Performance benchmarks for common circuit simulators

Evan

September 10, 2019

Abstract

abstract here

Contents

1 Methods

2 Simulator methods

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

$$\mathcal{O}\left(\left(\prod_i^{N_{\text{free}}} d_i\right)\left(\prod_i^{N_{\text{sum}}} d_i\right)\right) \tag{5}$$

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

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 in tensored Hilbert space:

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

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

with the restriction that $n \geq 2k$ to ensure that the state and matrix are compatible for multiplication. Here I enforce that these objects retain as many axes as there are elements in the tensored space, so that they have a general tensor representation instead of being treated as matrices and vectors. Then the einsum computing $U\vec{\psi}$ is indexed as[3]

$$i_1 \cdots i_k, i'_1 \cdots i'_k i_1 \cdots i_n \to i_{k+1} \cdots i_n i'_1 \cdots i'_k$$

has k summation indices and $(n-2k)$ free indices, resulting in a complexity of $\mathcal{O}(2^{n+k})$ since every tensor axis is dimension two.

The complexity of method (3) does not saturate the lower bound for computing all of the elements in $\vec{\psi}' = U\vec{\psi}$ elements, which can require as few as 2^n operations (namely, $I\vec{\psi}$ using sparse multiplication saturates this bound). This motivates even more streamlined unitary action, which will be introduced in the following sections.

2.2 Nonlinear mappings

2.2.1 Tensor index notation

An wavefunction over n qubits defined in the tensored Hilbert space of Equation 7 is completely defined by k binary indices:

$$\psi = \psi_{i_1 i_2 \cdots i_n}$$

[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 $\mathcal{O}((d_i d_j)(d_k)) = \mathcal{O}(\ell m n)$, the expected complexity of matrix multiplication.

[3] Three worked examples:

(a) single-qubit matrix M acting on two-qubit state ψ ($k = 1$,$n = 2$): $(M\psi)_{jk} = M_{ij}\psi_{ik}$ has one summation index **i** and two free indices **j**,**k**, for $\mathcal{O}(2^3)$

(b) two-qubit matrix M acting on two-qubit state ψ ($k = 2$,$n = 2$): $(M\psi)_{\ell m} = M_{jk\ell m}\psi_{jk}$ has two summation indices **j**,**k** and two free indices ℓ,**m**, for $\mathcal{O}(2^4)$

(c) two-qubit matrix M acting on three-qubit state ψ ($k = 2$,$n = 3$): $(M\psi)_{\ell mn} = M_{jk\ell m}\psi_{jkn}$ has two summation indices **j**,**k** and three free indices ℓ,**m**,**n**, for $\mathcal{O}(2^5)$

1. Explain $(2,)^k$-shaped Tensors as nested 2×2 matrices for visualization/index intuition

2. someone else can explain tensor notation better than me.

2.2.2 Example: X_k

This section begins with an example of applying a permutation to a tensor subspace . Let $X^{(j)}$ be the Pauli-X gate acting on the qubit indexed "j". Let $\psi_{\dots i_{j-1}0 i_{j+1}\dots}$ be the subspace of ψ for which qubit j is in the "0" state. The action of $X^{(j)}$ is to swap the amplitudes of $|0\rangle_j$ and $|1\rangle_j$, which is reproduced by the following series of assignments:

1. Initialize $\phi := \psi_{\dots i_{j-1},0,i_{j+1}\dots}$ ($\mathcal{O}(2^{n-1})$ ops)
2. $\psi_{\dots i_{j-1},0,i_{j+1}\dots} \to \psi_{\dots i_{j-1},1,i_{j+1}\dots}$ ($\mathcal{O}(2^{n-1})$ ops)
3. $\psi_{\dots i_{j-1},1,i_{j+1}\dots} \to \phi$ ($\mathcal{O}(2^{n-1})$ ops)

Where the first step is required so that necessary information in ψ is not overwritten by assignments (2) and (3). The total complexity of this process is $\mathcal{O}(2^{n+1})$.

FIXME/TODO: I haven't been tracking read/write complexity for the other multiplication methods, which means this isn't the 2x speed up it ought to be compared to einsum.

2.2.3 General nonlinear assignments as tensor operations

This procedures of Section 2.2.2 can be generalized to apply permutation (FIXME: there is a special case for Hadamard, so I need a more general name for this set of matrices...) unitaries over k qubits to a state defined over n qubits. A bitstring $\vec{s} \in \{0,1\}^k$ can be used to select for the subspace of ψ in which k qubits are fixed to the state $|\vec{s}\rangle$ (here, the set of qubits is assumed to be contiguous for ease of notation, but this is not necessary in general). Then a the action of a permutation matrix with elements $|\vec{s}_i\rangle \langle\vec{s}_j|$ is accomplished by a series of tensor assignments of the form $\psi_{\vec{s}_j} \to \psi_{\vec{s}_i}$.

Section 2.2.2 demonstrated how a certain single-qubit gate acting on an n-qubit state can be implemented in $\mathcal{O}(2^n)$, which is at least 2× improvement over the best matrix-style operations (at the expense of exponential memory overhead in the number of qubits, i.e. the uninitialized buffer state).

TODO

- permutation matrices (plus phase)
- sparse matrices (Hadamard)
- why *doesn't* this work for T-gate

2.2.4 General Clifford operations

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

Here we introduce a formal equivalence between the methods of Section 2.2 with the efficient simulation methods for clifford circuits acting on stabilizer states.

3 Benchmark methods

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

Appendix A Appendix 1: Einsum operations

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