Performance benchmarks for common circuit simulators

Evan

August 4, 2019

Abstract

abstract here

Contents

1 Methods

2 Simulator methods

2.1 General unitary matrix multiplication

A complete wavefunction simulator (cite Biamonte for what that means; I'll be reffing tensor networks anyways) generally applies the action of a gate as an einsum to the qubit axes acted on by the gate.

Worked example: Computing $U_2 U_1 \vec{\psi}$ For example, let U_1 be a unitary over a subset of k_1 and U_2 be qubits on a state defined over k_2, and the target state $|\psi\rangle$ be defined over n qubits. Then we have:

$$\vec{\psi} \in \mathbb{C}^{2^n} \tag{1}$$

$$U_1 \in \mathbb{C}^{2^{k_1} \times 2^{k_1}} \tag{2}$$

$$U_2 \in \mathbb{C}^{2^{k_2} \times 2^{k_2}} \tag{3}$$

$$\tag{4}$$

The following list details approaches to computing $U_2 U_1 |\psi\rangle$ in order of worst to best:

1. **kronecker product + matmul**: This requires resizing the matrices to n dimensions then performing matrix multiplication
   (a) kronecker product $U_1 \to U_1' \in \mathbb{C}^{2^{\mathbb{n}} \times 2^{\mathbb{n}}}$ using I_2 ($\mathbb{O}(2_1^k 2_1^k 2^{n-k_1} 2^{n-k_1}) = \mathbb{O}(2^{2n})$)
   (b) kronecker product $U_2 \to U_2' \in \mathbb{C}^{2^{\mathbb{n}} \times 2^{\mathbb{n}}}$ using I_2 ($\mathbb{O}(2_2^k 2_2^k 2^{n-k_2} 2^{n-k_2}) = \mathbb{O}(2^{2n})$)
   (c) matmul $U_2' U_1' \to U_f \in \mathbb{C}^{2^{\mathbb{n}} \times 2^{\mathbb{n}}}$ ($\mathbb{O}(2^{3n})$)[1]
   (d) matmul $U_f \vec{\psi}$ ($\mathbb{O}(2^{2n})$)
2. **kronecker product + matmul, associativity**: This swaps (c) and (d), taking advantage of the fact that $U\vec{\psi}$ has complexity $\mathbb{O}(2^{2n})$ for $U \in \mathbb{C}^{2^n \times 2^n}$, meaning both products cost $\mathbb{O}(2^{2n+1})$ instead of $\mathbb{O}(2^{3n} + 2^{2n})$. Conceptually this results from not having to calculate the full intermediate U_f.
3. **einsum**: An einstein summation allows general tensor transformations using repeated (summation) and permuted (free) indices (See Appendix A

[1] Assuming "schoolbook" matrix multiplication algorithm. For square matrix multiplication this can be improved to $\mathbb{O}(2^{2.807n})$ or even $\mathbb{O}(2^{2.37n})$ with ML but that's not the point of this worst-case example

for some common examples). The austere implementation results in a complexity of**?**:

$$\mathbb{O}\left(\left(\prod_{i}^{N_{\text{free}}} d_i\right)\left(\prod_{i}^{N_{\text{sum}}} d_i\right)\right) \tag{5}$$

where N_{free} is the number of free indices (indices of output), N_{sum} is the number of unique input indices between input tensors, and d_i is the size of the axis corresponding to the i-th index. [2]

Let unitaries and wavefunctions be represented as tensors of the shape:

$$U \in \mathbb{C}^{\overbrace{2\times 2\times\cdots}^{\text{k times}}} \tag{6}$$

$$\vec{\psi} \in \mathbb{C}^{\overbrace{2\times\cdots}^{\text{n times}}} \tag{7}$$

Then the einsum computing $U\vec{\psi}$ is indexed as

$$i_1 \cdots i_k, i_1 \cdots i_n \to i_1 \cdots i_n$$

has k summation indices and $(n-k)$ free indices, resulting in a complexity of $\mathbb{O}(2^n)$ since every tensor axis is dimension two. This is a two-fold speedup over square matrix math. Conceptually this results from ?!?!?!?!.

The complexity of method (3) is obviously a lower bound since *computing* all of the elements in $\vec{\psi}' = U\vec{\psi}$ elements can require no less than 2^n computations. However this is cannot be the minimum complexity of processing (FIXME: computing vs processing vs solving..?!?!?!?!) $U\vec{\psi}$, since the simple counterexample of $I\vec{\psi}$ requires exactly 0 computations. This motivates even more streamlined unitary action, which will be introduced in the following sections.

2.2 Clifford circuits

2.2.1 Example: X_k

This section begins with an example of applying a permutation to a tensor subspace (taken from the `apply_unitary` from the CIRQ library). Let X_k be the Pauli-X gate acting on the qubit indexed "k". Define the operation `slice_for_bitstring`$(T, \{i \text{ for i=1...k}\}, \vec{s})$ to access elements of tensor T for which the subspace corresponding to qubits $\{0, 1, \cdots, k\}$ is represented by the (little-endian) bitstring $\vec{s}$ (and so $|\vec{s}| = |\{i_k\}|$). TODO: example of this....

This slices for a subset of 2^{n-k} elements, but for an array stored with a known memory layout and stride the slice can be accessed in $\mathbb{O}(k)$ time (DEMONSTRATED EMPIRICALLY, PROVE THIS?). Then the action of X_k on $\vec{\psi}$ can be computed in two steps using a memory buffer T' sized the same as T:

[2] For example, the einsum string **ik,kj->ij** multiplies two matrices using free indices **i**, **j** and summation index **k**; if **i**, **j**, **k** index axes of sizes ℓ, m, n respectively, the complexity of this einsum is $\mathbb{O}((d_i d_j)(d_k)) = \mathbb{O}(\ell m n)$, the expected complexity of matrix multiplication.

- Compute $S_0 = \texttt{slice\_for\_bitstring}(T, \{k\}, (0))$ and $S_1 = \texttt{slice\_for\_bitstring}(T, \{k\}, (0))$ which slice T for all elements in which the k-th qubit is a "0" or "1" respectively

- Set $T'[S_1] = T[S_0]$ and $T'[S_0] = T[S_1]$. In plain English, this sets the amplitude for each basis state in T' to be the same as the amplitude in T where the k-th qubit of that basis state is flipped.

2.2.2 General Clifford operations

Section 2.2.1 demonstrated how a single-qubit gate acting on an n-qubit state can be implemented in $\mathbb{O}(n)$, which is exponential improvement over the best matrix-style operations (at the expense of exponential memory overhead in the number of qubits, i.e. the buffer state).

- Clifford circuits are easy to simulate (Gottesman). Computationally these are relatively sparse matrices.

Here we review an algorithmic app

3 Benchmark methods

This repo uses the PYTEST-BENCHMARK fixture to profile python code.

4 Results

Appendix A Appendix 1: Einsum operations

Some common einsum-compatible operations: transpose, inner produc, matrix multiplication, trace.