High-Performance Quantum Compilation via JAX-Accelerated Tensor Pipelining and KAK Decomposition

Ata Khadivi
Quantum AI Research Team
github.com/atakhadiviom/Quantum-AI-Career
linkedin.com/in/atakhadivi

December 14, 2025

Abstract

We present a high-throughput, JAX-accelerated quantum compilation pipeline designed for the Google Sycamore processor. By leveraging Map-Reduce parallelism and XLA-compiled tensor processing, we eliminate the Python Global Interpreter Lock (GIL) overhead, achieving a **27.1x speedup** over sequential baselines (9,400 gates/second vs. 350 gates/second). Furthermore, we address the non-convex optimization landscape of two-qubit gate synthesis by introducing a **Targeted Initialization** strategy based on KAK decomposition. This approach utilizes invariant interaction coefficients to seed the variational optimizer, ensuring rapid convergence to high-fidelity solutions ($F \approx 1.0$) compared to random initialization.

1 Introduction

Quantum compilation—the transformation of logical quantum circuits into hardware-native operations—is a critical bottleneck in the quantum computing stack. As quantum processors scale, the latency introduced by classical compilation can exceed the coherence time of the qubits, preventing real-time error correction and dynamic circuit execution.

Standard compilation frameworks often suffer from significant overhead due to Python's interpreted nature and sequential execution models. This paper introduces a scalable architecture that integrates `JAX` for hardware-accelerated tensor processing and `concurrent.futures` for parallel task distribution. We focus specifically on compiling arbitrary two-qubit unitaries into the native Sycamore gate set used by Google's quantum processors.

2 Methodology

2.1 Parallel Architecture

To maximize hardware utilization, we implement a three-stage pipeline:

1. **Map-Reduce Parallelism:** We utilize `concurrent.futures.ProcessPoolExecutor` to distribute independent gate synthesis tasks across available CPU cores. This dynamic load balancing prevents straggler processes from stalling the pipeline.

2. **Data-Centric Design:** Inter-process communication overhead is minimized by passing raw numerical arrays (NumPy/JAX) rather than heavy object-oriented wrappers.

2.2 JAX-Accelerated Synthesis

The core synthesis kernel is implemented in JAX, enabling Just-In-Time (JIT) compilation to optimized XLA machine code.

- **Vectorization:** We employ `jax.vmap` to process batches of unitaries simultaneously, exploiting SIMD instructions.
- **Differentiable Branching:** Using `jax.lax.cond`, we implement hybrid logic that applies analytical decomposition for standard gates (e.g., CNOT) and variational optimization for arbitrary unitaries.

2.3 Targeted Initialization via KAK Decomposition

Synthesizing a target unitary U_{target} using the Sycamore gate (SYC) involves finding parameters $\vec{\theta}$ such that $U(\vec{\theta}) \approx U_{target}$. This is a non-convex optimization problem.

To avoid local minima and accelerate convergence, we employ KAK decomposition (Cartan decomposition). For any unitary $U \in SU(4)$, the KAK decomposition identifies invariant interaction coefficients (c_1, c_2, c_3). We map these coefficients to the initial parameters of our parameterized ansatz, placing the optimizer in the basin of attraction of the global minimum.

The loss function is defined as the infidelity:

$$\mathcal{L}(\vec{\theta}) = 1 - \frac{1}{4}\left|\mathrm{Tr}(U_{target}^{\dagger} U(\vec{\theta}))\right| \tag{1}$$

We use the Adam optimizer for gradient-based refinement.

3 Results

3.1 Computational Throughput

We evaluated the pipeline on a benchmark of 72,000 randomized two-qubit operations.

- **Baseline (Sequential):** $\sim$206 seconds.
- **JAX Parallel Pipeline:** 7.62 seconds.

This represents a **27.1x speedup**, demonstrating that the system is limited only by physical core count and XLA compilation throughput.

3.2 Synthesis Fidelity

We compared random initialization versus KAK-based Targeted Initialization for the variational solver (500 steps, Adam optimizer).

- **Random Initialization:** Converges to $F \approx 1.0$ but requires more iterations to escape local minima plateaus.
- **KAK Initialization:** Starts with significantly lower loss and converges rapidly to near-perfect fidelity ($F > 0.999$), validating the efficacy of topological coordinate extraction.

4 Conclusion

We have demonstrated a production-grade quantum compilation engine that combines the flexibility of variational algorithms with the speed of compiled tensor processing. By integrating KAK decomposition for intelligent initialization, we solve the fidelity-speed trade-off, enabling real-time compilation for next-generation quantum control systems.