Using Machine Learning to Manage Big Data ...Securely on the Cloud

Yale N. Patt
The University of Texas at Austin

Severo Ochoa Lecture
June 6, 2017
Just kidding!
Processor Paradigms: Evolution or Disruption
(and what does this have to do with Moore’s Law)

Yale N. Patt
The University of Texas at Austin

Severo Ochoa Lecture
June 6, 2017
What I want to do today

- Moore's Law
- The VonNeumann Paradigm
- The Transformation Hierarchy
- Why paradigms
- Examples and their evolution
- What is needed moving forward
- How do we get there
Moore’s Law

• **What it is**
  – A law of physics
  – A law of microarchitecture
  – A law of psychology

• **What it enabled**
  – Smaller transistors → higher frequencies
  – More transistors → more functions to go faster

• **Why it can not go on forever**

• **And they often create new needs/problems**

• **So they give way to newer paradigms**
What it enabled (starting in 1971)

- Pipelining
- Branch Prediction
- Speculative execution (and recovery)
- Special execution units (FP, Graphics, MMX)
- Out-of-order execution (and in-order retirement)
- Wide issue
- Trace cache
- SMT, SSMT
- Soft errors
- Handling LLC misses
What will Moore’s Law ending mean?

• No more faster transistors
• No more increase in the number of transistors
• We won’t even be able to have them all on at once
• So how can we stay on the performance curve?
The VonNeumann Paradigm

• A straightforward model of executing programs
  – Fetch, Decode, Evaluate address, Fetch Data, ...

• Many have suggested its demise
  – I insist it will remain
  – Our best mechanism for maintaining order, not chaos

• But it will be augmented with other structures.
The Transformation Hierarchy

• **What it is**
  – Much more than just the “Software Stack”

• **Why it will be useful moving forward**
Problem

Algorithm

Program

ISA (Instruction Set Arch)

Microarchitecture

Circuits

Electrons
Why Paradigms?

- Paradigms are invented to satisfy needs/problems
- And they often create new needs/problems
- So they give way to newer paradigms
We could start with some very old Paradigms

• Approximate Computing

• Machine Learning

• Quantum Computing
With proper context…

• Floating point $\rightarrow$ Approximate computing

• Adaline/Perceptron/Learn Matrix $\rightarrow$ Machine Learning

• Accelerator $\rightarrow$ Quantum
Some Paradigms

- Tomasulo (what was good, what was bad)
- Data Flow (what was good, what was bad)
- HPS
- CDC6600
- HEP (what was good, what was the problem)
- SMT (what was good, what was bad)
- SIMD
- GPU
- Systolic Array
- Spatial Computing
- Non-Von, BVM, Connection Machine
- Multi-core
- RISC
- User-writeable Control Store
...with some Assists: some good, some not

• In the microarchitecture
  – Branch prediction
  – Wider issue
  – Predicated execution
  – Extra memory pipes
  – FPGAs (big increase in flexibility at small cost?)

• In the ISA
  – Predicated execution
  – Unaligned access
  – Register windows
  – Delayed branch
…so I must add: BE CAREFUL

• **Add something to the microarchitecture: No problem**
  – If a bad idea, discard it on the next implementation

• **Add something to the ISA**
  – You are stuck with it forever
Examples of Paradigm Evolution

- HPS
- SMT
- GPU
- VLIW
- Spatial Computing
- Many core
- Accelerators
HPS

- **Tomasulo + Data Flow → HPS**

- **Tomasulo had out-of-order, NOT precise exceptions**
  - Also, ONLY the floating point unit
  - Also, ONLY one operation per instruction
  - Also, Stall on a branch (no steady supply of operations)

- **Data Flow had micro-ops, but too unwieldy**
  - Hard to take interrupts
  - Hard to debug

**HPS took the good, added in-order retirement,**

**Restricted window, wide issue, aggressive br.predictor**
The HPS Paradigm

- **Processing micro-ops! (Restricted Data Flow)**

- **Incorporated the following:**
  - Aggressive branch prediction
  - Speculative execution
  - Wide issue
  - Out-of-order execution
  - In-order retirement
SMT

•  **HEP + ooo → SMT → SSMT**

•  **HEP was brilliant, ahead of its time (SPIE 1977)**
  –  *But issued only one instruction each clock cycle*

•  **Actually, CDC6600 → HEP**

•  **SMT (Hirata, ISCA 1992, Nemirovsky 1994, UW 1995)**

•  **What if you only have one thread?**
  **SSMT (Chappell, ISCA 1999, Dubois, USC Report ’98)**
GPU

- SIMD + SMT + Predicated Execution $\rightarrow$ GPU
- If the software can pay attention to branches
- If the software can organize memory accesses
VLIW

• Horizontal microcode → VLIW
• Not good for General Purpose Computing
• But good for domain specific stuff
  – Microcode Emulation
  – DSP chips
• i.e., when the software is known in advance
Spatial Computing

- Systolic Array + FPGAs $\rightarrow$ Spatial Computing
- HT Kung (1979): not enough transistors, too “asic”
- Today, stream data through a data flow graph
- If the software can produce the data flow graph
Multicore, Manycore

- *Early days + Moore’s Law* → Multicore, Manycore
- *(Early days = Nonvon, BVM, Connection Machine)*
- *Not enough transistors in 1985 (one-bit data path)*
- *Still have the problem: how to program them!*
Accelerators

• Many implementation mechanisms
  – ASICs
  – FPGAs
  – EMT instruction (with writeable control store)

• Examples (Quantum computing, Machine learning)

• Requires the attention of
  – The person writing the algorithm
  – The programmer
  – The compiler writer
  – The microarchitect
In fact, as Moore’s Law finally ends

• We will have to think smarter

• That will mean bringing to the table
  – Those working at all levels of the transformation hierarchy
Some Thoughts

• Each paradigm came because there was a need, and someone saw a way to accommodate that need
  – It got replaced because the paradigm exposed subsequent needs due to new technology or new requirements

• Be careful what you put something into the ISA

• ILP is still important; MORE important as Moore’s Law fades

• We need to engage everyone (The transformation hierarchy)

• What must happen in order for us to be able to engage everyone
People need to understand more than one layer
...which requires a fresh approach to Education!
Thank you!
RISC

• What was it? (Depends on who you ask!)

• The soul: John Cocke – Open microcode.
  The compiler generates the control signals

• Then the young professors picked it up
  – Patterson: Simple instructions needing single cycle execute
  – Hennessy: The compiler and his pipeline reorganizer

• By 1989, Hennessy said: fast streamlined pipelines
  – ...which is actually consistent with John Cocke

• As a useful paradigm:
  – Streamlined hardware
  – Very sophisticated compiler