Gate Fusion is Map Fusion

Martin Elsman[1]

Department of Computer Science,
University of Copenhagen (DIKU)

ARRAY 2025
Seoul, Republic of Korea

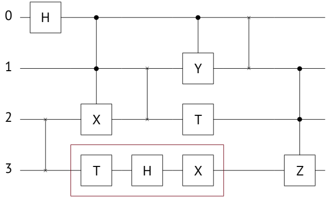
June 17, 2025

Paper pdf

[1]Joined work with Troels Henriksen, DIKU

State-Vector Quantum Simulators and Gate Fusion

- Efficient quantum circuit simulators are **imperative**.
- Gates operate on a **global state vector** (of size 2^k) with the precise access and update patterns depending on which qubits the gates operate on.
- Gate-fusion is **important for performance**.



(Example circuit, $k = 4$)

Contribution: Using the well-known concept of *vectorisation* (i.e., column stacking meets the Kronecker product), we demonstrate that *gate operations become index-transformed maps* and *gate fusion becomes map fusion*.

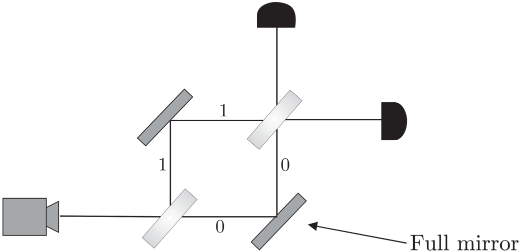
Efficient quantum simulators set the bar for quantum supremacy and are used for exploring quantum algorithms (qsim, QuEST, ...).

The Breakdown of Classical Reasoning: The Double Beam Splitter

One beam splitter puts a photon in "superposition", but two beam splitters in sequence act as the identity!

A beam splitter can be modelled by a so-called Hadamard gate (H):

$$
\begin{aligned}
HH &= \left(\frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\1 & -1\end{bmatrix}\right)\left(\frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\1 & -1\end{bmatrix}\right)\\
&= \begin{bmatrix}1 & 0\\0 & 1\end{bmatrix}\\
&= I
\end{aligned}
$$



Setup with two beam splitters[2]

[2] From *Phillip Kaye, Raymond Laflamme, and Michele Mosca. An Introduction to Quantum Computing. Oxford University Press. 2007*

Qubits and Single-Qubit Gates

- 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 $|\alpha|^2 + |\beta|^2 = 1$.
- A *single-qubit gate* models a transformation of a qubit and can be represented as a 2×2 *unitary* complex matrix U, meaning $U^\dagger U = UU^\dagger = I$ (norm-preserving and reversible).
- Pauli-gates:

$$X = \begin{bmatrix}0 & 1 \\ 1 & 0\end{bmatrix} \qquad Y = \begin{bmatrix}0 & -i \\ i & 0\end{bmatrix} \qquad Z = \begin{bmatrix}1 & 0 \\ 0 & -1\end{bmatrix}$$

- Hadamard-gate and T-gate:

$$H = \frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1 \\ 1 & -1\end{bmatrix} \qquad T = \begin{bmatrix}1 & 0 \\ 0 & e^{i\pi/4}\end{bmatrix}$$

Multi-Qubit States

A state of more qubits is modelled as a product of all standard bases.

- A two-qubit state is $\alpha|00\rangle + \beta|01\rangle + \gamma|10\rangle + \theta|11\rangle$, where $|\alpha|^2 + |\beta|^2 + |\gamma|^2 + |\theta|^2 = 1$.
- A three-qubit state is modelled by an eight-element complex vector.

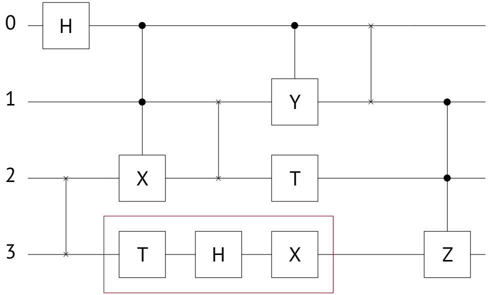
$$|000\rangle = \begin{bmatrix}1\\0\\0\\0\\0\\0\\0\\0\end{bmatrix} \qquad |001\rangle = \begin{bmatrix}0\\1\\0\\0\\0\\0\\0\\0\end{bmatrix} \qquad |010\rangle = \begin{bmatrix}0\\0\\1\\0\\0\\0\\0\\0\end{bmatrix} \qquad \cdots \qquad |111\rangle = \begin{bmatrix}0\\0\\0\\0\\0\\0\\0\\1\end{bmatrix}$$

Qubits, Gates, Statements

$$
\begin{array}{lll}
n & \in & \mathbb{N} \\
q & \in & \mathrm{Qubits} = \mathbb{N} \\
\\
g & ::= & \mathtt{X} \mid \mathtt{Y} \mid \mathtt{Z} \mid \mathtt{H} \mid \mathtt{T} \\
\\
s & ::= & \mathtt{gate}\; g\; q \mid \mathtt{swap}\; q \mid \mathtt{cntrl}\; n\; g\; q \\
 & \mid & s \,;\, s \mid \mathtt{nop}
\end{array}
$$

Circuits can be represented by statements (different statements may represent the same circuit).

Notice: Gates T, H, and X on qubit 3 are subject to *gate fusion!*



(Example circuit)

$$
\begin{array}{lll}
s_0 & = & \mathtt{gate\ H\ 0} \,;\, \mathtt{swap\ 2} \,; \\
 & & \mathtt{cntrl\ 2\ X\ 0} \,;\, \boxed{\mathtt{gate\ T\ 3}} \,; \\
 & & \mathtt{swap\ 1} \,;\, \boxed{\mathtt{gate\ H\ 3}} \,; \\
 & & \mathtt{cntrl\ 1\ Y\ 0} \,;\, \mathtt{gate\ T\ 2} \,;\, \boxed{\mathtt{gate\ X\ 3}} \,; \\
 & & \mathtt{swap\ 0} \,;\, \mathtt{cntrl\ 2\ Z\ 1}
\end{array}
$$

Multi-Qubit Circuits

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** ($C_n\ U$).

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

Semantics (denotation)

Single-qubit gates g are denoted, written $[\![g]\!]$, by their corresponding 2×2 unitary matrix:

$$[\![\texttt{I}]\!] = I \qquad [\![\texttt{X}]\!] = X \qquad [\![\texttt{Y}]\!] = Y \qquad [\![\texttt{Z}]\!] = Z \qquad \cdots$$

A k-qubit circuit (i.e., a statement) s is denoted by a unitary matrix $[\![s]\!]_k : \mathbb{C}^{2^k \times 2^k}$:

$$\begin{array}{lcl}
[\![\texttt{gate}\ g\ q]\!]_k & = & I_q \otimes [\![g]\!] \otimes I_{k-q-1} \\
[\![\texttt{swap}\ q]\!]_k & = & I_q \otimes SW \otimes I_{k-q-2} \\
[\![\texttt{cntrl}\ n\ g\ q]\!]_k & = & I_q \otimes C_n\ [\![g]\!] \otimes I_{k-q-n-1} \\
[\![s_1 ; s_2]\!]_k & = & [\![s_2]\!]_k [\![s_1]\!]_k \\
[\![\texttt{nop}]\!]_k & = & I_k
\end{array}$$

For evaluating a k-qubit circuit s on a state vector $v : \mathbb{C}^{2^k}$, we can compute $[\![s]\!]_k\ v$.

Can we do better?

Futhark: A Data-Parallel Array Language

Futhark is a data-parallel array language featuring standard operations on arrays:

$$
\begin{aligned}
\texttt{map} &: \forall n\alpha\beta.(\alpha \to \beta) \to [n]\alpha \to [n]\beta \\
\texttt{transpose} &: \forall mn\alpha.[m][n]\alpha \to [n][m]\alpha \\
\texttt{flatten} &: \forall mn\alpha.[m][n]\alpha \to [m*n]\alpha \\
\texttt{unflatten} &: \forall mn\alpha.[m*n]\alpha \to [m][n]\alpha \\
\texttt{vec} &: \forall mn\alpha.[m][n]\alpha \to [n*m]\alpha \\
&= \texttt{flatten} \circ \texttt{transpose} \\
\texttt{unvec} &: \forall mn\alpha.[m*n]\alpha \to [n][m]\alpha \\
&= \texttt{transpose} \circ \texttt{unflatten}
\end{aligned}
$$

`https://futhark-lang.org`

Futhark targets CUDA, OpenCL, C, and Multi-cores.

Kronecker-Free, Data-Parallel Interpretation

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

> *For any statement s, integer $k > 0$, and state vector $v : \mathbb{C}^{2^k}$, we have* $\mathcal{I}[s]_k\ v = [\![\, s \,]\!]_k\ v$

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

$$\boxed{(A \otimes B)\ v = \mathtt{vec}(B(\mathtt{unvec}\ v)A^T)} \quad (1) \qquad \boxed{AB = (\mathtt{map}\ A\ B^T)^T} \quad (2) \qquad \boxed{(AB)^T = B^T A^T} \quad (3)$$

Calculations yield:

$$\begin{aligned}
\mathcal{I}[\mathtt{gate}\ g\ q]_k\ v &= ((I_q \otimes g) \otimes I_{k-q-1})\ v \\
&\overset{1,3}{=} \mathtt{vec}(((I_q \otimes g)(\mathtt{unvec}\ v)^T)^T) \\
&\overset{2}{=} \mathtt{vec}(\mathtt{map}\ (I_q \otimes g)\ (\mathtt{unvec}\ v)) \\
&= \mathtt{let}\ F\ u = \mathtt{flatten}(\mathtt{map}\ [\![\, g \,]\!]\ (\mathtt{unflatten}\ u)) \\
&\quad\ \mathtt{in}\ \mathtt{vec}(\mathtt{map}\ F\ (\mathtt{unvec}\ v))
\end{aligned}$$

Similar calculations for swap and cntrl

Fusion

Futhark fuses succinct gate operations that target identical qubits:

$$\mathcal{I}[\texttt{gate}\ g\ q; \texttt{gate}\ g'\ q]_k \;=\; \mathcal{I}[\texttt{gate}\ g''\ q]_k \qquad\qquad (\llbracket g' \rrbracket \llbracket g \rrbracket = \llbracket g'' \rrbracket \wedge k > q)$$

The above identity can be derived from the definition of $\mathcal{I}[\cdot]_k$, properties of data-parallel operations, and from *map fusion*:

$$\texttt{map}\, f \circ \texttt{map}\, g \;=\; \texttt{map}\,(f \circ g)$$

Other fusion identities hold for consecutive `cntrl` gates performed on identical qubits and for consecutive `swap` gates.

The dq Futhark Library

dq

The **dq** Futhark library provides rich functionality for programming circuits.

```
module type gates = {
  type c                 -- complex numbers
  type q = i64           -- qubits
  type st[n] = [2**n]c   -- state vectors
  type^ stT[n] = *st[n] → *st[n]
  ...
  val gateH [n] : q → stT[n]
  val cntrlX[n] : (m:i64) → q → stT[n]
  val swap  [n] : q → stT[n]
  val swap2 [n] : (q:q) → (r:q) → stT[n]
  ...
  type ket[n] = [n]i64   -- ket vectors
  val fromKet[n] : ket[n] → *st[n]
  val toKet : (n:i64) → (i:i64) → ket[n]
  type dist[n] = [2**n](ket[n],f64)
  val dist    [n] : st[n] → dist[n]
  val distmax [n] : dist[n] → (ket[n],f64)
}
```

```
  -- Some utility functions
  val >*> 'a : (q → *a → *a) → (q → *a → *a)
                 → (q → *a → *a)
  val |*>   'a 'b : *a → (*a → *b) → *b
  val >* 'a 'b 'c : (*a → *b) → (*b → *c)
                      → (*a → *c)
  val repeat   'a : i64 → (i64 → *a → *a)
                      → *a → *a
```

Notice:

- Stars ($*$) on types indicate uniqueness...
- **dq** can serve as a target language or as a source language for directly expressing quantum algorithms.
- `https://github.com/diku-dk/dq`

Grover's Search Algorithm in Futhark

```
-- Grover's algorithm: search for the
-- index where oracle returns 1, which
-- it does for the binary encoding of
-- the argument (< 2**n) using a sub-
-- linear number of quantum operations.

import "dqfut"
open mk_gates(f64)

def grover_diff [n] : stT[n] =
     repeat n (gateH >*> gateX)
  >* gateH (n-1)
  >* cntrlX (n-1) 0
  >* gateH (n-1)
  >* repeat n (gateX >*> gateH)

def encNum [n] (i:i64) : stT[n] =
  λs → (loop (s,i) = (s,i) for n in n..>0 do
          if i % 2 == 0
          then (gateX (n-1) s, i/2)
          else (s,i/2)
        ).0
```

```
def oracle [n] i : stT[n] =
  encNum i >* cntrlZ (n-1) 0 >* encNum i

def grover (n:i64) (i:i64) : (ket[n], f64) =
  let k = 2**n |> f64.i64 |> f64.sqrt
          |> (*(f64.pi/4)) |> f64.ceil
          |> i64.f64
  let s = fromKet (replicate n 0)
      |*> repeat n gateH
      |*> repeat k (λ_ →
                     oracle i >*
                     grover_diff)
  in dist s |> distmax
```

Notice:

- Boxed $\boxed{\text{gate operations}}$ are fused!

Performance

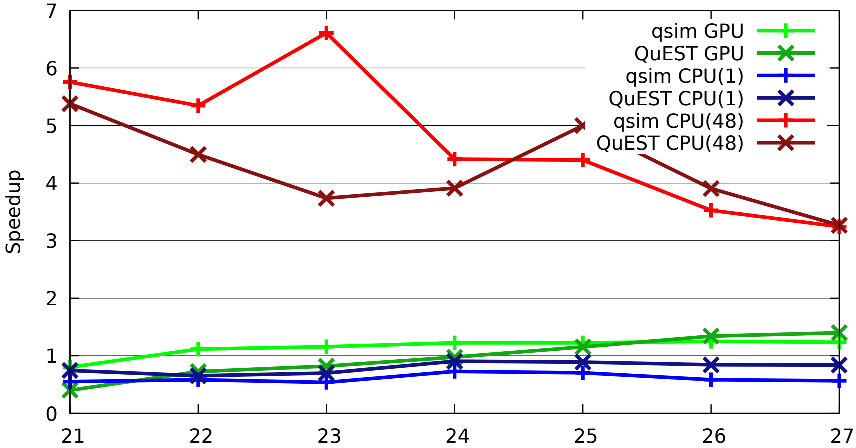
We compare with the highly tuned imperative simulators qsim and QuEST for three standard benchmarks (`qft`, `grover`, and `ghz`).

The results are promising:

- Fusion leads to significant speedups (i.e., for `grover`).
- We see slowdowns (sometimes a factor of 10) compared to the highly tuned simulators, which have pointed us towards issues with index transformations in Futhark.

Benchmark: GHz

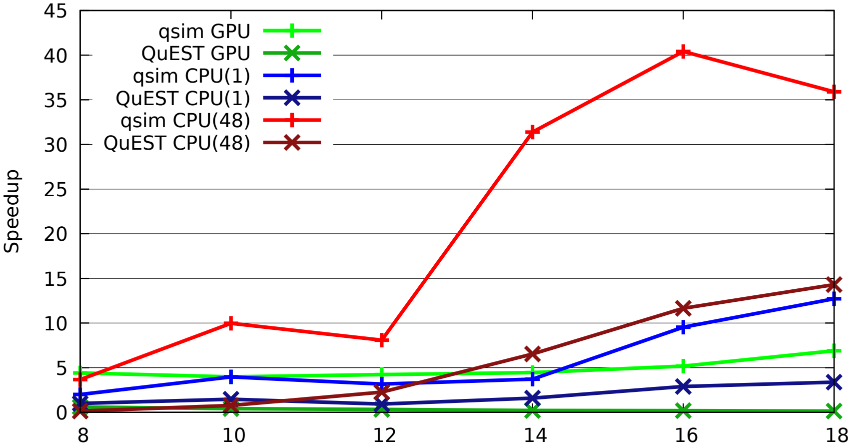
dq



Speedups > 1 indicate that the specified tool performs better than **dq** on the same platform.

Benchmark: Grover

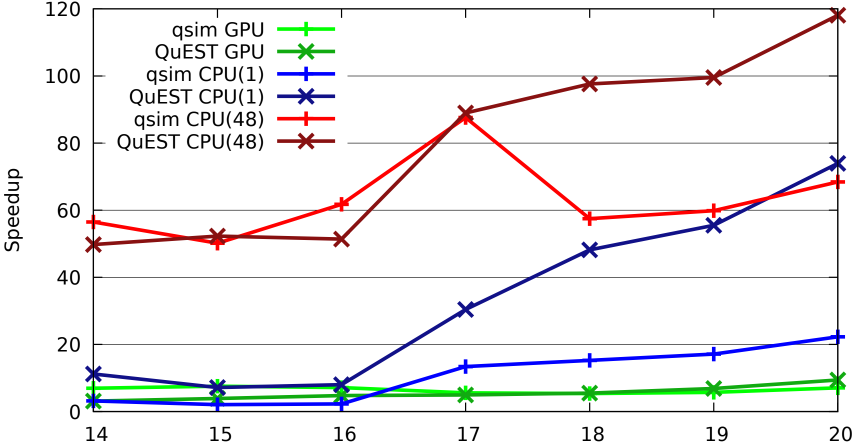
dq



Speedups > 1 indicate that the specified tool performs better than **dq** on the same platform.

Benchmark: QFT

dq



Speedups > 1 indicate that the specified tool performs better than **dq** on the same platform.

More in the Paper

- **Extensive derivations** of quantum gate operations.
- Derivation of operation for **swapping distant qubits**.
- Derivation showing that gate fusion can be derived from **map fusion**.
- Comparison with **related work** (e.g., other quantum simulation frameworks).
- Directions for **future work** (offline circuit optimisations, lazy state expansion, improvements of Futhark index-optimisations, ...)