Deriving a Kronecker-Free Functional Quantum Simulator

Martin Elsman

Department of Computer Science,
University of Copenhagen (DIKU)

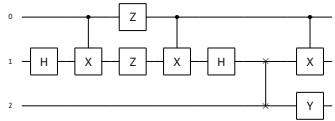
WQS 2025
Seoul, Republic of Korea

June 17, 2025

Paper pdf

Quantum Circuit Semantics and State-Vector Simulators

- The semantics of quantum circuits are specified **declaratively** and **compositionally** as unitary matrices.
- Efficient quantum simulators are **imperative** and operate using in-place array-updates on a global state vector with indices calculated using complex addressing.
- A formal connection is seldom established!



(Example circuit)

$$\begin{bmatrix} -i & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & i & 0 & 0 & 0 & 0 & 0 \\ 0 & -i & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & i & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -i & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & i \\ 0 & 0 & 0 & 0 & -i & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & i & 0 \end{bmatrix}$$

(Denotation)

Contributions:

- We derive, from the semantics of circuits, an interpreter that works directly on state vectors without first generating large unitary matrices.
- The derivation builds on the well-known concept of *vectorisation* (i.e., a connection between column stacking and the Kronecker product).

Qubits, Gates, and Multi-Qubit States

- A *qubit* can be modelled as a two-dimensional complex vector $\begin{bmatrix}\alpha\\ \beta\end{bmatrix}$ specifying a linear combination $\alpha|0\rangle + \beta|1\rangle$ of the basis vectors $|0\rangle$ and $|1\rangle$ such that $|\alpha|^2 + |\beta|^2 = 1$.
- *Single-qubit gates* model transformations of a qubit and can be represented as 2×2 *unitary* complex matrices U, meaning $U^\dagger U = UU^\dagger = I$ (norm-preserving and reversible):

$$X = \begin{bmatrix}0 & 1\\ 1 & 0\end{bmatrix} \; Y = \begin{bmatrix}0 & -i\\ i & 0\end{bmatrix} \; Z = \begin{bmatrix}1 & 0\\ 0 & -1\end{bmatrix} \; H = \frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix} \; T = \begin{bmatrix}1 & 0\\ 0 & e^{i\pi/4}\end{bmatrix} \; I = \begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix}$$

- A state of more qubits is modelled as a product of all standard bases.
  - E.g., a two-qubit state is $\alpha|00\rangle + \beta|01\rangle + \gamma|10\rangle + \theta|11\rangle$, where $|\alpha|^2 + |\beta|^2 + |\gamma|^2 + |\theta|^2 = 1$. I.e., a four-element complex vector $[\alpha\ \beta\ \gamma\ \theta]^T$.
  - The state space doubles with every added qubit!

Multi-Qubit Circuits (Composition of Gates)

Multi-qubit circuits are modelled using **matrix multiplication** (horizontal composition) and **tensor products** (vertical composition).

$$A \otimes B = \begin{bmatrix} a_{11}B & \cdots & a_{1n}B \\ \vdots & \ddots & \vdots \\ a_{m1}B & \cdots & a_{mn}B \end{bmatrix} \qquad A \otimes \mathbf{1} = A = \mathbf{1} \otimes A$$

Special matrices are used for specifying **qubit swapping** (SW) and for specifying so-called **control gates** ($C_n\ U$).

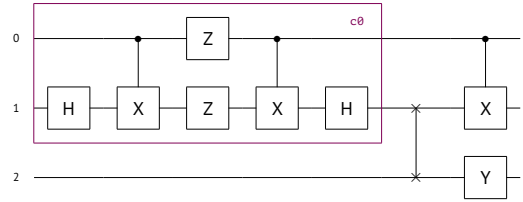
$$SW = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad I_n = \begin{cases} \mathbf{1} & \text{if } n = 0 \\ I \otimes I_{n-1} & \text{otherwise} \end{cases} \qquad \begin{aligned} C_0\ U &= U \\ C_n\ U &= \begin{bmatrix} I_n & 0 \\ 0 & C_{n-1}\ U \end{bmatrix} \end{aligned}$$

A Language for Specifying Circuits

We can define a small DSL for specifying circuits:

$$
\begin{array}{rcl}
c & ::= & \mathtt{X} \mid \mathtt{Y} \mid \mathtt{Z} \mid \mathtt{H} \mid \mathtt{T} \mid \mathtt{I} \mid \mathtt{SW} \\
& \mid & c + c \mid c \otimes c \mid \mathtt{C}\, c
\end{array}
$$

Circuits may be composed sequentially (using $+$) and vertically (using $\otimes$).



(Example circuit)

```
val c0 = I⊗H + C X + Z⊗Z + C X + I⊗H
val c1 = c0⊗I + I⊗SW + C X⊗Y
```

(Circuit code)

Semantics (denotation)

We can define a partial function $h : \mathit{Circ} \rightharpoonup \mathbb{N}$ for computing the height of a circuit c:

$$h(c_1 \otimes c_2) = h\ c_1 + h\ c_2 \qquad h(c_1 + c_2) = h\ c_1 \quad (h\ c_1 = h\ c_2)$$
$$h(\mathtt{C}\ c) = 1 + h\ c \qquad h\ c = 1 \quad (c \in \{\mathtt{X}, \mathtt{Y}, \mathtt{Z}, \mathtt{H}, \mathtt{T}, \mathtt{I}\})$$

The semantics of a circuit c with height $k = h\ c$ is specified by the unitary matrix $[\![c]\!] : \mathbb{C}^{2^k \times 2^k}$ defined as follows:

$$[\![c_1 + c_2]\!] = [\![c_2]\!][\![c_1]\!] \qquad [\![\mathtt{X}]\!] = X \qquad [\![\mathtt{H}]\!] = H$$
$$[\![c_1 \otimes c_2]\!] = [\![c_2]\!] \otimes [\![c_1]\!] \qquad [\![\mathtt{Y}]\!] = Y \qquad [\![\mathtt{T}]\!] = T$$
$$[\![\mathtt{C}\ c]\!] = C_{(h\ c)}\ ([\![c]\!]) \qquad [\![\mathtt{Z}]\!] = Z \qquad [\![\mathtt{I}]\!] = I$$

For evaluating a circuit c on a state vector $v : \mathbb{C}^{2^{(h\ c)}}$, we can compute $[\![c]\!]\ v$.

Can we do better?

Kronecker-Free Interpretation (I)

We can *derive* an efficient interpreter $\mathcal{I}[c]\ v$ inductively:

For any circuit c such that $h\ c = k$ and state vector $v : \mathbb{C}^{2^k}$, we have $\mathcal{I}[c]\ v = [\![c]\!]\ v$

The derivation is based on *vectorisation* and properties of matrix multiplication:

$$
\begin{aligned}
(A \otimes B)\ v &= \mathtt{vec}(B(\mathtt{unvec}\ v)A^T) && \textit{(vectorisation)} \\
\mathtt{vec}\ A &= \mathtt{flatten}\ (A^T) && \textit{(column stacking)} \\
\mathtt{unvec}\ v &= (\mathtt{unflatten}\ v)^T && \textit{(cut vector into columns)} \\
\\
AB &= (\mathtt{map}\ (\lambda v.A\ v)\ B^T)^T && \textit{(matmul functionalise)} \\
(AB)^T &= B^T A^T && \textit{(matmul transpose)}
\end{aligned}
$$

Kronecker-Free Interpretation (II)

We proceed by case analysis.

For base cases, the denotation gives us the desired implementations immediately:

$$\mathcal{I}[c]\ v = v \qquad c = \mathtt{I}$$
$$\mathcal{I}[c]\ v = [\![c]\!]\ v \qquad c \in \{\mathtt{X}, \mathtt{Y}, \mathtt{Z}, \mathtt{H}, \mathtt{T}, \mathtt{SW}\}$$

For sequential composition, we have

$$\begin{aligned}
\mathcal{I}[c_1 + c_2]\ v &= [\![c_1 + c_2]\!]\ v && \textit{from property} \\
&= ([\![c_2]\!][\![c_1]\!])\ v && \textit{from denotation} \\
&= [\![c_2]\!]([\![c_1]\!]\ v) && \textit{matrix-vector multiply} \\
&= [\![c_2]\!](\mathcal{I}[c_1]\ v) && \textit{by induction} \\
&= \mathcal{I}[c_2](\mathcal{I}[c_1]\ v) && \textit{by induction}
\end{aligned}$$

Kronecker-Free Interpretation (III)

For the tensor-product case, we have

$$
\begin{aligned}
\mathcal{I}[c_1 \otimes c_2]\ v &= [\![c_1 \otimes c_2]\!]\ v && \textit{from property} \\
&= ([\![c_1]\!] \otimes [\![c_2]\!])\ v && \textit{from denotation} \\
&= \mathtt{vec}([\![c_2]\!](\mathtt{unvec}\ v)([\![c_1]\!]^T)) && \textit{vectorisation} \\
&= \mathtt{vec}(([\![c_1]\!]([\![c_2]\!](\mathtt{unvec}\ v))^T)^T) && \textit{matmul transpose} \\
&= \mathtt{let}\ W = \mathtt{map}\ [\![c_2]\!]\ (\mathtt{unvec}\ v)^T && \textit{matmul functionalise twice} \\
&\quad\ \mathtt{let}\ Y = \mathtt{map}\ [\![c_1]\!]\ W^T \\
&\quad\ \mathtt{in}\ \mathtt{vec}\ Y \\
&= \mathtt{let}\ W = \mathtt{map}\ (\mathcal{I}[c_2])\ (\mathtt{unvec}\ v)^T && \textit{by induction twice} \\
&\quad\ \mathtt{let}\ Y = \mathtt{map}\ (\mathcal{I}[c_1])\ W^T \\
&\quad\ \mathtt{in}\ \mathtt{vec}\ Y
\end{aligned}
$$

Full Kronecker-Free Interpreter in a Functional Language

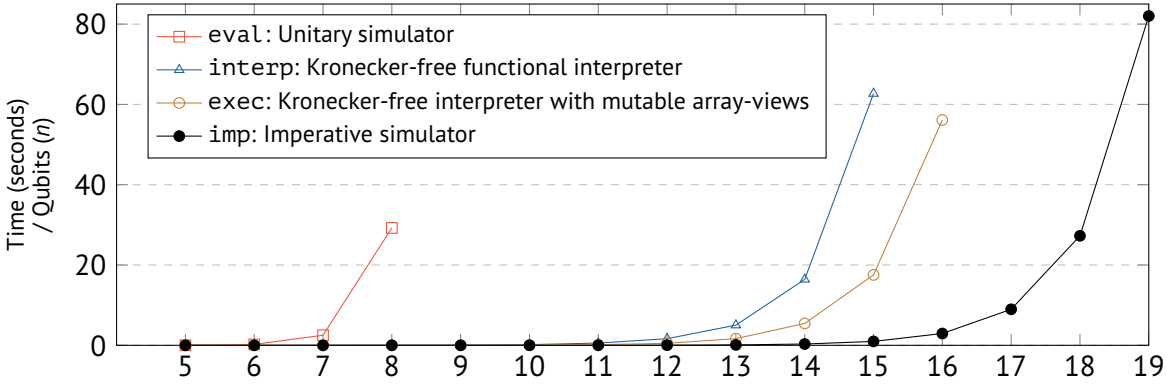
```
fun eval (c:c, v:vec) : vec =
  M.matvecmul_gen C.* C.+ (C.fromInt 0) (sem c) v

fun interp (c:c) (v:vec) : vec =
  case c of I ⇒ v
          | A + B ⇒ interp B (interp A v)
          | A ⊗ B ⇒ let val V = unvec (pow2(height A),pow2(height B)) v
                        val W = mapRows (interp B) (M.transpose V)
                        val Y = mapRows (interp A) (M.transpose W)
                    in vec Y
                    end
          | C A ⇒ Vector.concat [vecTake (pow2(height A)) v,
                                 interp A (vecDrop (pow2(height A)) v)]
          | _ ⇒ eval (c,v)
```

Recall: A controlled circuit *C A* acts as the identity on the first half of a state vector and as *A* on the second half.

Benchmarks

We have implemented Grover's search algorithm and measured the running time for different variations of simulators and different sizes of search problems.[1]



[1] Compiled with MLton and executed on a 2023 MacBook Pro (M2 Max) with 32GB of memory.

Much Related Work

- Many other simulators (qsim, QuEST, ...)
- Deriving a data-parallel gate-library (ARRAY '25: Gate Fusion is Map Fusion).

Opportunities and Future Work

- Performance comparisons with state-of-the-art simulators.
- Monadic generation of circuit-specialised simulators (e.g., for generating GPU code).
- Specialised code for diagonal and antidiagonal circuits.
- Offline gate fusion.
- More basic gates, measurements, and support for swapping distant qubits.
- ...