ATPL 2024: Hybrid Quantum-Classical Programming.

Lecture 1: Why do we need hybrid programming models?

James Avery (avery@diku.dk)

November 18, 2024

Outline

Table of Contents

Week 1-3: Overview Lectures

18/11	Introduction to Hybrid Quantum Computing (James)
19/11	Quantum bits: Calculi and languages (Michael)
25/11	Multilinear algebra and tensors algebra (Fritz)
26/11	Functional programming for hybrid computing (Martin)
2/12	Clifford Algebra and separating mixed programs (James)
3/12	Parallel functional programming (Martin)

Week 4: Project preparation

- Pair up in groups.
- Select project topic and select literature.
- Read and discuss the literature.
- Prepare seminar presentation and student exercises.

Weeks 5-7: Seminars and invited talks

First hour will be student seminars, second hour will be invited talks:

12/12	Jaco van de Poel, Aarhus University (Optional, outside schedule)
16/12	Sven Carlsson, DTU
17/12	Raphael Seidel, Fraunhofer FOKUS
6/1	Mark Jones, Molecular Quantum Solutions
7/1	Mohammad Mousavi, King's College London

Weeks 9-12: Project work

	Work on project in groups.
	Weekly discussions with supervisor.
20/1	Hand in final project report.
30/1	Oral exams.

Table of Contents

Reversible programs permute the state space

A classical reversible program with a fixed number of bits acts as a *permutation* on the bit state space:

$$\underbrace{\begin{array}{c|cccc} & 00 & 01 & 10 & 11 \\ \hline 00 & 1 & 0 & 0 & 0 \\ 01 & 0 & 1 & 0 & 0 \\ 10 & 0 & 0 & 0 & 1 \\ 11 & 0 & 0 & 1 & 0 \end{array}}_{\text{Reversible XOR / CNOT}}$$

Reversible programs permute the state space

A classical reversible program with a fixed number of bits acts as a *permutation* on the bit state space:

$$\underbrace{\begin{array}{c|cccccccc|}
 & 000 & 001 & 010 & 011 & 100 & 101 & 110 & 111 \\
000 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
001 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
010 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
011 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
100 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
101 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
110 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
111 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0
\end{array}}_{\text{Reversible AND / Toffoli}}$$

Reversible programs permute the state space

A classical reversible program with a fixed number of bits acts as a *permutation* on the bit state space:

$$\underbrace{\begin{array}{c|cccccccc} & 000 & 001 & 010 & 011 & 100 & 101 & 110 & 111 \\ \hline 000 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 001 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 010 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 011 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 100 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 101 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 110 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 111 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{array}}_{\text{Reversible AND / Toffoli}}$$

A special corollary: all reversible programs act *linearly* on the state space, and we could compose their semantics simply by multiplying their matrices.

(Why would this be a bad idea in practice?)

Quantum programs unitarily transform the state space

Any quantum program must act as a *unitary* transformation on the qubit state space.

Why? The squared amplitudes in a quantum state are the probabilities of measuring each basis state, so given a qubit state before and after acting with a program P:

$$|\Psi\rangle = \sum_{i=0}^{2^n-1} a_i |i\rangle \quad \text{and} \quad P|\Psi\rangle = \sum_{i=0}^{2^n-1} b_i |i\rangle$$

we must have $1 = a_0^2 + \ldots + a_{2^n-1}^2 = b_0^2 + \ldots + b_{2^n-1}^2$.

Why? Because the probability of measuring *something* must always be 1.

What can quantum programs compute?

Permutation matrices are one kind of unitary matrix, so every reversible program is a valid quantum program. But instead instead of transforming between bit-strings, we transform *linear combinations* of bit-strings.

$$\begin{array}{c|cccc} & 00 & 01 & 10 & 11 \\ \hline 00 & 1 & 0 & 0 & 0 \\ 01 & 0 & 1 & 0 & 0 \\ 10 & 0 & 0 & 0 & 1 \\ 11 & 0 & 0 & 1 & 0 \end{array} \qquad |10\rangle = |11\rangle$$

What can quantum programs compute?

Permutation matrices are one kind of unitary matrix, so every reversible program is a valid quantum program. But instead instead of transforming between bit-strings, we transform *linear combinations* of bit-strings.

$$\begin{array}{c|cccc} & 00 & 01 & 10 & 11 \\ \hline 00 & 1 & 0 & 0 & 0 \\ 01 & 0 & 1 & 0 & 0 \\ 10 & 0 & 0 & 0 & 1 \\ 11 & 0 & 0 & 1 & 0 \end{array} \quad (a_{00}|00\rangle + a_{01}|01\rangle + a_{10}|10\rangle + a_{11}|11\rangle) = a_{00}|00\rangle + a_{01}|01\rangle + a_{11}|10\rangle + a_{10}|11\rangle$$

What can quantum programs compute?

Permutation matrices are one kind of unitary matrix, so every reversible program is a valid quantum program. But instead instead of transforming between bit-strings, we transform *linear combinations* of bit-strings.

$$\begin{array}{c|cccc} & 00 & 01 & 10 & 11 \\ \hline 00 & 1 & 0 & 0 & 0 \\ 01 & 0 & 1 & 0 & 0 \\ 10 & 0 & 0 & 0 & 1 \\ 11 & 0 & 0 & 1 & 0 \end{array} \quad (a_{00}\,|00\rangle + a_{01}\,|01\rangle + a_{10}\,|10\rangle + a_{11}\,|11\rangle) = a_{00}\,|00\rangle + a_{01}\,|01\rangle + a_{11}|10\rangle + a_{10}|11\rangle$$

All unitary matrices can be generated by the **permutations** and **Bloch-sphere rotations**. Thus, quantum programs on *n* are classical reversible programs on *n* bits, extended with single-qubit rotations.

(In fact: CNOT + rotations suffice.)

Table of Contents

The Input Problem

Consider a quantum program on n qubits.
The qubits can be set to a single n-bit string in constant time (reset to zero + selective negation).

However, this only gives access to classical states.

To prepare a general quantum state with 2^n amplitudes, we need to actually perform the unitary operator

$$|00\cdots 0\rangle \mapsto \sum_{i=0}^{2^n-1} a_i\,|i\rangle$$

This requires $\mathcal{O}\left(4^n\right)$ rotations and CNOTs

The Input Problem

Consider a quantum program on n qubits.
The qubits can be set to a single n-bit string in constant time (reset to zero + selective negation).

However, this only gives access to classical states.

To prepare a general quantum state with 2^n amplitudes, we need to actually perform the unitary operator

$$|00\cdots 0\rangle \mapsto \sum_{i=0}^{2^n-1} a_i \, |i\rangle$$

This requires $\mathcal{O}\left(4^n\right)$ rotations and CNOTs $\Longrightarrow$ ☠!

The Input Problem

Consider a quantum program on n qubits.
The qubits can be set to a single n-bit string in constant time (reset to zero + selective negation).

However, this only gives access to classical states.

To prepare a general quantum state with 2^n amplitudes, we need to actually perform the unitary operator

$$|00\cdots 0\rangle \mapsto \sum_{i=0}^{2^n-1} a_i \,|i\rangle$$

This requires $\mathcal{O}\left(4^n\right)$ rotations and CNOTs $\Longrightarrow$ ☠!

Punchline:

Complicated input kills quantum advantage.

The Input Problem

Example: *"Quantum Machine Learning will be great for huge data sets! Just use QSVM and HHL on exponentially large matrices in polynomial time!"*

Nice idea, but how do we input that data to the quantum computer?
Answer: in general we can't, without losing quantum advantage.

Conclusion:

Quantum advantage requires <u>small</u> input.

The Input Problem

Example: *"Quantum Machine Learning will be great for huge data sets! Just use QSVM and HHL on exponentially large matrices in polynomial time!"*

Nice idea, but how do we input that data to the quantum computer?
Answer: in general we can't, without losing quantum advantage.

Conclusion:

Quantum advantage requires small input.

(Or structured, which means compressible $\rightsquigarrow$ small.)

The Output Problem

We can never observe the actual state

$$|\Psi\rangle = \sum_{i=0}^{2^n-1} a_i \, |i\rangle$$

We can only *measure* the qubits, which collapse the information-rich state (dim $= 2^n$ complex numbers) down to a single classical bit-string (dim $= n$ bits), and by doing so, destroy the quantum state.

(More accurately: collapse it to the projection where the measured qubits are fixed with one classical value).

I.e., measuring $|\Psi\rangle = a_{00}\,|00\rangle + a_{01}\,|01\rangle + a_{10}\,|10\rangle + a_{11}\,|11\rangle$ in the Z basis gives: $|00\rangle$ with probability $|a_{00}|^2, \ldots, |11\rangle$ with probability $|a_{11}|^2$.

The Output Problem

If our algorithm prepares a state $|\Psi\rangle = \sum_{i=0}^{2^n-1} a_i\,|i\rangle$ for which the desired solution $|i^*\rangle$ has high probability $|a_{i^*}|^2$, we only have to run the algorithm a few times (esp. if we can verify the correctness classically).

But: If our algorithm prepares a state for which we're interested in the *actual coefficients* a_i, we need to re-run the computation sufficiently many times to statistically determine the output a_i to desired numerical precision.

Punchline:

It's OK if we're interested in a few large a_i, but if we want all or most of them:

The Output Problem

If our algorithm prepares a state $|\Psi\rangle = \sum_{i=0}^{2^n-1} a_i |i\rangle$ for which the desired solution $|i^*\rangle$ has high probability $|a_{i^*}|^2$, we only have to run the algorithm a few times (esp. if we can verify the correctness classically).

But: If our algorithm prepares a state for which we're interested in the *actual coefficients* a_i, we need to re-run the computation sufficiently many times to statistically determine the output a_i to desired numerical precision.

Punchline:

It's OK if we're interested in a few large a_i, but if we want all or most of them:
$\mathcal{O}(2^n)$ repetitions with a large constant factor

The Output Problem

If our algorithm prepares a state $|\Psi\rangle = \sum_{i=0}^{2^n-1} a_i |i\rangle$ for which the desired solution $|i^*\rangle$ has high probability $|a_{i^*}|^2$, we only have to run the algorithm a few times (esp. if we can verify the correctness classically).

But: If our algorithm prepares a state for which we're interested in the *actual coefficients* a_i, we need to re-run the computation sufficiently many times to statistically determine the output a_i to desired numerical precision.

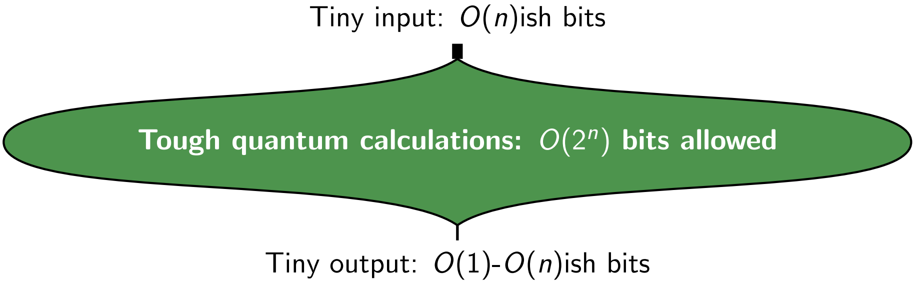
Punchline:

It's OK if we're interested in a few large a_i, but if we want all or most of them:
$\mathcal{O}(2^n)$ repetitions with a large constant factor $\Longrightarrow$ ☠!

Conclusion

Quantum computers (even in the future) can only be used to accellerate problems with small classical input and small classical output.

The problems are allowed exponentially large intermediate state (2^n complex numbers encoded in the wave-function amplitudes), but we can never look at it directly.


Tiny input: $O(n)$ish bits
Tough quantum calculations: $O(2^n)$ bits allowed
Tiny output: $O(1)$-$O(n)$ish bits

Anything dealing with large data is better left to massively parallel classical devices (multi-GPU, CPU, TPU, FPGA, etc.).

Asymptotics matter, but so do constants

Quantum computers are improving rapidly, but so are classical computers.

Frontier: 10^{19} bits with error rate $\sim 10^{-15}$/op. 10^{18} float64 ops/sec. on 6×10^5 CPU cores and 8×10^6 GPU cores.

IBM Condor: 1.121 physical qubits with error rate $\sim 10^{-2}$/op. (0 logical qubits.)
Hope: $\sim 10^3$ logical qubits with error rate $\sim 10^{-12}$ in 10-20 years.

But: Error correction also slows down each operation by a factor of $\sim 10^3$.

What will quantum computers do faster than classical ones?

5-6ish order of magnitude difference per operation $\rightsquigarrow$ we mostly care about exponential speedups.

When can exponential speedups occur?

Gottesman-Knill Theorem

Theorem: A quantum circuit using only the following elements can be simulated efficiently on a classical computer:

1. Preparation of qubits in computational basis states,
2. Clifford gates (Hadamard, CNOT, and phase gate S), and
3. Measurements in the computational basis.

What will quantum computers do faster than classical ones?

Extension of Gottesman-Knill by Gottesman and Aaronson

A general quantum circuit comprised of M Clifford gates and T non-Clifford gates can be simulated in time $\mathcal{O}(M \log M + \exp(T))$.

What will quantum computers do faster than classical ones?

Extension of Gottesman-Knill by Gottesman and Aaronson

A general quantum circuit comprised of M Clifford gates and T non-Clifford gates can be simulated in time $\mathcal{O}\left(M \log M + \exp(T)\right)$.

$\rightsquigarrow$ Unless we need at least $\mathcal{O}\left(n\right)$ non-Clifford gates, we can just simulate the quantum circuit on a classical computer, no need for a quantum computer.

What will quantum computers do faster than classical ones?

Extension of Gottesman-Knill by Gottesman and Aaronson

A general quantum circuit comprised of M Clifford gates and T non-Clifford gates can be simulated in time $\mathcal{O}\left(M \log M + \exp(T)\right)$.

$\rightsquigarrow$ Unless we need at least $\mathcal{O}\left(n\right)$ non-Clifford gates, we can just simulate the quantum circuit on a classical computer, no need for a quantum computer.

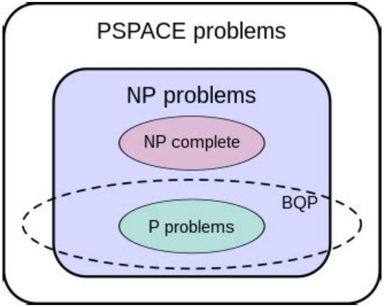
Conclusion

Necessary (but not sufficient) conditions for quantum advantage:

1. Problem needs $\simeq \mathcal{O}\left(n\right)$ or less input, and $\simeq \mathcal{O}\left(n\right)$ or less output. **(We can solve it fast on a quantum computer)**.
2. All known quantum algorithms for the problem require at least $\mathcal{O}\left(n\right)$ non-Clifford gates. **(We can't solve it fast by classical simulation)**.

What will quantum computers do faster than classical ones?

We expect that QC will not be able to solve NP-hard problems in polynomial time. $BQP \overset{?}{=} NP$ is just as open as $P \overset{?}{=} NP$, but we expect the hierarchy to look like this:



To have hope for efficient quantum algorithms for a given problem, its complexity has to fall in the sweet spot.

The need for Hybrid Quantum-Classical Programming

Obviously not all problems have the shape required for quantum advantage. Indeed, most problems don't.

However, real world problems often have *sub-problems* that would benefit from quantum computing, embedded in a larger context that is more efficiently solved classically.

The need for Hybrid Quantum-Classical Programming

Obviously not all problems have the shape required for quantum advantage. Indeed, most problems don't.

However, real world problems often have *sub-problems* that would benefit from quantum computing, embedded in a larger context that is more efficiently solved classically.

For example, in Shor's algorithm, the only quantum part is the phase-estimation in the period-finding subroutine.

The need for Hybrid Quantum-Classical Programming

Obviously not all problems have the shape required for quantum advantage. Indeed, most problems don't.

However, real world problems often have *sub-problems* that would benefit from quantum computing, embedded in a larger context that is more efficiently solved classically.

For example, in Shor's algorithm, the only quantum part is the phase-estimation in the period-finding subroutine.

In physics, chemistry, and biology problems, we would often have a large environment that can be efficiently simulated on a classical computer (consuming and producing large amounts of data), but with small hard quantum sub-problems modeling e.g. chemical reactions, catalysis, contact surface interactions, etc.

Table of Contents

The need for Hybrid Quantum-Classical Programming

To leverage quantum computing in practice, we need to be able to:

1. Identify the sub-problems that can benefit from quantum computing.
2. Decompose the problem into quantum and classical parts and identify the interfaces and interaction.
3. Efficiently encode the input and output of the quantum sub-problems.
4. Efficiently combine the quantum and classical parts.

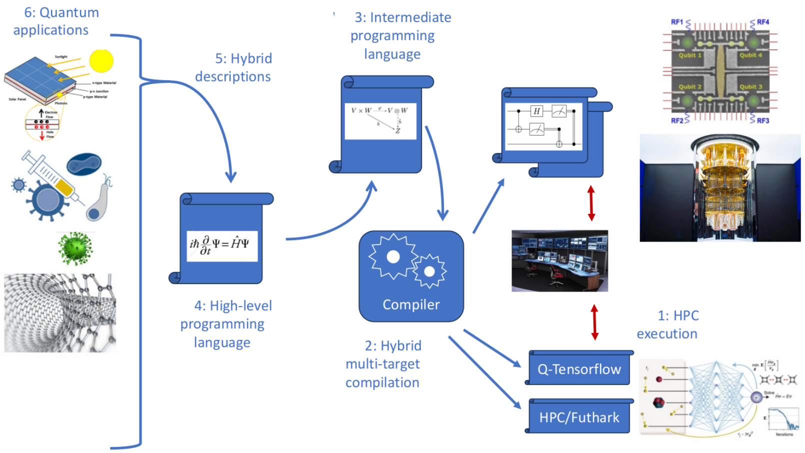
The need for Hybrid Programming Models, Languages and Compilers

Thus, we will need:

- Hybrid **programming languages** that allow us to express the QC and CC parts and their interactions.
- Hybrid **programming models** and formal semantics that allow us to reason about hybrid programs.
- Hybrid **compilers** that can optimize the quantum and classical parts of the computation, and ultimately produce code that runs on a combination of classical HPC and quantum accellerator hardware, and orchestrates their interactions.

This does *not* currently exist. You will take part in building this brave new world!

Known unknowns: A few open challenges



Let’s get to work!

Quantum computing is in its infancy - think Charles Babbage’s 19th century Analytical Engine.

We don’t yet know the technology future quantum devices will use. But we already know enough about their strengths and limitations to know that we’ll only use them to accellerate specialized tasks, and effective use will require hybrid quantum-classical programming models.

It’s not too early to start designing and building these! It can easily take a decade to get this right.

So: Let’s get to it!