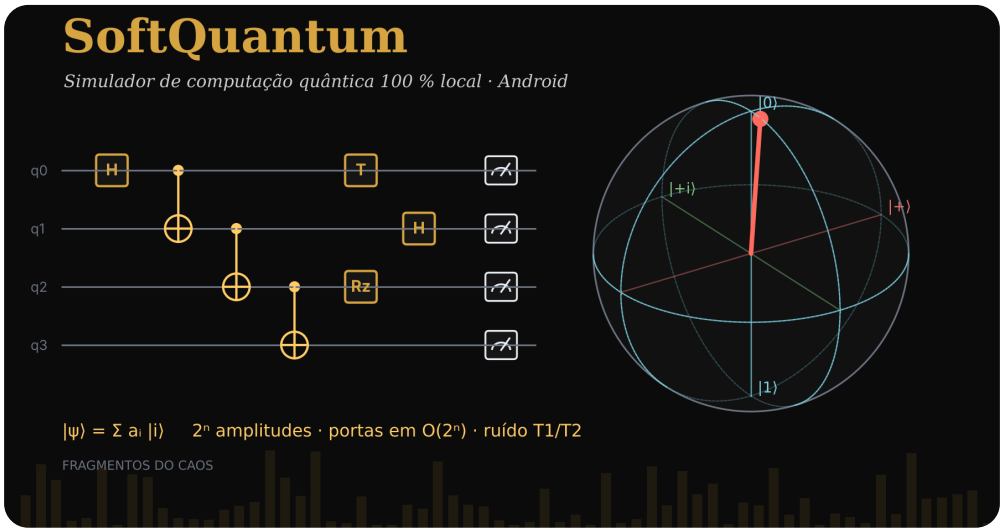
SoftQuantum: como simular um computador quântico no bolso, sem nuvem

Publicado em 2026-11-26 12:16:44



Simular um computador quântico no bolso, sem nuvem

Vector de estado, portas em $O(2^n)$, ruído por trajectórias e uma esfera de Bloch — tudo a correr num telemóvel Android.

> **Nota de abertura:**
> O SoftQuantum é um simulador clássico de circuitos quânticos para Android. Não é um computador quântico e não oferece vantagem quântica; os resultados de desempenho citados foram medidos num computador, não num telemóvel.

Simular um computador quântico num computador clássico parece uma contradição, mas é um exercício útil: permite aprender, validar circuitos pequenos e perceber, de forma concreta, porque é que a computação quântica é difícil de imitar. O **SoftQuantum** é um simulador de vector de estado para Android, desenvolvido pela SofteLabs, que corre integralmente no aparelho — sem

O custo exponencial: 2^n amplitudes

O estado de n qubits é um vector de 2^n amplitudes complexas. Cada amplitude ocupa 16 bytes (dois números de vírgula flutuante de 64 bits), pelo que a memória duplica a cada qubit acrescentado. É esta a razão pela qual o simulador se limita a 20 qubits e calcula o máximo permitido pela memória do aparelho.

Qubits	Amplitudes	Memória do estado
10	1 024	16 kB
16	65 536	1,0 MB
20	1 048 576	16,8 MB
30	1 073 741 824	17,2 GB (fora do alcance de um telemóvel)

Para ter uma escala de referência, o trabalho de Häner e Steiger simulou 45 qubits num supercomputador, com 8 192 nós e 0,5 petabytes de memória [4]. Um telemóvel está no extremo oposto desta curva, e o que se ganha em portabilidade paga-se em escala.

construir a matriz $2^n \times 2^n$ (o produto de Kronecker da porta com identidades) e multiplicá-la pelo vector. Com n = 20 a matriz teria 2^{40} entradas, cerca de 17,6 TB. Nenhuma porta exige isso: uma porta de um qubit actua apenas sobre pares de amplitudes cujos índices diferem no bit desse qubit.

```
def apply1(state, t, U):          # U = [[a,b],[c,d]] sobre o qubit
    for k in range(2**(n-1)):      # metade do vector: um par por it
        i0 = ((k >> t) << (t+1)) | (k & ((1 << t) - 1))   # insere u
        i1 = i0 | (1 << t)                                # o mesmo
        x, y = state[i0], state[i1]
        state[i0] = U[0][0]*x + U[0][1]*y
        state[i1] = U[1][0]*x + U[1][1]*y
```

Cada iteração trata um par independente, pelo que o custo de uma porta é $O(2^n)$ em tempo e $O(1)$ em memória adicional, e um circuito de g portas custa $O(g \cdot 2^n)$. Aplicar uma camada que toque em todos os qubits custa, assim, $O(n \cdot 2^n)$. Como os pares são independentes, o ciclo divide-se em blocos por várias threads — o equivalente a um *parallel for* de OpenMP.

As portas especiais são ainda mais baratas. As portas diagonais (Z, S, T, Rz, CZ, fase controlada, Z multi-controlado) limitam-se a multiplicar por uma fase as amplitudes seleccionadas por uma máscara de bits, sem

Medição, repetições e medições parciais

Num circuito sem ruído e sem medições a meio, o estado final calcula-se **uma só vez**. As repetições (*shots*) obtêm-se depois por amostragem: constrói-se a distribuição cumulativa das probabilidades $|a_i|^2$ e, para cada repetição, sorteia-se um número e procura-se o índice por pesquisa binária. Mil repetições custam uma ninharia comparadas com o cálculo do estado.

A medição parcial de um qubit obriga a mais cuidado: calcula-se P(1) como a soma de $|a_i|^2$ sobre os índices com esse bit a 1, sorteia-se o resultado, anulam-se as amplitudes do ramo não escolhido e renormaliza-se o vector. Quando o circuito tem medições a meio, cada repetição passa a ser uma trajectória própria, porque o colapso muda o futuro do estado.

> *Um simulador clássico não herda a velocidade de um computador quântico;*

Ruído por trajectórias de Monte Carlo

Um computador quântico real não é ideal. Para modelar o ruído sem recorrer a matrizes de densidade (que ocupam 4^n entradas em vez de 2^n), o SoftQuantum usa o método das funções de onda de Monte Carlo, ou trajectórias quânticas [3]. Cada trajectória é um vector de estado puro ao qual se aplicam, depois de cada porta, erros aleatórios; a média sobre muitas trajectórias reproduz o canal de ruído.

- **Despolarização**: X, Y ou Z com probabilidade p/3 cada, com p_1 nas portas de um qubit e p_2 por qubit nas portas de dois ou mais.
- **Bit-flip e phase-flip**: aplicam X ou Z com a probabilidade indicada.
- **Amortecimento de fase** (λ): equivale a um *phase-flip* com $p = (1 - \sqrt{(1 - \lambda)})/2$.
- **Amortecimento de amplitude** (γ, relaxação T1): operadores de Kraus K_0 e K_1; o «salto» ocorre com probabilidade $\gamma \cdot P(1)$, seguido de renormalização.

por porta: $\gamma = 1 - e^{-t/T1}$, $1/T_{\varphi} = 1/T2 - 1/(2 \cdot T1)$ e $\lambda = 1 - e^{-2t/T_{\varphi}}$, em que t é a duração da porta. A aplicação mede também a **fidelidade** face ao estado ideal, calculada como a média de $|\langle \psi_{ideal} | \psi_{trajectória} \rangle|^2$. Como o número de trajectórias é limitado por um orçamento de cálculo, repetições da mesma trajectória estão correlacionadas: a estatística é correcta, mas com menos graus de liberdade efectivos do que o número de repetições sugere.

A esfera de Bloch e o emaranhamento

O estado reduzido de um qubit num registo maior obtém-se a partir das mesmas amplitudes: $x = 2 \cdot \Sigma(a_r b_r + a_i b_i)$, $y = 2 \cdot \Sigma(a_r b_i - a_i b_r)$ e $z = P(0) - P(1)$, somando sobre os pares (a com o bit a 0, b com o bit a 1). O comprimento do vector $|r|$ é um indicador de emaranhamento: vale 1 para um estado puro e fica abaixo de 1 quando o qubit está emaranhado com o resto. Num par de Bell, cada qubit tem $|r| = 0$, no centro da esfera.

A esfera é desenhada em *Canvas* com uma projecção 3D feita à mão (rotação em torno de Z e inclinação), com eixos, três círculos máximos, o vector e um vector

Algoritmos de referência e validação

O SoftQuantum inclui Bell, GHZ, Deutsch–Jozsa, Bernstein–Vazirani [5], transformada de Fourier quântica (directa e inversa), Grover [2] com o número óptimo de iterações $\lfloor \pi/4 \cdot \sqrt{N} \rfloor$ e teleporte com medição diferida. Exporta e importa um subconjunto do OpenQASM 2.0 [1].

A validação foi feita numa JVM, num computador (não num telemóvel), com o mesmo código do motor:

DADOS ESSENCIAIS

- 40 circuitos aleatórios (todas as portas, 1 a 8 qubits) comparados com uma implementação densa independente: diferença máxima nula.
- Grover com 3 a 12 qubits: probabilidade do item marcado igual ou superior a 94,5 %.
- Bell, GHZ, QFT, Deutsch–Jozsa, Bernstein–Vazirani, teleporte e medição a meio dão os resultados teóricos.

- Desempenho num computador: 20 qubits → cerca de 1,6 ms por porta e 16,8 MB; Grover com 14 qubits (7 614 portas, 4 096 repetições) em 165 ms.
- Limite: 20 qubits e a memória do aparelho. Não medido num telemóvel.

Opções de desenho e limites

O pedido inicial sugeria C++ com NDK, instruções NEON e OpenMP. O SoftQuantum foi escrito em Kotlin puro, com paralelismo por *threads*, por duas razões práticas: o servidor de compilação pode não dispor do NDK, e uma falha de CMake impediria a compilação de toda a aplicação. O motor está isolado num único ficheiro, sem dependências do Android, e os ciclos críticos são simples, o que facilita uma futura migração para JNI. Não há, portanto, instruções SIMD explícitas, e é razoável esperar que uma versão nativa seja mais rápida.

Há limites que nenhuma optimização elimina. Acima de cerca de 20 qubits, um telemóvel esgota a memória; métodos como redes de tensores ou o formalismo dos estabilizadores permitiriam ir mais longe em circuitos com estrutura especial, mas não foram implementados. O

Experimente o SoftQuantum — grátis

Simulador de computação quântica para Android, 100 % local: sem anúncios, sem nuvem e sem permissões de rede. Disponível gratuitamente na loja de aplicações da SofteLabs.

Obter na loja SofteLabs

Nota Editorial

Este artigo descreve um simulador, não um computador quântico. Os números de desempenho foram obtidos num computador, com o código do motor, e não foram medidos em telemóvel; a interface Android foi verificada apenas quanto à sintaxe e compilada no servidor do autor.

A escolha de Kotlin em vez de C++ foi pragmática e é assumida como desvio face à

simplificação didáctica.

Os dados de contexto (como a simulação de 45 qubits) vêm das publicações citadas. Convém consultar as fontes originais antes de reutilizar qualquer valor.

Referências credíveis

- [1] Cross, Bishop, Smolin, Gambetta — Open Quantum Assembly Language (arXiv:1707.03429, 2017)
- [2] Lov K. Grover — A fast quantum mechanical algorithm for database search (arXiv:quant-ph/9605043; STOC 1996)
- [3] Dalibard, Castin, Mølmer — Wave-function approach to dissipative processes in quantum optics (Physical Review Letters 68, 580, 1992)
- [4] Häner, Steiger — 0.5 Petabyte Simulation of a 45-Qubit Quantum Circuit (arXiv:1704.01127; SC 2017)
- [5] Bernstein, Vazirani — Quantum Complexity Theory (SIAM Journal on Computing 26(5), 1411–1473, 1997)

Blogue Fragmentos do Caos

A verdade nasce onde o pensamento é livre.

PC-Chrome-News

☁️ GitHub Pages

🛰️ CodeBerg Pages

🌌 **Fragmentos do Caos:** Blogue • Ebooks • Carrossel

👁️ Esta página foi visitada ... vezes.

Contactos