Verified Optimization in a Quantum Intermediate Representation

Kesha Hietala, Robert Rand, Shih-Han Hung, Xiaodi Wu and Michael Hicks

University of Maryland, College Park
Verified compiler stack

- End goal: *verified compiler stack* for quantum programs

1. **High-level Language**
   - E.g. QWIRE, Quipper, Q#
   - Formally verified that the transformation is semantics-preserving.

2. **General Purpose IR**
   - E.g. OpenQASM, Quil

3. **Machine Specific IR**
   - E.g. OpenQASM, Quil... other?

4. **Hardware Instructions**

Optimization
Circuit synthesis
Circuit mapping
...
Verified compiler stack

- End goal: *verified compiler stack* for quantum programs
Outline

- SQIRE
- Verified compilation
- General verification
- Ongoing work
SQIRE

• A Small Quantum Intermediate REpresentation

• Design goals:
  ▶ As simple as possible
  ▶ Expressive enough to describe interesting algorithms
  ▶ Similar to existing quantum IRs (e.g. OpenQASM, Quil)
  ▶ Easy to use in proofs
Unitary programs

• Syntax

\[ P \rightarrow \text{skip} \]
\[ | P_1; P_2 \]
\[ | U \ q_1 \ldots q_n \]
\[ U \rightarrow H \mid X \mid Y \mid Z \mid R_\phi \mid \text{CNOT} \]

- Qubits are referred to by indices into a *global register*

• Semantics

\[ [\text{skip}]^{\text{dim}} = I_{2^{\text{dim}}} \]
\[ [P_1; P_2]^{\text{dim}} = [P_2]^{\text{dim}} \times [P_1]^{\text{dim}} \]
\[ [U \ q_1 \ldots q_n]^{\text{dim}} = \begin{cases} \text{ueval}(U, q_1\ldots q_n) & \text{well-typed} \\ 0_{2^{\text{dim}}} & \text{otherwise} \end{cases} \]
Example

- Superdense coding protocol in SQIRE

```
Definition bell00 := H 0; CNOT 0 1.

Definition encode (b1 b2 : B) :=
  (if b2 then X 0 else skip);
  (if b1 then Z 0 else skip).

Definition decode := CNOT 0 1; H 0.

Definition superdense (b1 b2 : B) :=
  bell00; encode b1 b2; decode.
```

- Correctness: Lemma superdense_correct : \( \forall (b1 b2 : B), \langle superdense b1 b2 \rangle \otimes |0,0\rangle = |b1, b2\rangle. \)
Full SQIRE language

- Includes initialization and measurement

- Two different semantics:
  - Density matrix semantics (used by QWIRE, QHL)
Includes initialization and measurement

Two different semantics:

\[ \text{Density matrix semantics (used by QWIRE, QHL)} \]

\[ \text{Full SQIRE language} \]

- \([\text{skip}]^{\text{dim}}(\rho) = \rho\]
- \([P_1; P_2]^{\text{dim}}(\rho) = (\left[ P_2 \right]^{\text{dim}} \circ \left[ P_1 \right]^{\text{dim}})(\rho)\]
- \([U \ q_1 \ldots q_n]^{\text{dim}}(\rho) = \begin{cases} \text{ueval}(U) \times \rho \times \text{ueval}(U)^\dagger & \text{well-typed} \\ 0_{2^{\text{dim}}} & \text{otherwise} \end{cases}\]
- \([\text{meas} \ q]^{\text{dim}}(\rho) = |0\rangle_q\langle 0|\rho|0\rangle_q\langle 0| + |1\rangle_q\langle 1|\rho|1\rangle_q\langle 1|\]
- \([\text{reset} \ q]^{\text{dim}}(\rho) = |0\rangle_q\langle 0|\rho|0\rangle_q\langle 0| + |0\rangle_q\langle 1|\rho|1\rangle_q\langle 0|\]

Full SQIRE language
Full SQIRE language

• Includes initialization and measurement

• Two different semantics:
  ▶ Density matrix semantics (used by QWIRE, QHL)
  ▶ Non-deterministic semantics (allows states to be represented as vectors)
Full SQIRE language

- Includes initialization and measurement

```latex
Inductive nd_eval \{dim : \mathbb{N}\} : com \to Vector (2^{\text{dim}}) \to Vector (2^{\text{dim}}) \to \mathbb{P} :=
| nd_app : \forall n (u : \text{Unitary} n) (l : \text{list} \mathbb{N}) (\psi : Vector (2^{\text{dim}})),
  \quad \text{app} \ u \ l \ / \ \psi \downarrow ((\text{ueval} \ \text{dim} \ u \ l) \times \psi)
| nd_meas0 : \forall n (\psi : Vector (2^{\text{dim}})),
  \quad \text{let} \ \psi' := \text{pad} \ n \ \text{dim} \ |0\rangle\langle 0| \times \psi \ \text{in}
  \quad \text{norm} \ \psi' \neq 0 \to
  \quad \text{meas} \ n / \ \psi \downarrow \psi'
...

where "c":"psi":"\downarrow"" psi" := (nd_eval c \ psi \ psi').
```
Full SQIRE language

- Includes initialization and measurement

- Two different semantics:
  - Density matrix semantics (used by QWIRE, QHL)
  - Non-deterministic semantics (allows states to be represented as vectors)
  - Example proof of teleport protocol using both semantics in our paper
SQIRE vs. QWIRE

• SQIRE is built on the Coq libraries developed for QWIRE

• Why a new language?
  ▶ Verification of QWIRE program transformations is complicated by the use of higher-order abstract syntax
  ▶ Referencing qubits by natural numbers indexing into a global register makes denotation function simpler
  ▶ Other simplifying assumptions (only multi-qubit gate is CNOT, no dynamic lifting)

• However, by using a global register we sacrifice compositionality
Composition in SQIRE

• Composition in SQIRE requires renaming qubits
Composition in SQIRE

- Composition in SQIRE requires renaming qubits
Composition in SQIRE

- Composition in SQIRE requires renaming qubits
Composition in SQIRE

- Composition in SQIRE requires renaming qubits

![Diagram showing qubit renaming in SQIRE](image-url)
Composition in SQIRE

- Composition in SQIRE requires renaming qubits

- This is tedious to do by hand, but we don’t expect to write SQIRE programs manually
Outline

• SQIRE

• Verified compilation

• General verification

• Ongoing work
Verified transformations

• In general, we want to verify that the input to a given transformation is semantically equivalent to the output.

• We say that program $c_1$ is equivalent to $(\equiv) c_2$ if for all $dim$, $[c_1]^{dim} = [c_2]^{dim}$.

▶ Some useful equivalences:

- **Lemma** useq_assoc : $\forall$ c1 c2 c3, $((c_1 \; c_2) \; c_3) \equiv (c_1 \; (c_2 \; c_3))$.

- **Lemma** useq_congruence : $\forall$ c1 c1' c2 c2', $c_1 \equiv c_1' \rightarrow c_2 \equiv c_2' \rightarrow c_1 \; c_2 \equiv c_1' \; c_2'$.

- **Lemma** uskip_id_l : $\forall$ c, $\text{uskip} \; c \equiv c$.

- **Lemma** X_CNOT_comm : $\forall$ c t, $(X \; t; \text{CNOT} \; c \; t) \equiv (\text{CNOT} \; c \; t; X \; t)$.
Skip removal

• Simple optimization that removes “skip” constructs

```plaintext
Fixpoint rm_skips c :=
  match c with
  | c1 ; c2 ⇒ match rm_skips c1, rm_skips c2 with
    | skip, c2' ⇒ c2'
    | c1', skip ⇒ c1'
    | c1', c2' ⇒ c1'; c2'
  end
  | _  ⇒ c
end.
```
Skip removal

- Simple optimization that removes “skip” constructs

  - \textit{rm\_skips} preserves semantics

    \textit{Lemma} \textit{rm\_skips\_sound} : \forall \; c, \; c \equiv (\textit{rm\_skips} \; c).

  - \textit{rm\_skips} removes skips

\textit{Inductive} \textit{skip\_free} : \textit{ucom} \rightarrow \mathbb{P} :=
\begin{align*}
| \text{SF\_seq} & : \forall \; c1 \; c2, \; \text{skip\_free} \; c1 \rightarrow \text{skip\_free} \; c2 \rightarrow \text{skip\_free} \; (c1; \; c2) \\
| \text{SF\_app} & : \forall \; n \; l \; (u : \text{Unitary} \; n), \; \text{skip\_free} \; (\text{uapp} \; u \; l).
\end{align*}

\textit{Lemma} \textit{rm\_skips\_correct} : \forall \; c,
\begin{align*}
(\text{\textit{rm\_skips} \; c}) & = \text{skip} \lor \text{skip\_free} \; (\text{\textit{rm\_skips} \; c}).
\end{align*}
X propagation

• Pre-processing step from a recent circuit optimizer\textsuperscript{1}
  ‣ Idea is to cancel redundant X gates

\begin{center}
\begin{tikzpicture}
\node[draw,shape=rectangle,minimum size=1 cm] (a) at (0,0) {$X$};
\node[draw,shape=rectangle,minimum size=1 cm] (b) at (1,0) {$X$};
\node[circle,fill,inner sep=1.5pt] (c) at (2,0) {};
\node[draw,shape=rectangle,minimum size=1 cm] (d) at (3,0) {$X$};
\node[draw,shape=rectangle,minimum size=1 cm] (e) at (4,0) {$X$};
\node[draw,shape=rectangle,minimum size=1 cm] (f) at (5,0) {$X$};
\node[circle,fill,inner sep=1.5pt] (g) at (6,0) {};
\node[draw,shape=rectangle,minimum size=1 cm] (h) at (7,0) {$X$};
\node[draw,shape=rectangle,minimum size=1 cm] (i) at (8,0) {$X$};
\draw (a) -- (b);
\draw (b) -- (c);
\draw (c) -- (d);
\draw (d) -- (e);
\draw (e) -- (f);
\draw (f) -- (g);
\draw (g) -- (h);
\draw (h) -- (i);
\end{tikzpicture}
\end{center}

• We have proved that this transformation is sound

\textsuperscript{1} Nam et al. “Automated optimization of large quantum circuits with continuous parameters” npj Quantum Information 4. 2018.
LNN mapping

• Convert a SQIRE program into a program that can be run on a linear nearest neighbor (LNN) architecture
  ▶ Consider a naïve mapping strategy that adjusts qubit placement using swaps before & after each CNOT
  ▶ E.g. \( \text{CNOT 1 3} \rightarrow \text{SWAP 1 2; CNOT 2 3; SWAP 1 2} \)

• The transformation is sound, and the output program satisfies the LNN constraint
Outline

• SQIRE

• Verified compilation

• General verification

• Ongoing work
GHZ state preparation

- $n$-qubit GHZ state preparation in SQIRE

Definition \( \text{ghz} \ (n : \mathbb{N}) : \text{Vector} \ (2 \ ^{\wedge} \ n) := \)
\[
\text{match } n \ \text{with} \\
| 0 \Rightarrow \text{I} \ 1 \\
| S \ n' \Rightarrow 1/\sqrt{2} \cdot (\text{nket} \ n \ |0\rangle) \ + \ 1/\sqrt{2} \cdot (\text{nket} \ n \ |1\rangle)
\]
end.

\[|\text{GHZ}\rangle = \frac{1}{\sqrt{2}}(|0\rangle^{\otimes n} + |1\rangle^{\otimes n}).\]

Fixpoint \( \text{GHZ} \ (n : \mathbb{N}) : \text{ucom} := \)
\[
\text{match } n \ \text{with} \\
| 0 \Rightarrow \text{uskip} \\
| 1 \Rightarrow \text{H} \ 0 \\
| S \ n' \Rightarrow \text{GHZ} \ n'; \ \text{CNOT} \ (n' - 1) \ n'
\]
end.

- Correctness

Theorem \( \text{ghz\_correct} : \forall \ n : \mathbb{N}, \ [\text{GHZ} \ n]^{n} \times \text{nket} \ n \ |0\rangle = \text{ghz} \ n. \)
**Boolean oracle compilation**

\[(b_0 \land b_1) \oplus b_2\]

- Verify the compilation of boolean formulas into quantum circuits with X, CNOT, and Toffoli gates
  - Previously done by Amy et al.\(^2\) and Rand et al.\(^3\)
  - The output should be logically correct, and all ancilla should be returned to the zero state

\(^2\) Amy et al. "Verified compilation of space-efficient reversible circuits" CAV 2017.
\(^3\) Rand et al. "ReQWIRE: Reasoning about reversible quantum circuits" QPL 2018.
Outline

- SQIRE
- Verified compilation
- General verification
- Ongoing work
Ongoing work

• Full-featured optimizer for quantum circuits, in the style of Nam et al., Qiskit, etc.

• Support for mapping to arbitrary architectures

• SQIRE for teaching: see Verified Quantum Computing
  ▶ This has led to several proof engineering questions:
    - What program representation should we use?
    - What mathematical representation (e.g. of tensor product) should we use?
    - What automation do we need?
Conclusions

• Initial progress on a verified compiler stack for quantum programs
  ▶ Presented an IR for quantum programs, embedded in Coq
  ▶ Verified simple optimization and mapping algorithms

High-level Language
E.g. QWIRE, Quipper, Q#

General Purpose IR
E.g. OpenQASM, Quil

Machine Specific IR
E.g. OpenQASM, Quil... other?

Hardware Instructions

General Purpose SQIRE

Machine Specific SQIRE

Paper on arXiv: 1904.06319
Code on github: inQWIRE/SQIRE