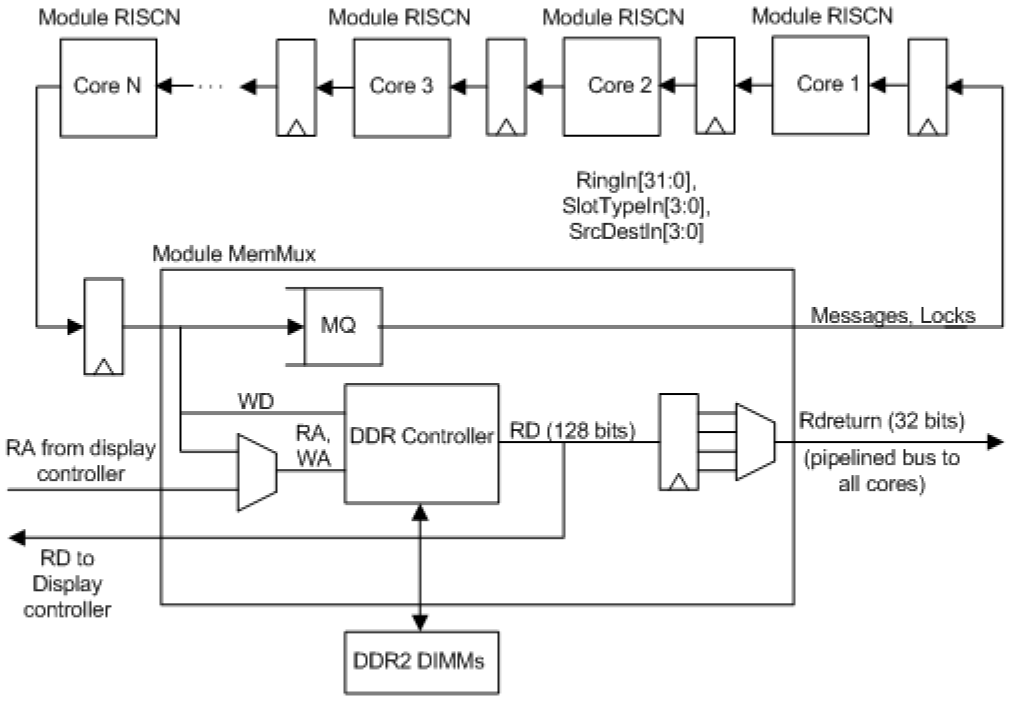
Exploring future architectural ideas: Beehive

- Implemented on a single FPGA development board.
  - Simple Verilog description (6K lines) that can be understood *and modified* by students. Our first users are MIT students.
  - Tool chain: MSIL and C compilers, assembler, simulator.
  - Licensed for research use.
- System provides:
  - 13 simple RISC cores. 100 MIPs each.
  - 1Gb Ethernet
  - Display control
  - 2 GB shared RAM. Non-coherent.
  - Message passing. No interrupts.
  - Hardware supported semaphores.
- Not intended to be competitive with “real” systems, but to be *much more malleable*, and *much faster* than simulators.
- Can run “real software”, e.g. complete OS .

Beehive Core



Beehive ring



Example 6: Packet-switched networks

- Today’s large data centers provide an opportunity for us to reexamine *local-area* networking.
- Data centers are not the Internet:
  - Known topology, known names
  - Small diameter => low latency
  - Fewer end nodes.
    - Tens of thousands, not hundreds of millions
  - Faster inter-switch links
    - 10 Gb today, 40 and 100 Gb/sec soon.
  - Different traffic patterns.

Why use packet switching?

- Adequate performance, at least initially.
  - 3 Mb/s was fast enough, even for voice.
- Retransmission on packet loss worked because of small network size.
- Short addresses weren’t a problem in small networks.

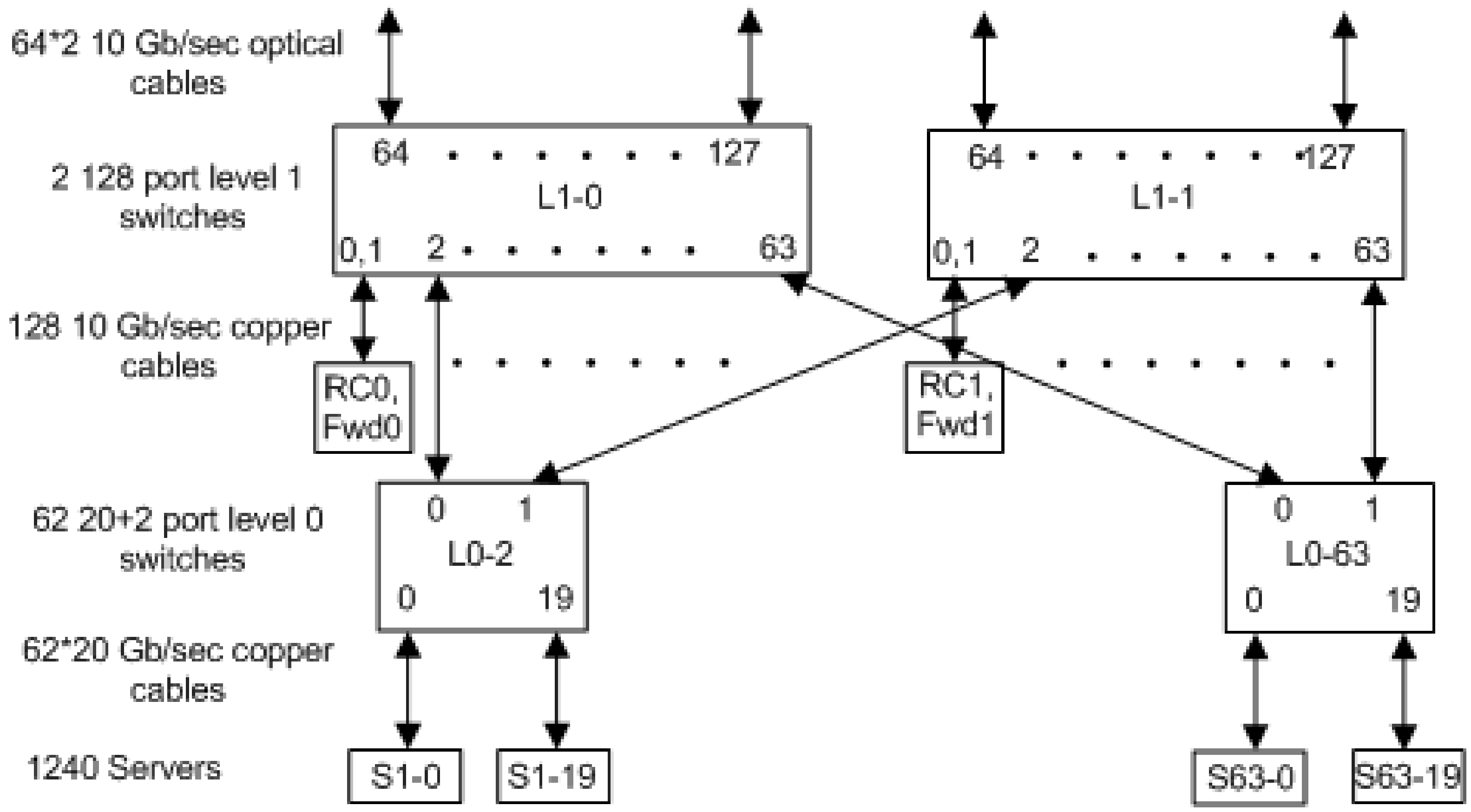
Faster, more reliable links brought problems

- Point-to-point links rarely make errors, but switches/routers still need large buffers.
  - And latency suffers.
- Routing decision at each switch along a path are complex.
  - Need more processing per packet.
- Switch/router complexity grew, and reliability suffered.
- Packet switching is problematic at 40/100 Gb/s.
  - Window-based congestion control is inefficient with thousands of packets in transit.
  - Sources get feedback after the damage is already done.

Should we revisit circuit switching?

- Circuit switching (ATM) *is* used in the Internet backbone.
- Admission control rather than congestion control is used to avoid data loss.
- Routing is determined at call setup, and doesn’t change dynamically.
- The key is to make call setup time << transmission time.
- This can work if the number of hops is small and the links are relatively short.
- Switches require almost no buffering.

A possible arrangement


64*2 10 Gb/sec optical cables
2 128 port level 1 switches
128 10 Gb/sec copper cables
62 20+2 port level 0 switches
62*20 Gb/sec copper cables
1240 Servers
64
127
L1-0
0,1
2
63
64
127
L1-1
0,1
2
63
RC0, Fwd0
RC1, Fwd1
0
1
L0-2
0
19
0
1
L0-63
0
19
S1-0
S1-19
S63-0
S63-19

Some final thoughts

- The systems of the 21st century are different.
- We should rethink earlier design decisions based on new realities.
- Computers still can’t do a lot of things, so we still have challenges:
  - Drive my car
  - Learn my preferences, and be a colleague rather than a slave
  - Help educate my grandchildren.
  - Enhance my privacy, rather than eroding it.