# Language, Compiler, and Optimization Issues in Quantum Computing

**Margaret Martonosi**
Princeton University

**Fred Chong**
**Diana Franklin**
**Jeff Heckey**
**Daniel Kudrow**
UC Santa Barbara

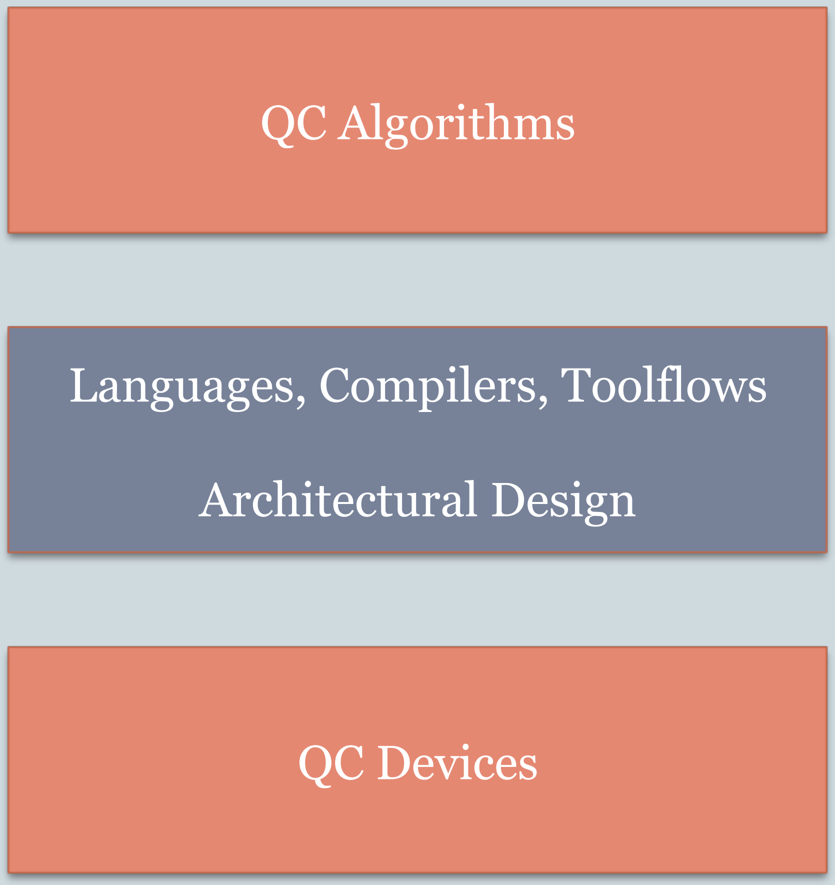
**Ali JavadiAbhari**
**Shruti Patil**
Princeton University

**Kenneth R. Brown**
Georgia Tech

**Adam Holmes**
Cornell University

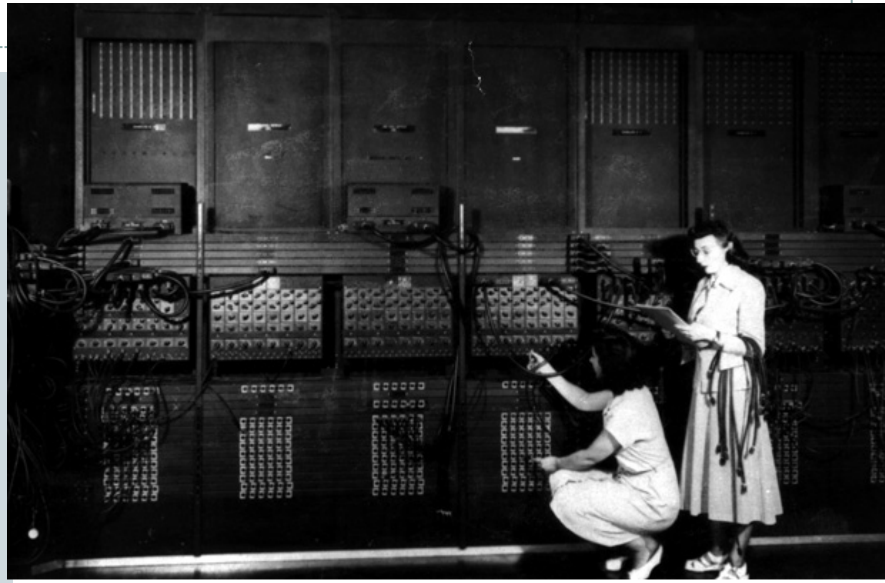
# Quantum Computing at a Crossroads

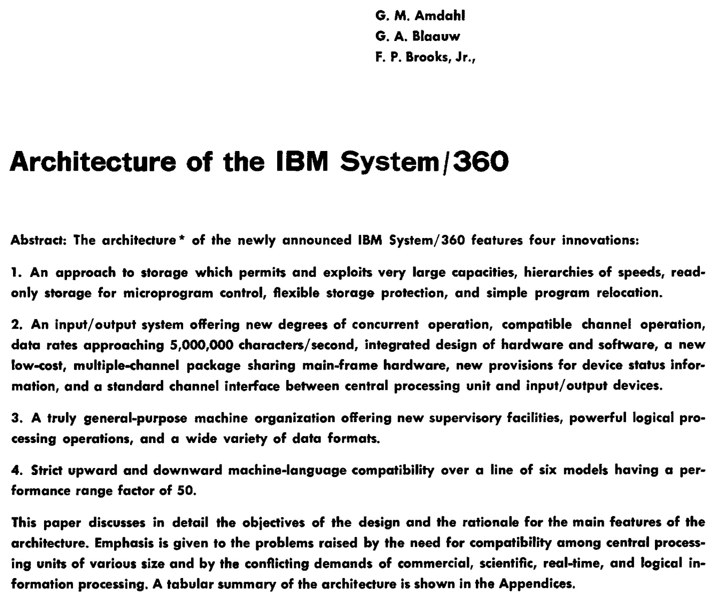
- QC Ten years ago:
  - High level algorithms
  - Low level devices
  - Little in between
- Today:
  - Increasing attention at languages and toolflows to connect top to bottom.
- Why?
- And why “regular” architecture, language, and compiler researchers?



# Analogy: Early Classical Computing

- No abstractions, no system layering.
  - Just people with big problems to solve
  - And people who could build machines.
- 1964: Instruction Set Architecture is born.
  - ISA = Fundamental abstraction: Allows same software to run on different implementations.
  - Gives software stable target. Allows hardware to optimize itself under stable abstraction layer.
  - Supports independent layers of optimization and analysis.
- Now: Time to begin similar layerings for QC.
  - Should be a collaboration of QC people and classical computer systems researchers.





G. M. Amdahl
G. A. Blaauw
F. P. Brooks, Jr.,

**Architecture of the IBM System/360**

**Abstract:** The architecture* of the newly announced IBM System/360 features four innovations:

1. An approach to storage which permits and exploits very large capacities, hierarchies of speeds, read-only storage for microprogram control, flexible storage protection, and simple program relocation.

2. An input/output system offering new degrees of concurrent operation, compatible channel operation, data rates approaching 5,000,000 characters/second, integrated design of hardware and software, a new low-cost, multiple-channel package sharing main-frame hardware, new provisions for device status information, and a standard channel interface between central processing unit and input/output devices.

3. A truly general-purpose machine organization offering new supervisory facilities, powerful logical processing operations, and a wide variety of data formats.

4. Strict upward and downward machine-language compatibility over a line of six models having a performance range factor of 50.

This paper discusses in detail the objectives of the design and the rationale for the main features of the architecture. Emphasis is given to the problems raised by the need for compatibility among central processing units of various size and by the conflicting demands of commercial, scientific, real-time, and logical information processing. A tabular summary of the architecture is shown in the Appendices.

## QC Architecture & Compiler Research: Goals

- Identify and compare the viability of proposed technologies
  - Quantify physical bounds and hardware characteristics
  - Alert physicists of technological limits that are needed for computationally relevant implementation.
- Identify the Unknowns
  - Scaling to arbitrary sizes compounds challenges
- Identify correct microarchitectural abstractions

Necessarily Multidisciplinary:

Computer Engineers + Algorithmicists + Physicists

# QC Architecture & Compiler Challenges

Algorithms vs. Benchmarks:

- Few diverse and large-scale benchmarks exist —> write our own in our own high-level language.

Tools:

- Few openly-available tools for compilation and analysis of large QC programs
- Code/data specialization common in QC -> Results in massive compilation files and memory usage

QC impact on compilation/ computation:

- No cloning theorem -> Many data dependencies & long serial computation chains -> low parallelism. How to address?
- Long serial chains: Data dependences, rotation decomposition, ... cause lots of qubit movements -> Mitigating communication cost?

Technology Impacts:

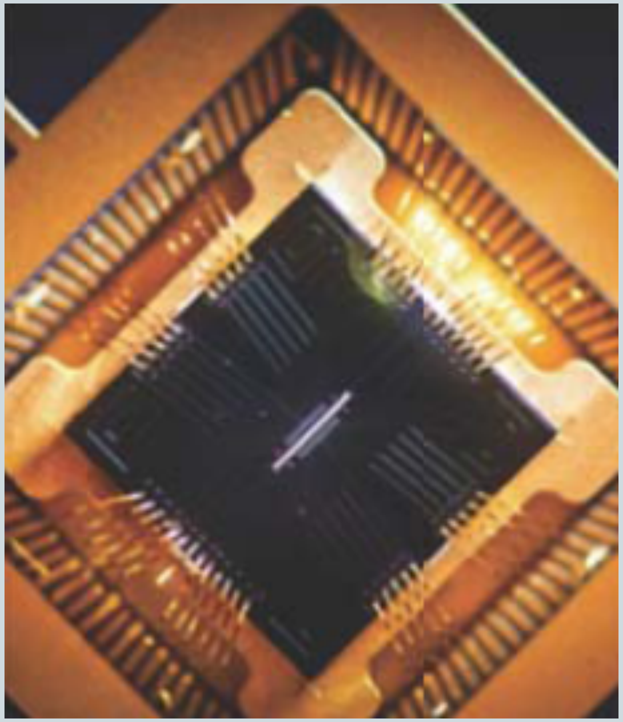
- eg Quantum Teleportation vs. Ballistic motion. -> Intelligent memory hierarchy design.

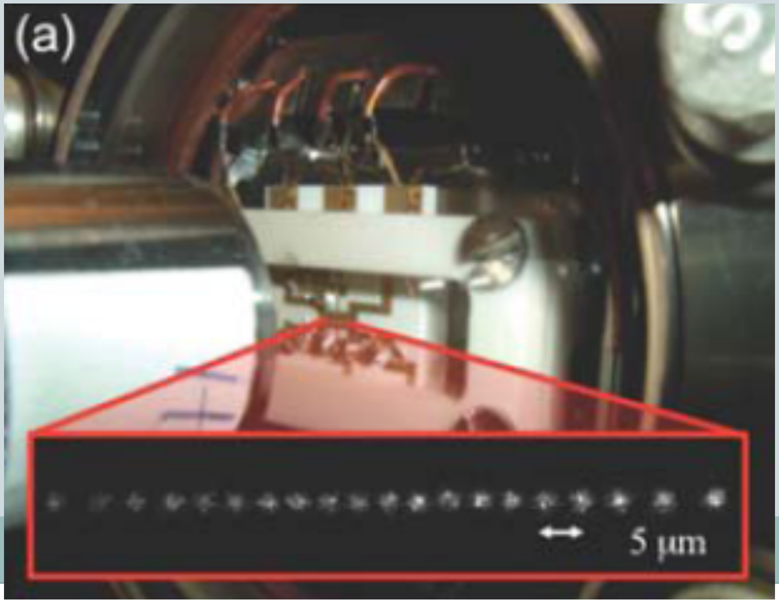
## This Talk

- Background & Basics
- Focus 1: Scalable Tailored Compilation
- Focus 2: QC Communication and Scheduling Issues
- Focus 3: QC Language design and evolution

# Quantum Device Technology

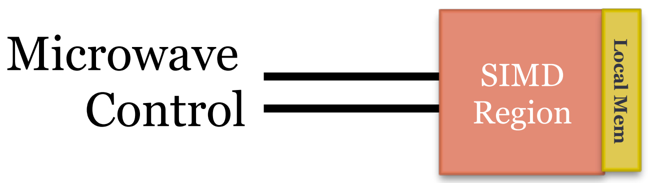
- Our Main Focus: Ion Trap Technology
  - Good experimental understanding
  - Microwave control
  - Allows for "SIMD" operation: Multiple ion, single control





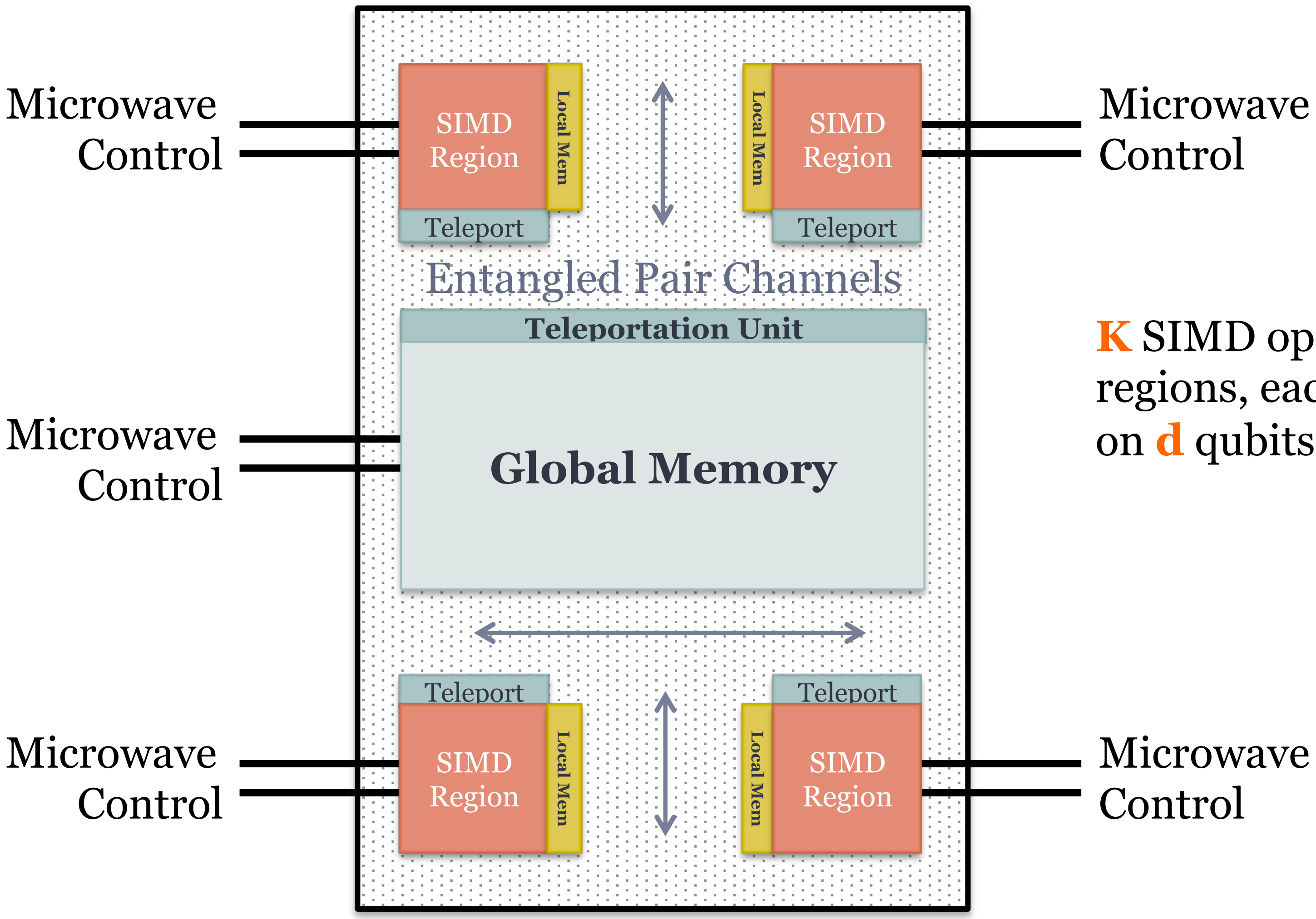
- But plan to broaden to other technologies too
  - Superconducting, QDOT…

# SIMD Operating Regions



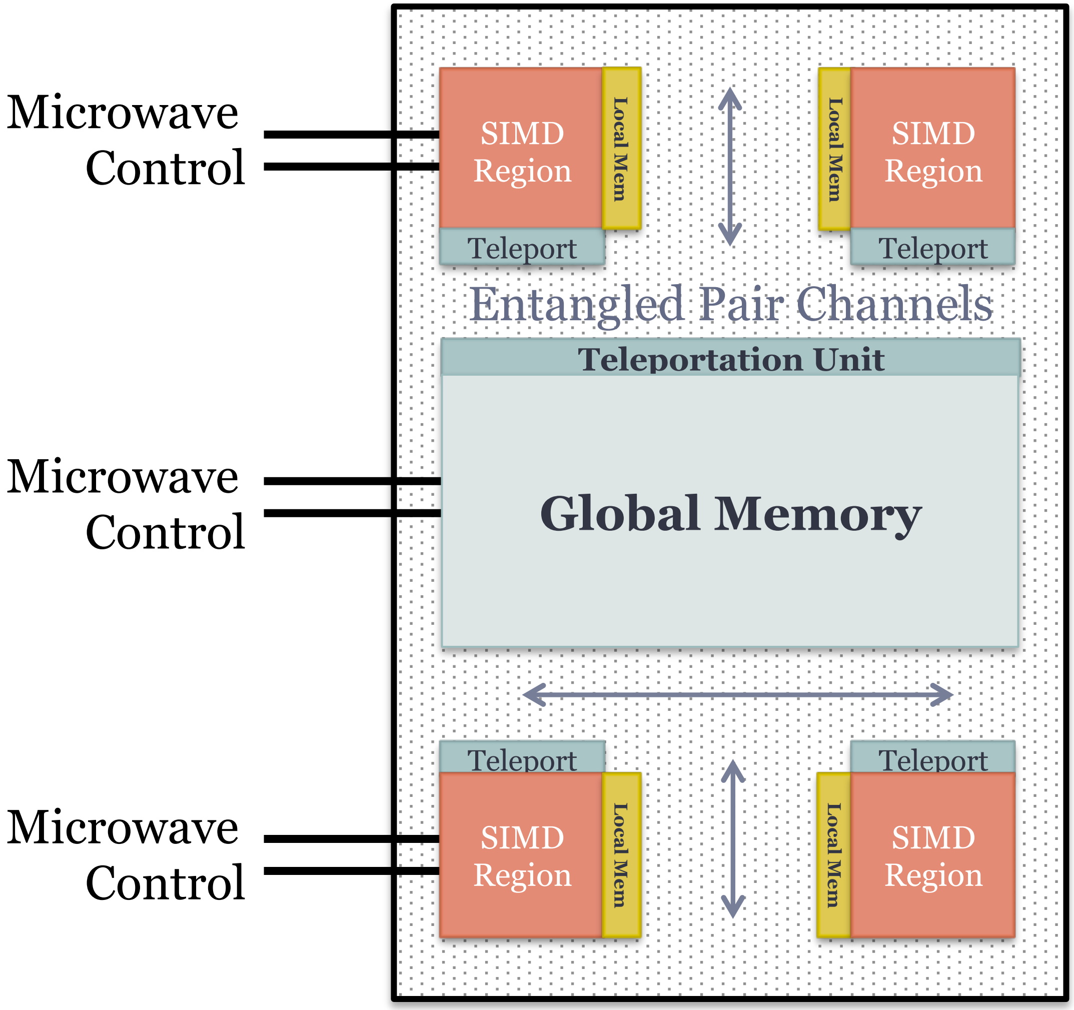
- SIMD: Single-Instruction, Multiple-Data
- Can apply the same gate operation (H, CNOT, etc) to many qubits at once.
  - Capacity (d): Few to 100's
  - K=2-8 SIMD regions are useful for QC apps we've studied

# Computational Architecture: Multi-SIMD (k,d)



**K** SIMD operation regions, each operating on **d** qubits per cycle

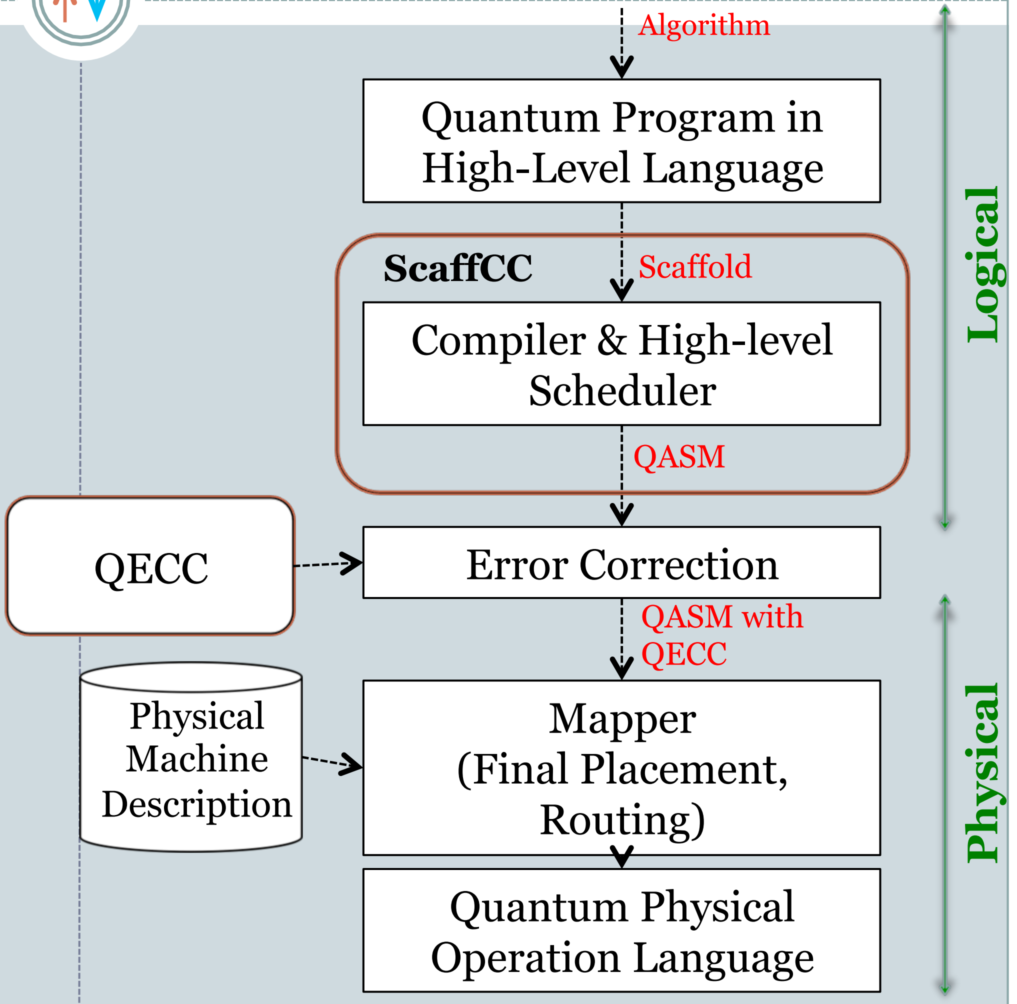
# Scheduling Challenge



- Primary Goal: Schedule moving qubits in/ out of SIMD regions to maximize parallelism & minimize communication
- Also: Manage tradeoffs between ballistic and teleportation comm, balance storage requirements, …

# Compiling Quantum Codes: Our Scaffold/ScaffCC Toolflow

- Data types and instructions in quantum computers:
  - Qubits, quantum gates
- Decoherence requires QECC
  - Logical vs. Physical Levels
- Efficiency crucial
  - Inefficiencies at logical level are amplified into greater physical level QECC requirements.
  - Optimizations performed at logic level can be more tractable and have high leverage.



Algorithm
Quantum Program in High-Level Language
ScaffCC
Scaffold
Compiler & High-level Scheduler
QASM
QECC
Error Correction
QASM with QECC
Physical Machine Description
Mapper (Final Placement, Routing)
Quantum Physical Operation Language
Logical
Physical

## Benchmarks

| Benchmark | Size | Lines of Code | Ops | Min Logical Qubits |
|---|---|---|---|---|
| **Grover's Search** | n = 40 | 208 | 2.4E+09 | 120 |
| **Binary Welded Tree** | n = 300<br>s = 3000 | 482 | 2.9E+09 | 2719 |
| **Ground State Estimation** | m = 10 | 9643 | 9.74E+07 | 13 |
| **SHA-1 Reversal** | n = 128 | 855 | 1.02E+11 | 486536 |
| **Shor's Factorization** | n = 512 | 1055 | 9.80E+10 | 5634 |
| **Triangle Finding Problem** | n = 5 | 4052 | 5.24E+09 | 176 |
| **Boolean Formula** | x = 2<br>y = 3 | 693 | 3.1E+08 | 1711 |
| **Class Number** | p = 6 | 383 | 3.5E+06 | 60052 |

## This Talk

- Background & Basics
- **Focus 1: Scalable Tailored Compilation**
- Focus 2: QC Communication and Scheduling Issues
- Focus 3: QC Language design and evolution

## Scalable Tailored QC Compilation

Quantum circuits often specialized to one problem input or size:

**Benefits of Customization:**

Efficient circuits. Deeply and statically analyzable.

**Vs. Lack of Scalability:**

Code explosion: > $10^{12}$ ops for some applications!

- Need better balance of optimization and scalability
  - QASM format changes: QASM-H, QASM-HL: 200,000X or more code size savings
  - Modular analysis
  - Memoization and Instrumentation-driven analysis

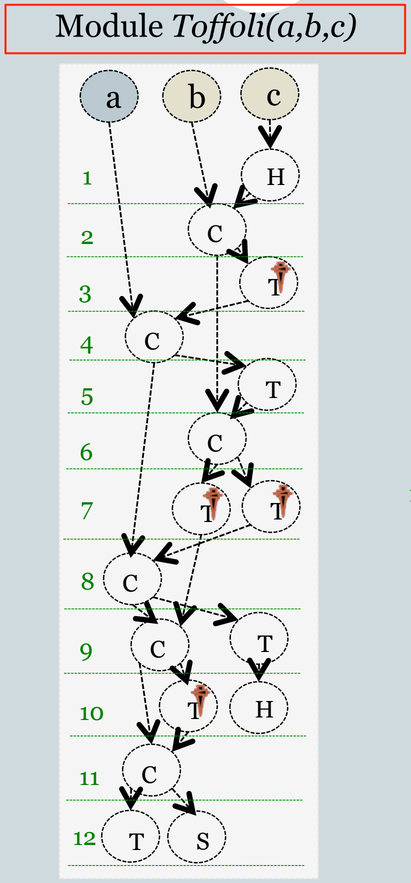
## Critical Path Estimates & Modular Analysis

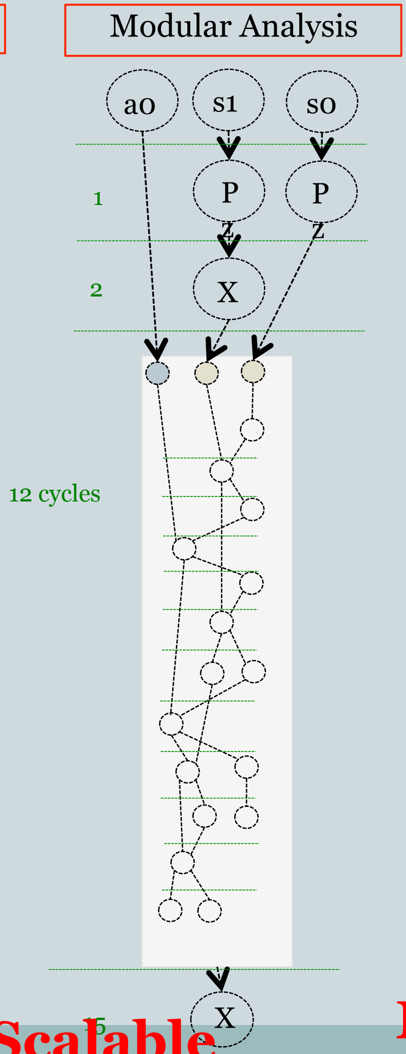
- Scheduling based on qubit data dependences:
  - Many compilation optimizations rest on critical path estimates.
- Intractable as whole-program analysis =>
  - Use Hierarchical / Modular techniques
  - Obtain module critical paths separately and then treat them as black boxes.

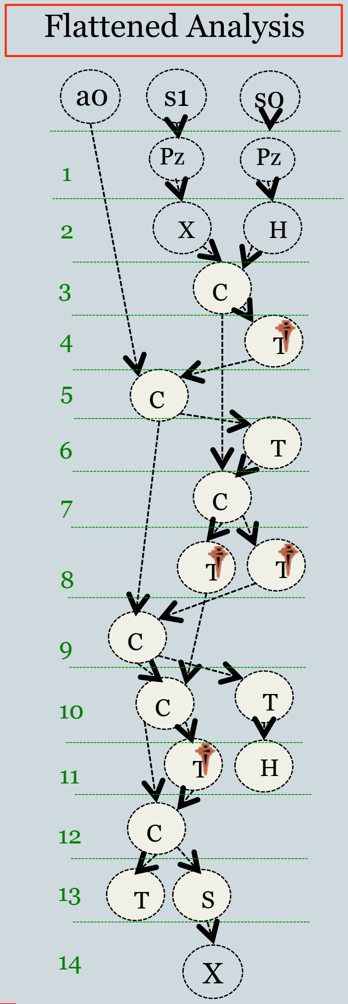
## Hierarchical Approach Improves Scalability, but Loses Optimality at Module Boundaries

```
PrepZ(s0)
PrepZ(s1)
X(s1)
Toffoli(a0,s1,s0)
X(s1)
. . .
```

- Closeness to actual critical path is dependent on the level of **modularity**
- Flatter overall program means more opportunity for discovering parallelism





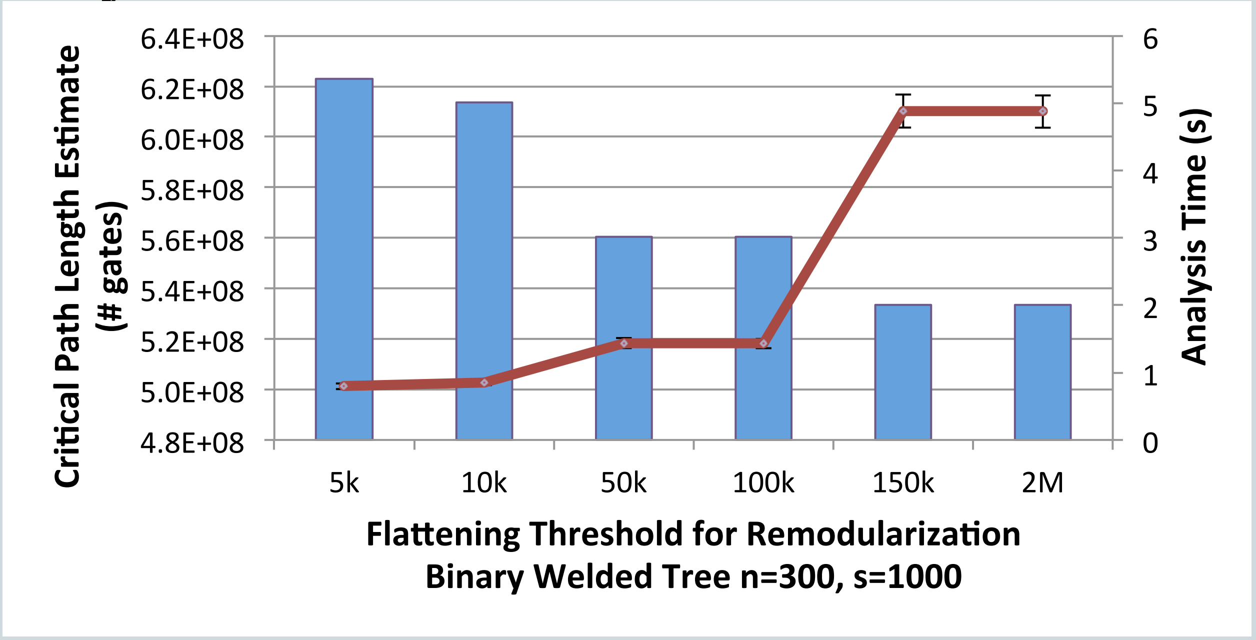


**Scalable**

**More Accurate**

# Effect of Remodularization

- Based on resource analysis, flatten modules with size less than a threshold
- Tradeoff between speed of analysis and its accuracy



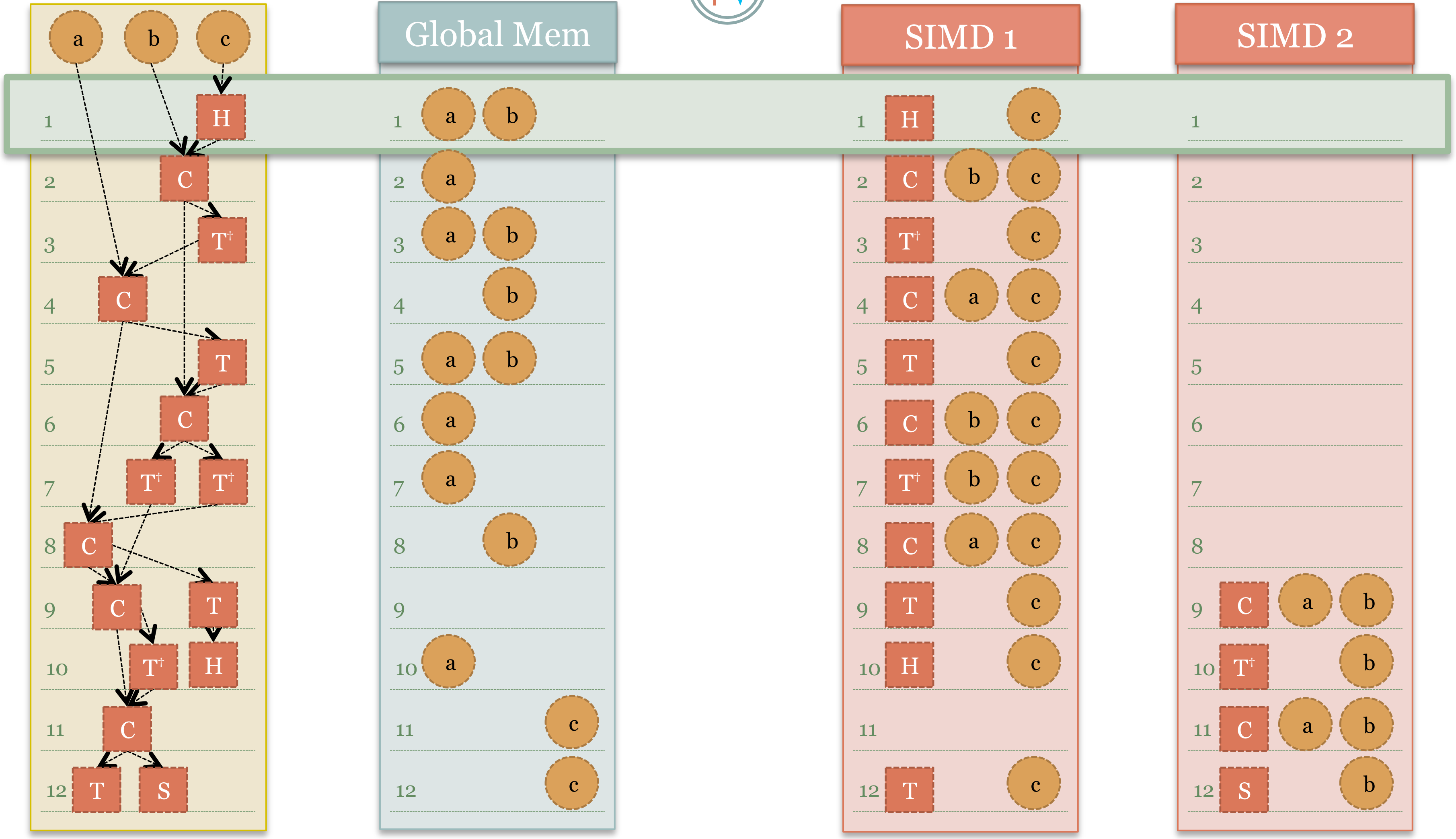
# Scalable Tailored Compilation: Summary

- Extended LLVM's classical framework for quantum compilation at the logical level
- Managed scalability through:
  - QASM Output format:
    - 200,000X on average + up to 90% for some benchmarks
  - Code generation approach:
    - Up to 70% for large problems
- For more info, see our ScaffCC papers:
  - Computing Frontiers (CF) 2014: ScaffCC overview.
  - IISWC 2014: Trials & Rotations in Quantum Phase Estimation
  - J. Parallel Computation: ScaffCC long version.
  - ASPLOS 2015: Communication optimizations.

## This Talk

- Background & Basics
- Focus 1: Scalable Tailored Compilation
- **Focus 2: QC Communication and Scheduling Issues**
- Focus 3: QC Language design and evolution

# Why Communication Time Matters



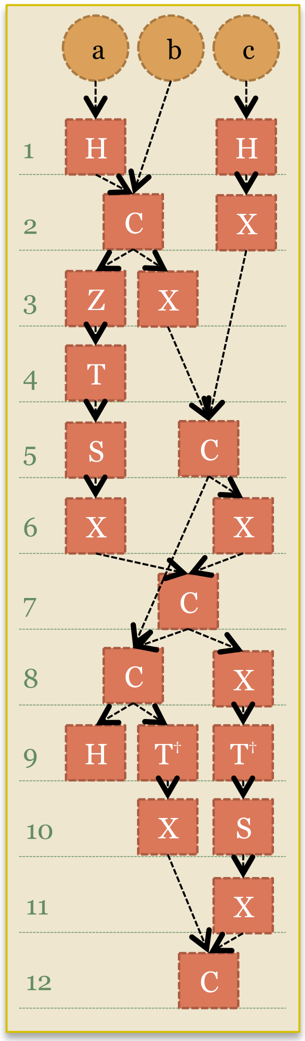
# Longest Path First Scheduling

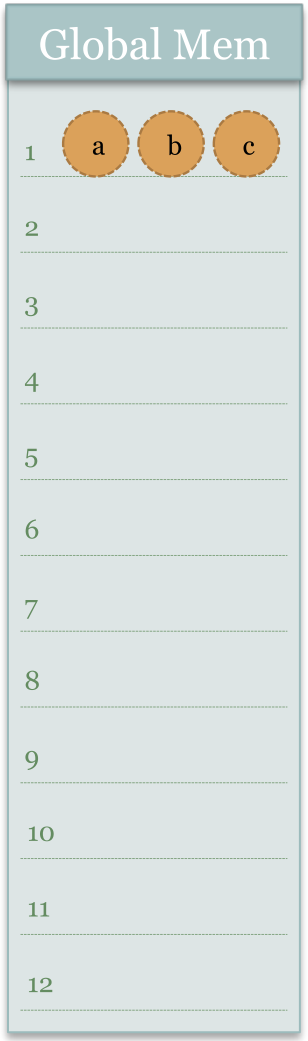
Strategy: Minimize qubit motion by assigning long dependence chains to a single SIMD unit, where they can compute locally with little communication.

Approach:

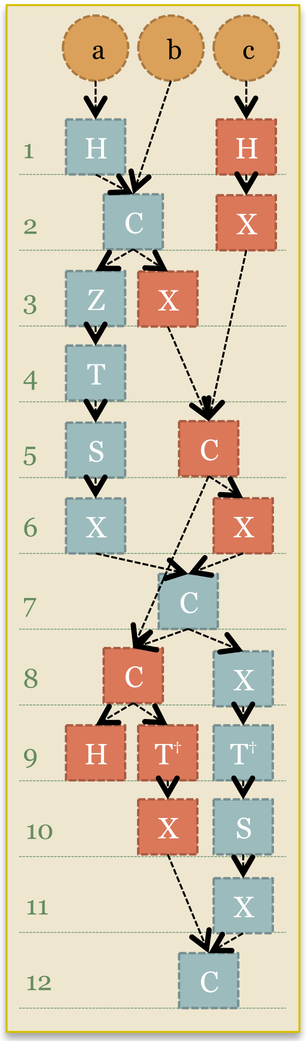
- Find $l$ longest paths
- Assign to $l$ SIMD regions
- Assign remaining operations to $k - l$ SIMD regions
  - Optionally: schedule any same operations to one of $l$ SIMDs
- Designed for arbitrary rotation decompositions

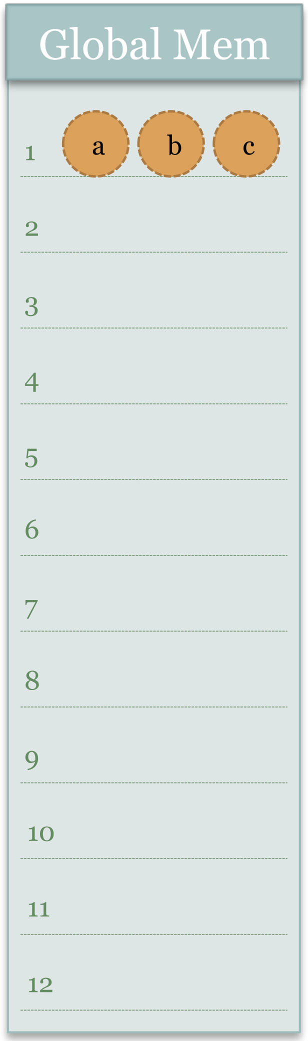
# Longest Path First Scheduling

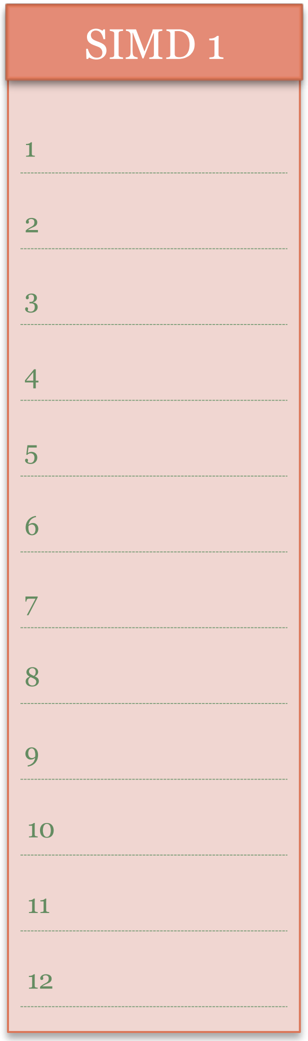


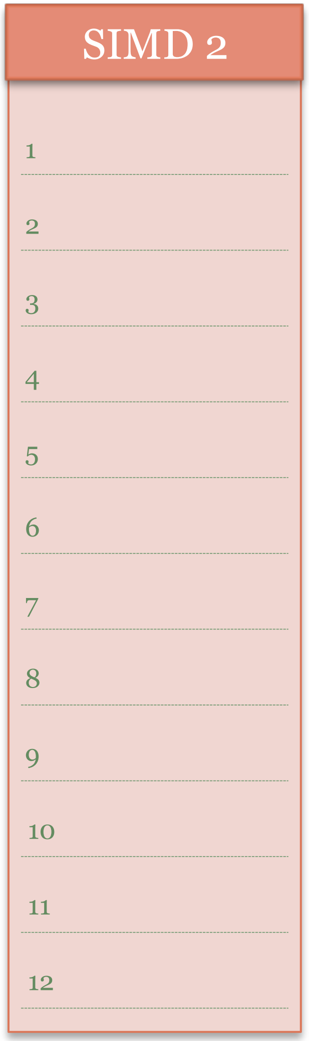


# Longest Path First Scheduling

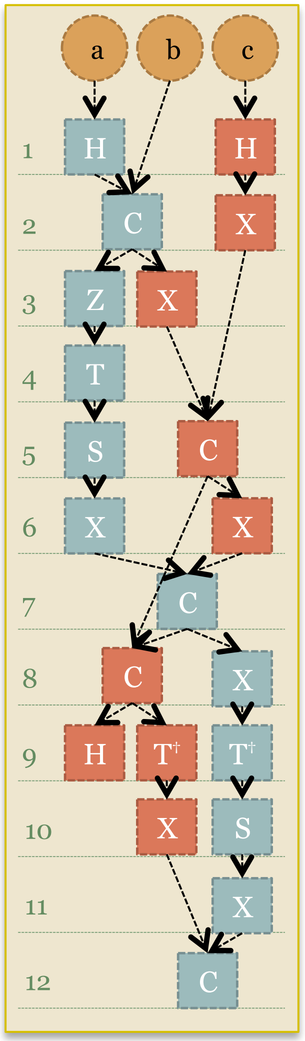






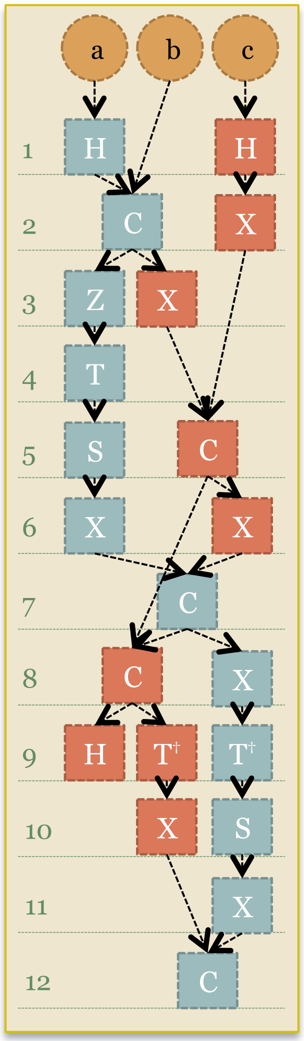


# Longest Path First Scheduling



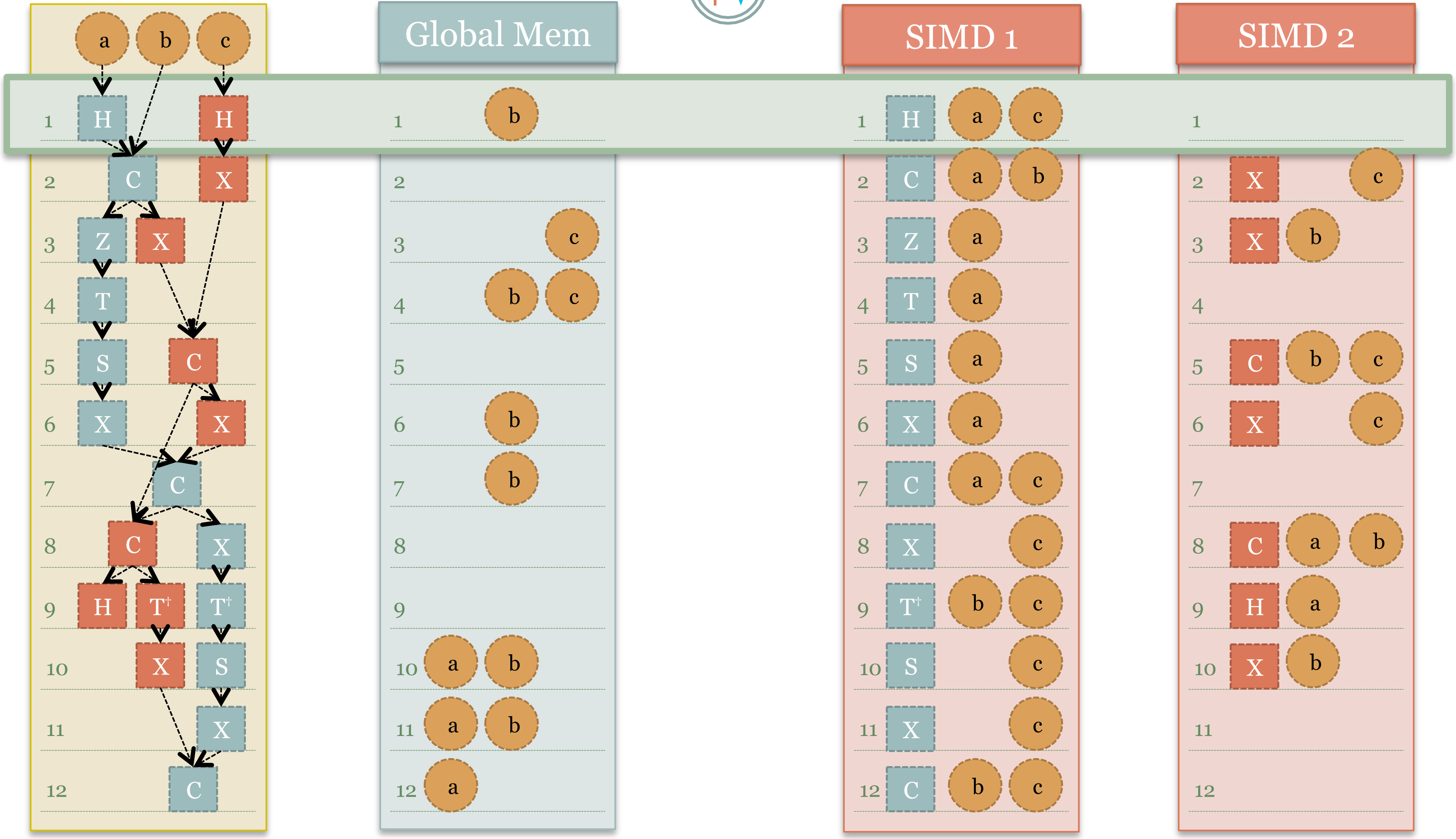
| | Global Mem | SIMD 1 | SIMD 2 |
|---|---|---|---|
| 1 | b c | H a | |
| 2 | c | C a b | |
| 3 | b c | Z a | |
| 4 | b c | T a | |
| 5 | b c | S a | |
| 6 | b c | X a | |
| 7 | c | C a b | |
| 8 | a b | X c | |
| 9 | a b | T† c | |
| 10 | a b | S c | |
| 11 | a b | X c | |
| 12 | a | C b c | |

# Longest Path First Scheduling

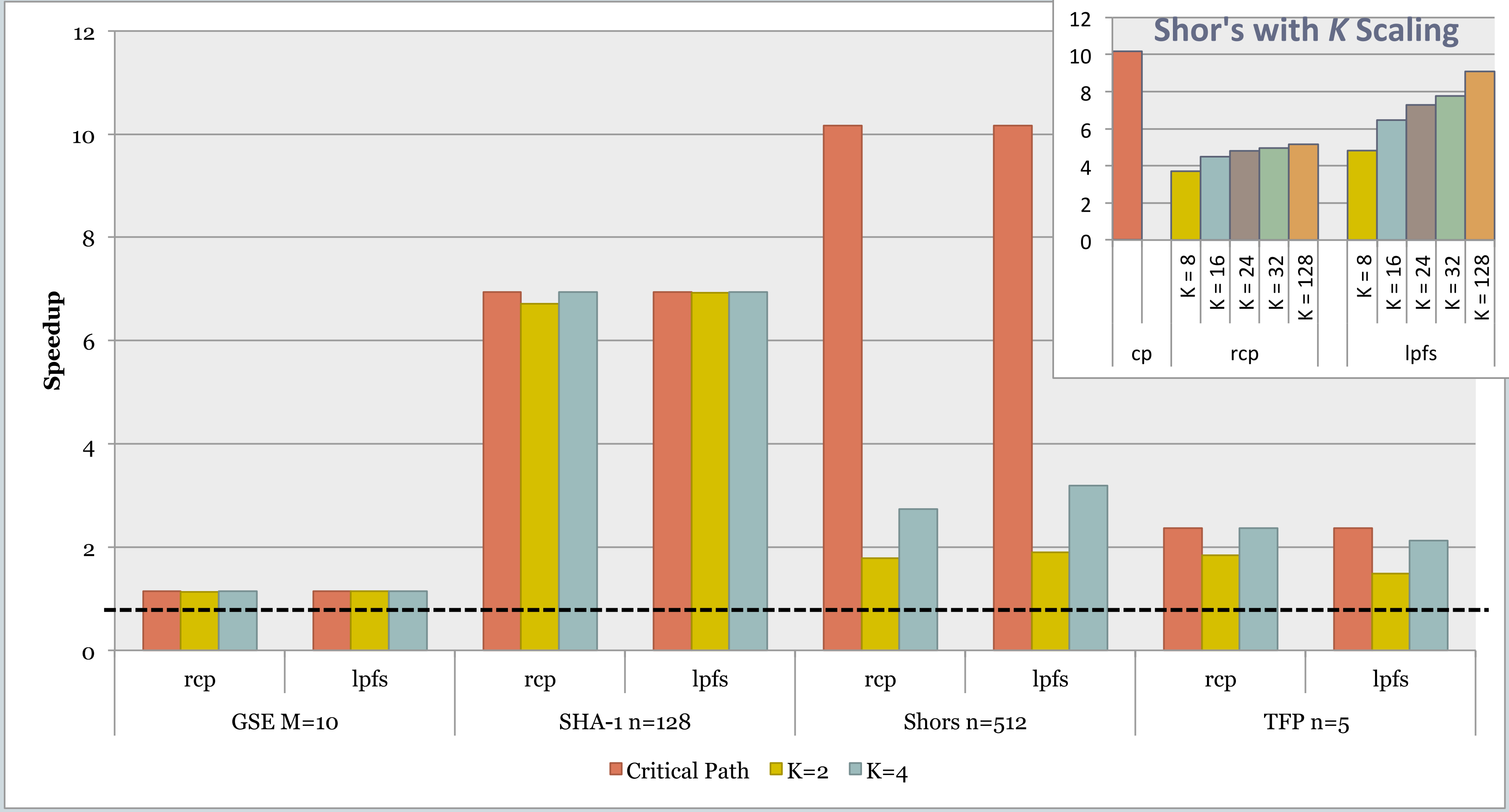


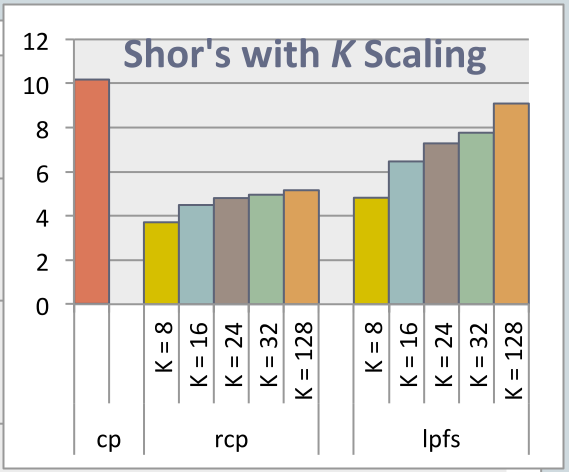
| | Global Mem | SIMD 1 | SIMD 2 |
|---|---|---|---|
| 1 | b | H a c | |
| 2 | | C a b | X c |
| 3 | c | Z a | X b |
| 4 | b c | T a | |
| 5 | | S a | C b c |
| 6 | b | X a | X c |
| 7 | b | C a c | |
| 8 | | X c | C a b |
| 9 | | T† b c | H a |
| 10 | a b | S c | X b |
| 11 | a b | X c | |
| 12 | a | C b c | |

# Longest Path First Scheduling

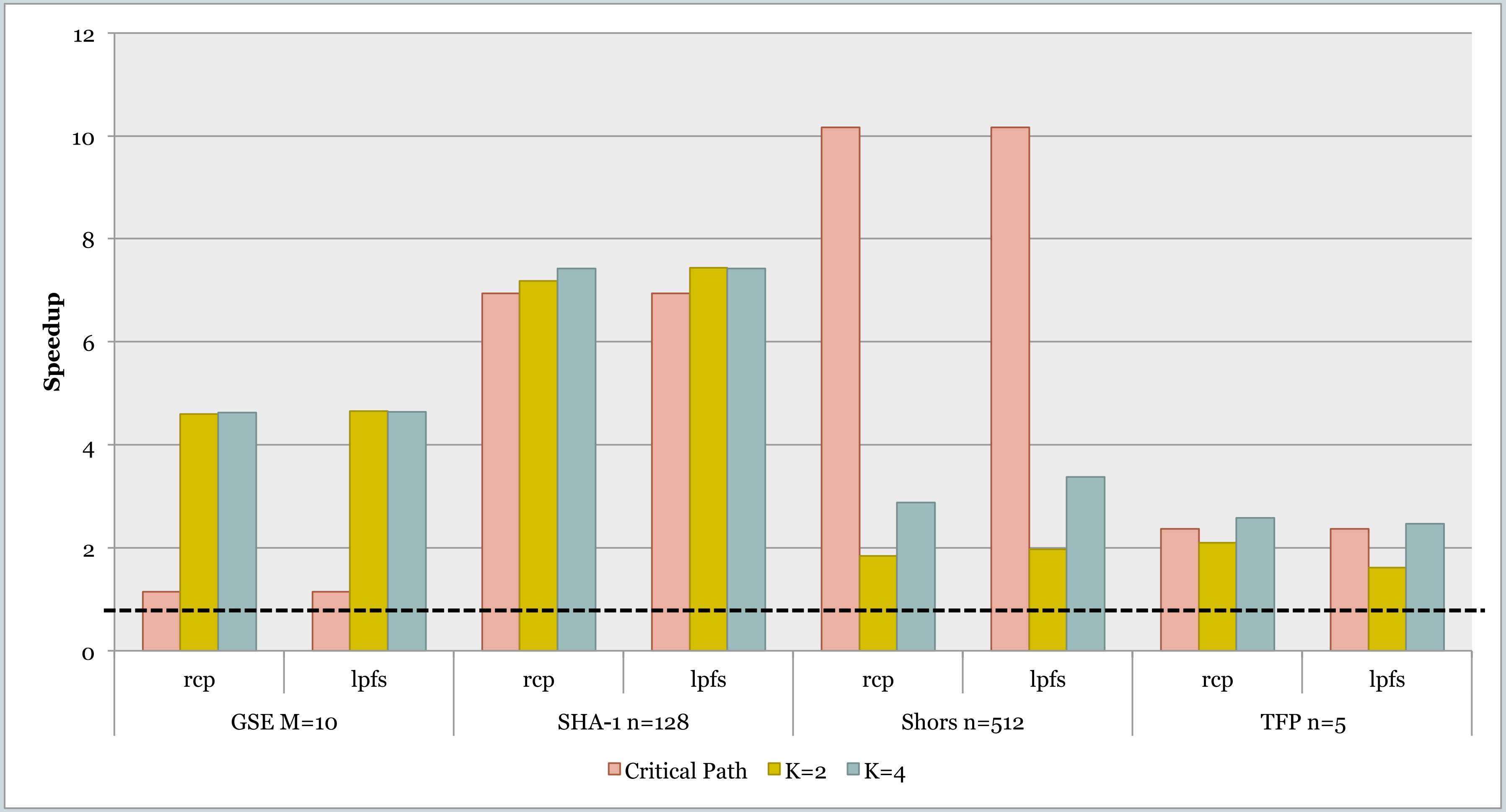


# Performance: Movement Unaware

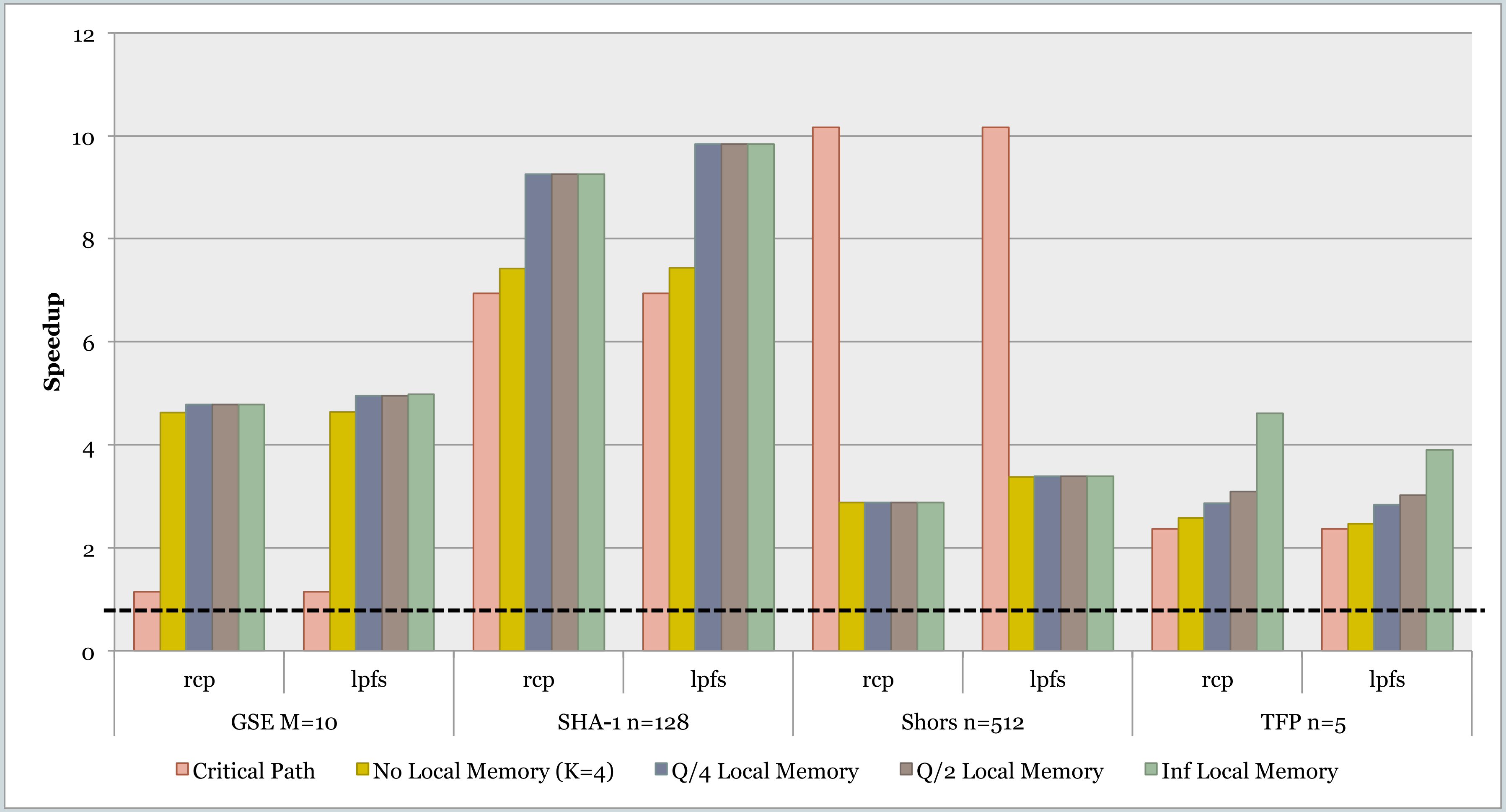




# Performance: Movement Aware



# Movement Aware with Local Memory



## Communication Optimization: Summary

- Architecture: Multi-SIMD with local memory architecture shows viable performance
  - Drawing from classical architectural techniques for QC performance improvements
- Compiler/Communication: Intelligent scheduling has high leverage
  - 2.3x- 9.8x in the best case
  - Logic-level (pre-QECC) scheduling limits qubit counts to improve tractability. Post-QECC schedules follow directly.
- Current/Future work: Fine-grained qubit orchestration. EPR (Bell) Pair scheduling for quantum teleportation…

## This Talk

- Background & Basics
- Focus 1: Scalable Tailored Compilation
- Focus 2: QC Communication and Scheduling Issues
- **Focus 3: QC Language design and evolution**

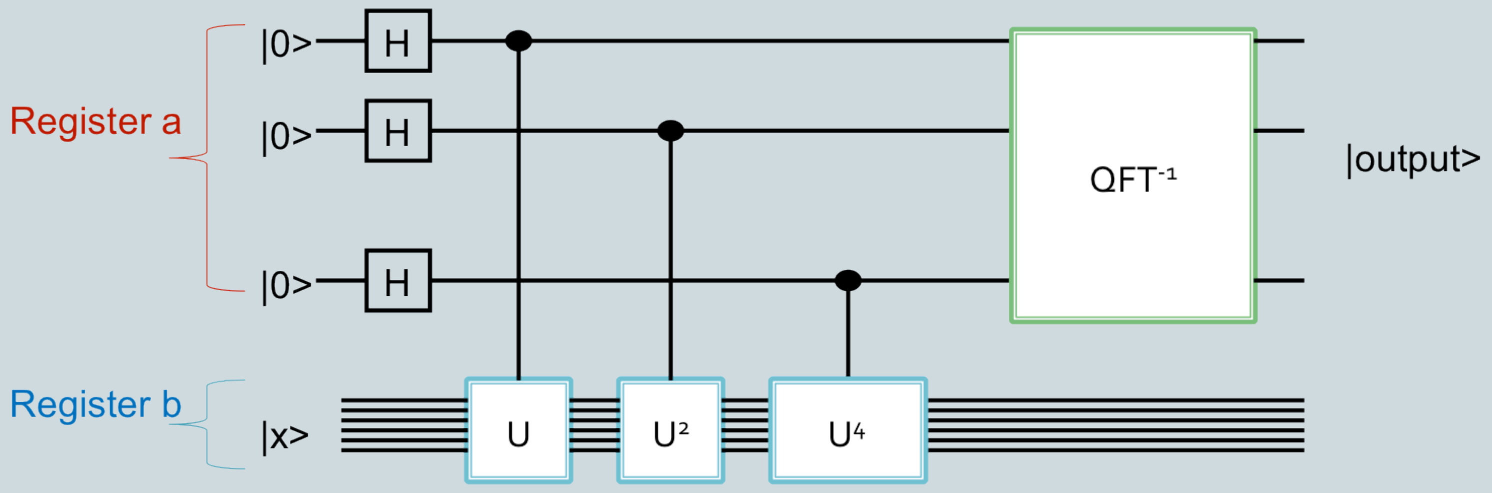
# Looking forward: QC Language Design

- Scaffold is based on C…
  - And like C, it emphasizes low-level functional orchestration over higher-level abstraction or analysis.
  - Good for mapping onto QASM and variants.
  - But…
- A QC Programming Language should support and automate analysis techniques prioritizing those aspects of QC that are ***particularly difficult***.
  - Verification, simulation, assertion checking
  - Error models and ECC support…

## How do I know if my QC program is correct?

- Need: Specification language for QC algorithms
- Check implementation against the specification
  - Simulation for small problem sizes (~30 qubits)
  - Symbolic execution for larger
  - Type systems
  - Model checking
  - Certified compilation passes
- Compiler checks general quantum properties
  - No-cloning, entanglement, uncomputation
- Checks or compiles based on programmer assertions too, where possible

# Programmer Assertion Example



```
b_eig_U = Eigenvalue(b,U)
CascadeU(a,b,U)
if not(b_eig_U)
   assert(Entangled(a,b))
```

## Success Probabilities / Error Bounds

- Quantum operations are approximate (eg rotations)
  - Need to track achieved precision
- Quantum programs often involve multiple trials
  - Assume error probability is low enough for success in small number of trials
- Type system that tracks probabilities
  - Static analysis when possible
  - Symbolic execution when necessary

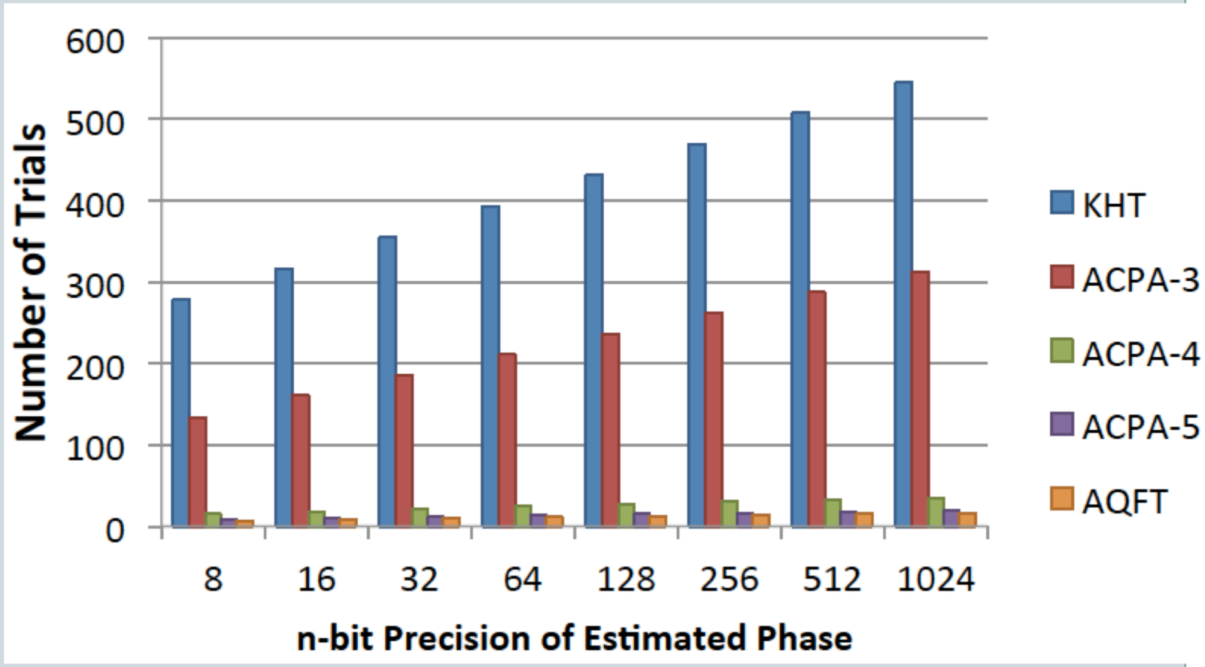
measure(a)

assert(precision(a, 8)) /**precision of* ***a*** *is at least* $10^{-8}$ **/*

assert(error(a, 0.5)) /**probability of error in* ***a*** *< 0.5* **/*

# Precision vs. Runtime Optimizations

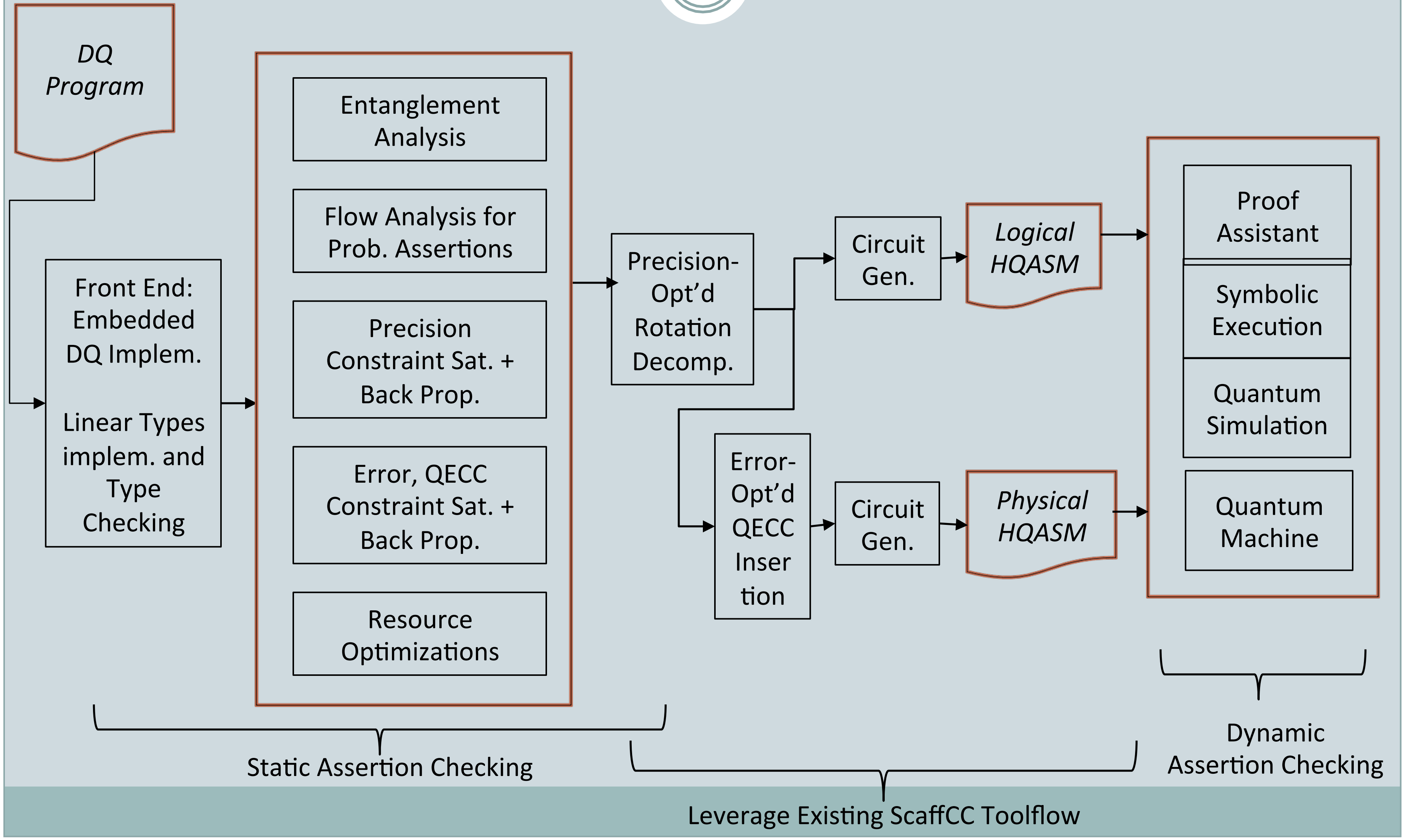
- Precision-optimized operations, paired with programmer-provided precision requirement assertions.
- Example: Select quantum phase estimation approach and number of trials, based on precision assertions provided by programmer.



# Putting it all together: DQ: Dependable Quantum Programming Language

- Improve analysis and verification support up front: Embedded, high-level front-end language
  - Expressibility of algorithms,
  - Verification of program correctness
  - Annotations of precision and error bounds
- Retain scalability and optimization support in backend: Lower-level backend language with industrial-strength, scalable analysis tools
- Type system for verifying quantum properties and calculating success probabilities / errors
- Longer term:
  - Verification or direct code generation using Unitary Transforms
  - Precision vs. runtime optimizations.

# DQ Toolchain Overview


*DQ Program*
Front End: Embedded DQ Implem.
Linear Types implem. and Type Checking
Entanglement Analysis
Flow Analysis for Prob. Assertions
Precision Constraint Sat. + Back Prop.
Error, QECC Constraint Sat. + Back Prop.
Resource Optimizations
Precision-Opt'd Rotation Decomp.
Circuit Gen.
*Logical HQASM*
Error-Opt'd QECC Insertion
Circuit Gen.
*Physical HQASM*
Proof Assistant
Symbolic Execution
Quantum Simulation
Quantum Machine
Static Assertion Checking
Leverage Existing ScaffCC Toolflow
Dynamic Assertion Checking

## Other Work

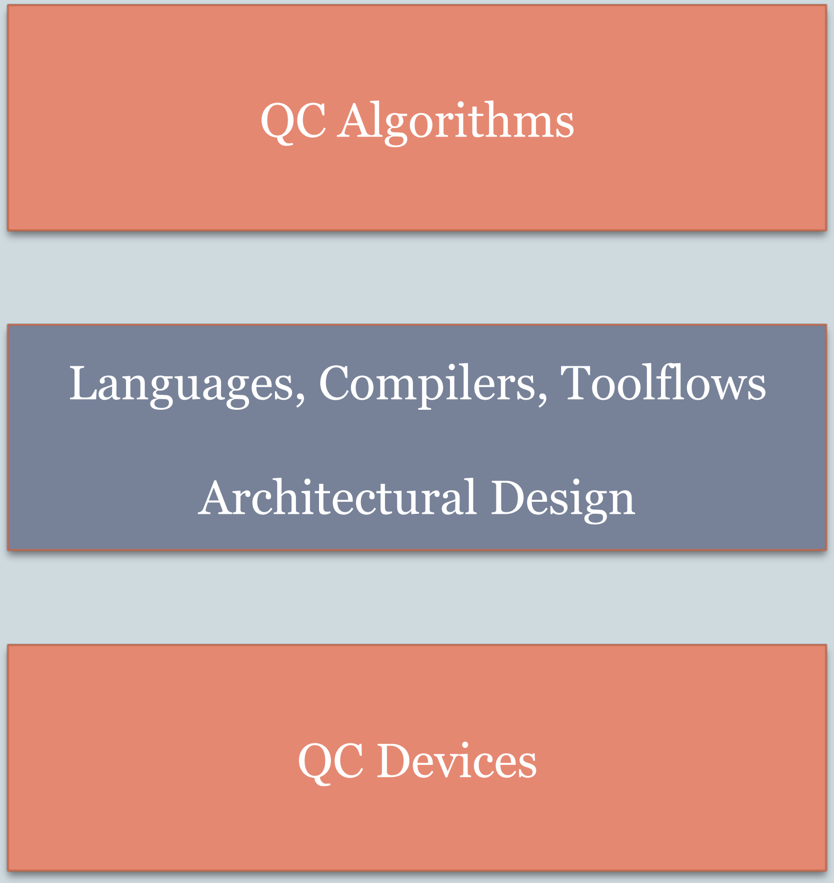
- Finer-grained Communication Management
  - EPR distribution network underlying teleportations. Bandwidth and scheduling?
  - Decoherence and purification rate of qubits.
- Balanced Communication/Storage optimizations
  - Communication-Avoiding Algorithms localize compute in particular regions, which imbalances storage requirements.
  - Current/Future work: Balance qubit storage/communication requirements.

# Conclusions

Great need for research and toolflows between QC algorithms and devices.

As tools are developed, layers become clearer:

- High-level languages for verification and analysis: DQ
- Mid-level languages for scalable optimization and communication scheduling: Scaffold and ScaffCC
- Toolflows and benchmarks drive research to clarify machine organizations and memory hierarchies: Multi-SIMD



## Acknowledgements

- My co-authors on several papers:
  - Students: Ali JavadiAbhari, Adam Holmes, Jeff Heckey
  - Post-docs and Profs: Shruti Patil, Fred Chong, Diana Franklin, Ken Brown.
- Other contributing researchers:
  - Alexey Lvov,  John Black, Lukas Svec, Aram Harrow, Amlan Chakrabarti, Chen-Fu Chiang, Oana Catu, and Mohammed Javad Dousti
- Funding Agencies:
  - NSF PHY-1415537, IARPA, …

- For more Info:
  - Computing Frontiers (CF) 2014: ScaffCC overview.
  - IISWC 2014: Trials & Rotations in Quantum Phase Estimation
  - J. Parallel Computation: ScaffCC long version.
  - ASPLOS 2015: Communication optimizations.
  - http://mrmgroup.cs.princeton.edu