End of Moore’s Law Challenges and Opportunities: Computer Architecture Perspectives for the Post-ISA Era

Prof. Margaret Martonosi
Dept. of Computer Science
Princeton University
In the beginning... Moore’s Law

Cramming more components onto integrated circuits

With unit cost falling as the number of components per circuit rises, by 1975 economics may dictate squeezing as many as 65,000 components on a single silicon chip

By Gordon E. Moore
Director, Research and Development Laboratories, Fairchild Semiconductor division of Fairchild Camera and Instrument Corp.

1965
In the beginning… Moore’s Law

1965
Decades of Moore’s Law scaling
Computer Architecture: Mediator between Technology & Applications

**Application Trends:**
AI, Machine Learning, Neural Nets, Big Data, Sensing, Media, ...

**Technology Trends:**
End of Moore’s Law, Dennard Scaling, Heterogeneity, Bio, Quantum, ...

Computer Architecture
We are here...

Heterogeneity at all levels: IoT to Datacenters
Disruptive Moment -> Research Questions

- Disruptive moment: HW/SW interface undergoing a seismic change.
  - ISAs still useful, but little/no relevance as abstraction layer.
    - Apple A8-present: >50% of chip area is accelerators that have no ISA.
    - NVIDIA PTX vs. SASS: ISA hidden under other layers.

Questions for the Future:

- How to program these highly heterogeneous systems?
- How to verify them?
- How long will these approaches sustain us before we need to employ more aggressive technology shifts?
Computer Architecture’s Post-Moore Response

Technology Shifts: "Post-Moore"

Short Term: Specialization

Longer Term: Bio, Quantum

All: Need new Software, Abstractions
Mitigating Complexity and Improving Verification in Heterogeneous and Specialized Hardware

Technology Shifts: "Post-Moore"

Short Term: Specialization

Longer Term: Bio, Quantum

All: Need new Software, Abstractions
Heterogeneity => Specifying and Verifying Multiple Memory Models

- ~6 or more ISAs on-chip
- Complex storage hierarchy
- Multiple memory consistency models
- Multiple bus protocols

- Growth in on-chip heterogeneity -> New ”Lingua Franca” Communication via shared memory

- Concurrency + Shared Memory Communication -> Need Memory Models to specify ordering rules.
First Memory Model: Sequential Consistency

- Defined by [Lamport 1979], execution is the same as if:
  - **(R1)** Memory ops of each processor appear in program order
  - **(R2)** Memory ops of all processors were executed in some global sequential order

<table>
<thead>
<tr>
<th>Program</th>
<th>Legal Executions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Thread 0</td>
<td>Thread 1</td>
</tr>
<tr>
<td>x=1</td>
<td>x=1</td>
</tr>
<tr>
<td>y=1</td>
<td>y=1</td>
</tr>
<tr>
<td>r1=y</td>
<td>r1=y</td>
</tr>
<tr>
<td>r2=x</td>
<td>r2=x</td>
</tr>
<tr>
<td>Thread 1</td>
<td></td>
</tr>
<tr>
<td>x=1</td>
<td>x=1</td>
</tr>
<tr>
<td>y=1</td>
<td>y=1</td>
</tr>
<tr>
<td>r1=y</td>
<td>r1=y</td>
</tr>
<tr>
<td>r2=x</td>
<td>r2=x</td>
</tr>
</tbody>
</table>
In a nutshell: MCMs are the spec of what value will be returned when your program does a load.
Motivating Example: ARM Read-after-Read Hazard

- ARM ISA spec ambiguous regarding same-address Ld → Ld ordering:
  - ARM compilers did not insert helper instructions
  - Some ARM hardware implementations relax same-address Ld → Ld ordering
- C/C++ variables with atomic type require same-addr. Ld → Ld ordering
- ARM issued errata 1:
  - Rewrite compilers to insert fences with performance penalties
- ARM had helper instructions in ISA to guarantee correctness

The Check Suite: An Ecosystem of Tools

So far, tools have found bugs in:
• Widely-used Research simulator
• Cache coherence paper
• IBM XL C++ compiler (fixed in v13.1.5)
• In-design commercial processors
• RISC-V ISA specification
• Compiler mapping proofs
• C++ 11 mem model

TriCheck [ASPLOS ’17] [IEEE MICRO Top Picks]
COATCheck [ASPLOS ’16] [IEEE MICRO Top Picks]
PipeCheck [Micro-47] [IEEE MICRO Top Picks]
CCICheck [Micro-48] [Nominated for Best Paper Award]
RTLCheck [Micro-50] [IEEE MICRO Top Picks Honorable Mention]

Our Approach
• Formal specifications -> Happens-before graphs
• Check Happens-Before Graphs via Efficient SMT solvers
  • Cyclic => A→B→C→A... Can’t happen
  • Acyclic => Scenario is observable
ARM Read-Read Hazard

Software Memory Model

Compilation

ISA Memory Model

Hardware Implementation

Microarchitecture

ARM Cortex-A9
ARM Read-Read Hazard

Software Memory Model

Compilation

ISA Memory Model

Hardware Implementation

Microarchitecture

Which HLL(s) to support?

<table>
<thead>
<tr>
<th>C11/C++11</th>
<th>ARMv7</th>
</tr>
</thead>
<tbody>
<tr>
<td>st(rlx)</td>
<td>STR</td>
</tr>
<tr>
<td>ld(rlx)</td>
<td>LDR</td>
</tr>
<tr>
<td>ld(acq)</td>
<td>LDR; DMB</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

ARM Cortex-A9
ARM Read-Read Hazard

How does this affect real programs?

Loading data through a pointer

Initial conditions: data=0, atomic *ptr=&data
Forbidden by C11: r1=2, r2=1

<table>
<thead>
<tr>
<th></th>
<th>T0</th>
<th>T1</th>
</tr>
</thead>
<tbody>
<tr>
<td>st</td>
<td>(data,1,rlx)</td>
<td>st(data,2,rlx)</td>
</tr>
<tr>
<td>r1</td>
<td>ld(*ptr,rlx)</td>
<td></td>
</tr>
<tr>
<td>r2</td>
<td>ld(data,rlx)</td>
<td></td>
</tr>
</tbody>
</table>

C11/C++11      ARMv7
st(rlx)        STR
ld(rlx)        LDR
ld(acq)        LDR; DMB
...

Software Memory Model

Compilation

ISA Memory Model

Hardware Implementation

Microarchitecture

ARM Cortex-A9
ARM Read-Read Hazard

Initial conditions: data=0, atomic *ptr=&data
Forbidden by C11: r1=2, r2=1

<table>
<thead>
<tr>
<th></th>
<th>T0</th>
<th>T1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>st(data, 1, rlx)</td>
<td>st(data, 2, rlx)</td>
</tr>
<tr>
<td>r1</td>
<td>ld(*ptr, rlx)</td>
<td></td>
</tr>
<tr>
<td>r2</td>
<td>ld(data, rlx)</td>
<td></td>
</tr>
</tbody>
</table>

C11/C++11                  ARMv7
st(rlx)                 STR
ld(rlx)                 LDR
ld(acq)                 LDR; DMB
...

<table>
<thead>
<tr>
<th></th>
<th>C0</th>
<th>C1</th>
</tr>
</thead>
<tbody>
<tr>
<td>ST [data] ← 1</td>
<td>ST [data] ← 2</td>
<td></td>
</tr>
<tr>
<td>LD [ptr] → r0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LD [r0] → r1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LD [data] → r2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Loading data through a pointer

Naïve compilation from C11 to ARMv7

Software Memory Model
Compilation
ISA Memory Model
Hardware Implementation
Microarchitecture
ARM Cortex-A9
ARM Read-Read Hazard

Software Memory Model

Compilation

ISA Memory Model

Hardware Implementation

Microarchitecture

Initial conditions: data=0, atomic *ptr=&data

Forbidden by C11: r1=2, r2=1

<table>
<thead>
<tr>
<th>T0</th>
<th>T1</th>
</tr>
</thead>
<tbody>
<tr>
<td>st(data,1,rlx)</td>
<td>st(data,2,rlx)</td>
</tr>
<tr>
<td>r1=ld(*ptr,rlx)</td>
<td></td>
</tr>
<tr>
<td>r2=ld(data,rlx)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>C11/C++11</th>
<th>ARMv7</th>
</tr>
</thead>
<tbody>
<tr>
<td>st(rlx)</td>
<td>STR</td>
</tr>
<tr>
<td>ld(rlx)</td>
<td>LDR</td>
</tr>
<tr>
<td>ld(acq)</td>
<td>LDR; DMB</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>C0</th>
<th>C1</th>
</tr>
</thead>
<tbody>
<tr>
<td>ST [data]⇒1</td>
<td>ST [data]⇒2</td>
</tr>
<tr>
<td>LD [ptr]⇒r0</td>
<td></td>
</tr>
<tr>
<td>LD [r0]⇒r1</td>
<td></td>
</tr>
<tr>
<td>LD [data]⇒r2</td>
<td></td>
</tr>
</tbody>
</table>

Two loads of the same address

Forbidden outcome observable on Cortex-A9
Our Approach

• Formal specifications -> Happens-before graphs
• Check Happens-Before Graphs via Efficient SMT solvers
  • Cyclic => A->B->C->A... Can’t happen
  • Acyclic => Scenario is observable
HLL <-> ISA <-> uArch: TriCheck Framework

- High-level Lang Litmus tests
- HLL Mem Model Sim
- Permitted/Forbidden
- Compare Outcomes

- HLL->ISA Compiler Mappings
- ISA Mem Model
- Observable/Unobservable

- ISA-level Litmus tests
- uArch Mem Model
- Permit: ok, Over strict
- Forbid: Bug, ok
HLL <-> ISA <-> uArch: TriCheck Framework

- High-level Lang Litmus tests
- HLL Mem Model Sim
- ISA Mem Model
- uArch Mem Model
- HLL->ISA Compiler Mappings
- ISA-level Litmus tests

Permitted/Forbidden

<table>
<thead>
<tr>
<th></th>
<th>Obs.</th>
<th>Not obs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Permit</td>
<td>ok</td>
<td>Over strict</td>
</tr>
<tr>
<td>Forbid</td>
<td>Bug</td>
<td>ok</td>
</tr>
</tbody>
</table>

Compare Outcomes

Iteratively Refine Design: HLL, Compiler, ISA, uArch

Observables/Unobservables
RISC-V Case Study

- Apply TriCheck approach to 7 distinct RISC-V implementation possibilities:
  - All abide by current RISC-V spec
  - Vary in preserved program order and store atomicity
  - TriCheck detects bugs: refine for correctness
  - TriCheck detects over-strictness

- Results:
  - Impossible to compile C11 for RISC-V as specified.
  - Out of 1,701 tested C11 programs:
    - RISC-V-Base-compliant design allows 144 buggy outcomes
    - RISC-V-Base+A-compliant design allows 221 buggy outcomes

Takeaway: Current RISC-V cannot serve as a compiler target for C11

Next Steps: RISC-V Memory Model Working Group formed to address these issues. Working to formalize a RISC-V memory model that supports C11, Linux, etc.
Computer Architecture’s Post-Moore Response

Technology Shifts: "Post-Moore"

Short Term: Specialization

Longer Term: Bio, Quantum

All: Need new Software, Abstractions
Quantum Computing: Fundamentally New Paradigm

Two key enablers:
- **Superposition** of states within a quantum bit (qubit)
- **Entanglement** of states between qubits

Potential for **exponential** speedups over classical
Why Quantum Computing?

- Fundamentally new paradigm, with probabilistic state based on superposition and entanglement
- Solve intractable problems: chemistry, simulation, optimization
- New industry and scaling curve to accelerate key applications
  - Not full Moore’s Law replacement, but helps in key places
- + More insights in classical computing

- “Quantum Annealers”
  - D-WAVE (2000 “qubits”)
  - UVA Cluster State Generator (3200 photonic qumodes)
  - Actual use of superposition/entanglement unclear

- “Universal”
  - IBM Quantum Experience: 16 qubits on the web
  - Intel, IBM: 49-50 qubit machines at CES last month.
  - Google: 72-qubit announcement at APS last week
  - Dozens of startups building QCs as well
Algorithms to Machines Gap

<table>
<thead>
<tr>
<th>Year</th>
<th>#Qubits Needed</th>
<th>#Qubits Buildable</th>
</tr>
</thead>
<tbody>
<tr>
<td>1995</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2005</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>2015</td>
<td>1000</td>
<td>1000</td>
</tr>
<tr>
<td>2025</td>
<td>1,000,000</td>
<td>1,000,000</td>
</tr>
</tbody>
</table>

- Grover's Algorithm (Database search)
- Shor's Factoring Algorithm (Crypto)
Algorithms to Machines Gap: HW Progress

• Qubit counts increasing
• Qubit coherence time improving: Now ~50 μsec
  • = Time before state collapses.
  • Schoelkopf’s Law: Trend of 3X every 10 years
• Microwave transmission of qubit signals
• QC HW expertise increasing
Algorithms to Machines Gap: Algorithm Progress

- Grover's Algorithm (Database search)
- Shor's Factoring Alg. (Crypto)
- Quantum Sim, Q Chem, QAOA

#Qubits

Year

1995 2005 2015 2025

- New breed of QC algorithm:
  - Lower qubit needs
  - Iterative with classical phases
  - Not exponential speedup, but promising demonstrations
- Hundreds of QC Algorithms in Quantum Zoo
- https://math.nist.gov/quantum/zoo/
How will Arch & Systems People Contribute?

~1950’s Classical Computing
- Algorithms
- Assembly Language
- Vacuum Tubes, Relay Circuits

Today’s Classical Computing
- Algorithms
- High-Level Languages
- Compiler
- OS
- Architecture
- Modular hardware blocks: Gates, registers
- VLSI Circuits
- Semiconductor transistors

Quantum Toolflows
- Algorithms
- High-level QC Languages, Compilers, Optimization, Error Correcting Codes
- Orchestrate classical gate control, Orchestrate qubit motion and manipulation.
- Qubit implementations
Why work on QC systems issues now?

- **System Design exploration:**
  - Tradeoffs in gate implementations
  - Qubit “movement” and communication approaches.
  - Connectivity topologies

- **Resource estimation:**
  - How many qubits do I need to do XYZ?

- **Tradeoffs in Error Correcting Code (ECC) approaches?**
  - For which algorithms/implementations/error rates, do different ECC approaches “win”?

- **Performance estimation:**
  - Impact of parallelism, scheduling, ECC on estimated runtime
Algorithms to Machines Gap: Our Past Work

#Qubits

- Grover's Algorithm (Database search)
- Shor's Factoring Alg. (Crypto)
- Quantum Sim, Q-Chem, QAOA

Our Past work

Gap!
Quantum Systems Design: Sweeping the ECC Design Space

- How to tailor QC error correction codes to a specific scenario?
- For different applications, problem sizes, device error rates, quantum technologies.
- Use compiler & scheduling techniques to answer!

[Javadi-Abhari et al. MICRO50]
QC programming and design tools that shrink the gap can move the feasibility point years sooner!

- Reduce algorithm qubit requirements
- Improve effectiveness of hardware qubits

<table>
<thead>
<tr>
<th>Year</th>
<th>#Qubits</th>
</tr>
</thead>
<tbody>
<tr>
<td>1995</td>
<td>1</td>
</tr>
<tr>
<td>2005</td>
<td>1000</td>
</tr>
<tr>
<td>2015</td>
<td>10000</td>
</tr>
<tr>
<td>2025</td>
<td>1000000</td>
</tr>
</tbody>
</table>

- Grover's Algorithm (Database search)
- Shor's Factoring Alg. (Crypto)
- Quantum Sim, Q Chem, QAOA

Our Future Work!
Newly Announced NSF Expedition in Computing Award

EPiQC: Enabling Practical-scale Quantum Computation

PI: Fred Chong

Co-PIs: Ken Brown, Eddie Farhi, Diana Franklin, Danielle Harlow, Aram Harrow, Margaret Martonosi, John Reppy, David Schuster, and Peter Shor

University of Chicago, MIT, Georgia Tech, Princeton, UCSB

https://www.epiqc.cs.uchicago.edu
Summary: Quantum Systems

- With QC hardware becoming more real, we need full-stack QC systems development as well:
  - Languages, simulators, debuggers...

- Full CS ecosystem needed to shift QC from theoretical to commercial

![Quantum Toolflows]

- **Algorithms**
  - High-level QC Languages.
  - Compilers.
  - Optimization.
  - Error Correcting Codes
  - Orchestrate classical gate control,
  - Orchestrate qubit motion and manipulation.

- **Qubit implementations**
New Golden Age of Computer Systems Design

Technology Shifts: "Post-Moore"
Short Term: Specialization
Longer Term: Bio, Quantum
All: Need new Software, Abstractions

Goal:
Stay out in front of technology shifts,
By developing and exploiting new compute elements and approaches
And by developing programming, compilation and OS techniques to support them.
Thanks!

- Students: Caroline Trippel, Yatin Manerkar, Prakash Murali
- Alumni: Ali Javadi-Abhari (now IBM), Dan Lustig (now NVIDIA)
- Other Collaborators: Michael Pellauer, Fred Chong, Diana Franklin, Ken Brown.
- Funding: NSF, DARPA, CFAR, NVIDIA

- mrm@princeton.edu
- http://www.princeton.edu/~mrm
- http://mrmgroup.cs.princeton.edu
- http://check.cs.princeton.edu
Quantum Systems: Layering Options

Classical Layering

- Algorithms
- High-Level Languages
- Compiler
- OS
- Architecture
- Modular hardware blocks: Gates, registers
- VLSI Circuits
- Semiconductor transistors

Quantum Toolflows

- Algorithms
- High-Level Languages
- Compiler
- Architecture
  - Architecture (Device-Independent)
  - Microarchitecture (Implen dependent)
- ECC
- Qubit implementations

Challenge:
Facing tight resource constraints, need more info flow up and down stack about device and algorithm characteristics.

Option 1
Quantum Systems: Layering Option 2

Classical Layering
- Algorithms
- High-Level Languages
- Compiler
- Architecture
- Modular hardware blocks: Gates, registers
- VLSI Circuits
- Semiconductor transistors

Quantum Toolflows
- Algorithms
- High-Level Languages
- App-specific Approaches
- App-specific Approaches
- Qubit implementations

Info flow up/down stack about:
- error characteristics,
- communication latencies
- Connectivity
- Gate decompositions
- App parallelism, resource requirements
- + other characteristics ...

Think of TensorFlow and TPU...
Conclusions & What’s next?

- QC is NOT a Moore’s Law replacement
  - Unique, special-purpose hardware
  - Focused applications
- But potentially game-changing
  - Make untractable tractable
  - Lessons learned (algs, systems, devices) drive innovation on classical side as well
- With QC hardware becoming more real, full CS ecosystem needed to shift QC from theoretical to commercial
  - Languages, simulators, debuggers...