# Energy Efficiency of Quantum Statevector Simulation at Scale

# Energy Efficiency of Quantum Statevector Simulation at Scale

Jakub Adamski
jakub.adamski@ed.ac.uk
EPCC, The University of Edinburgh
Edinburgh, UK

James Peter Richings
j.richings@epcc.ed.ac.uk
EPCC, The University of Edinburgh
Edinburgh, UK

Oliver Thomson Brown
o.brown@epcc.ed.ac.uk
EPCC, The University of Edinburgh
Edinburgh, UK



## ABSTRACT

Classical simulations are essential for the development of quantum computing, and their exponential scaling can easily fill any modern supercomputer. In this paper we consider the performance and energy consumption of large Quantum Fourier Transform (QFT) simulations run on ARCHER2, the UK's National Supercomputing Service, with QuEST toolkit. We take into account CPU clock frequency and node memory size, and use cache-blocking to rearrange the circuit, which minimises communications. We find that using 2.00 GHz instead of 2.25 GHz can save as much as 25% of energy at 5% increase in runtime. Higher node memory also has the potential to be more efficient, and cost the user fewer CUs, but at higher runtime penalty. Finally, we present a cache-blocking QFT circuit, which halves the required communication. All our optimisations combined result in 40% faster simulations and 35% energy savings in 44 qubit simulations on 4,096 ARCHER2 nodes.



## CCS CONCEPTS

• **Hardware** → **Impact on the environment**; • **Applied computing** → Physics; • **Theory of computation** → *Quantum computation theory*; • **Computing methodologies** → *Parallel computing methodologies.*

## 1 INTRODUCTION

Quantum computing has become a major area of research in recent years. In 2019, Google published a paper, in which they claimed to have successfully performed the classically intractable task of random circuit sampling [2], leading to an explosion of public interest and investment in quantum computers. The technology is still in its infancy and noise is still a major challenge for quantum hardware, so classical simulation plays an important role in algorithm development and testing. It is straightforward to fill any known or planned supercomputer with exact simulations of fewer than 50 qubits. Given the rapidly growing interest in these unprecedentedly large simulations, we believe it is important to consider their sustainability. ARCHER2 has been used to simulate 44-qubit systems using 4,096 nodes with 1 PB of memory[1] – a number far below the size of the recently announced 433-qubit 'Osprey' quantum processor from IBM[2]. This application domain is growing in popularity and often requires use of most of the available resources on a given target hardware, entailing high energy consumption. Focusing on its efficiency therefore plays a crucial role in HPC sustainability.

Most frequently, classical simulations are implemented by the Schrödinger's algorithm, which is a naïve but exact approach to simulating quantum circuits. The principle is to keep the whole statevector in memory, and evolve it by applying gates (a matrix-vector product) from the simulated circuit. This requires keeping track of all $2^n$ complex amplitudes, for $n$ qubits, and modifying them with each iteration. The maximum number of qubits in the register is bound by the total memory available on the target hardware. At the same time, the runtime of a circuit simulation scales linearly with the number of gates in the circuit, for a fixed size quantum register.

The main advantage of the statevector approach is that once a circuit is simulated, all amplitudes are available, which enables any required measurements to be made without the need to rerun the simulation. In contrast, many alternative approaches, like the Feynman path algorithm [1], or tensor network contraction [7], are restricted to a limited number of amplitudes in the output state in order to limit the memory usage. The trade-off is that those methods typically require repeated runs to perform all the desired measurements. This is also true of real quantum hardware, where the output of an individual run is always a classical bit string.

## 2 METHODOLOGY

### 2.1 QuEST Software

QuEST [5] is a *statevector simulator* implemented in C and parallelised with MPI+OpenMP, which acts as described in section 1. In order to evolve the statevector, each amplitude needs to be updated for every applied gate.

We can distinguish three kinds of quantum operators:

- *Fully local gates* – each amplitude can be updated without accessing other amplitudes. Mathematically, these are diagonal matrices.
- *Local memory gates* – each update is a linear combination of amplitudes on the same process. As matrices, those gates are block-diagonal, with blocks size less-or-equal to the number of amplitudes per process.
- *Distributed gates* – new amplitudes depend on amplitudes from other processes.

As long as the update dependencies are located on the same node, no message passing is required, and the gates can be applied relatively quickly. When the operators become distributed, MPI

[1] A feat in part inspired by a recent AWS HPC blog post where the authors also ran a 44 qubit simulation using QuEST: https://aws.amazon.com/blogs/hpc/simulating-44-qubit-quantum-circuits-using-aws-parallelcluster/.

[2] https://newsroom.ibm.com/2022-11-09-IBM-Unveils-400-Qubit-Plus-Quantum-Processor-and-Next-Generation-IBM-Quantum-System-Two.

takes up a large portion of the runtime. Hence, distributed gates are much more expensive to execute.

QuEST requires the statevector to be split evenly across $2^n$ processes. This ensures pairwise communication for any given gate. It also means that the entire local statevector needs to be exchanged – 64 GB per process on ARCHER2 assuming one MPI process per node. This was the case in all experiments presented here. Due to limitations of some implementations of MPI, individual messages cannot be larger than 2 GB, so the communication cannot be done in a single message. Instead, 32 messages are exchanged per distributed gate, and QuEST implements the communications as a sequence of blocking `MPI_Sendrecv`.

## 2.2 Potential Optimisations

We have identified three different strategies that help perform statevector simulations more efficiently.

(1) **Changing the clock frequency of the CPU.** ARCHER2 allows users to set the CPU frequency of computing nodes to reduce energy consumption. The default currently is 2.00 GHz (medium), but users have the ability to change this to either 1.50 GHz (low), or 2.25 GHz (high) with a SLURM environment parameter[3].

(2) **Using nodes with larger memory size.** There are two node types available on ARCHER2 – standard (256 GB), and high memory nodes (512 GB). We can use fewer high-mem nodes for a given size state vector simulation.

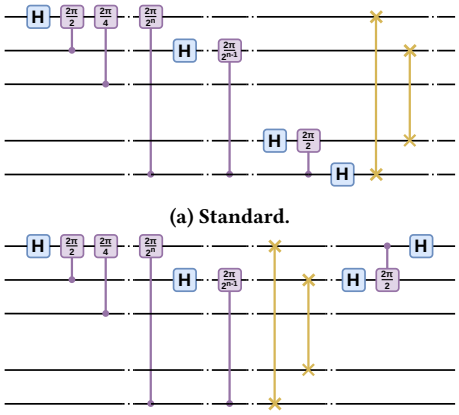
(3) **Transpiling the circuit to reduce communication via cache-blocking.** *Cache-blocking* is a technique that can improve memory locality of the circuit. When a gate acts on lower index qubits, the dependencies between statevector elements are located closer together. It is possible to lower a target qubit index by swapping it with another qubit. Each distributed SWAP gate requires communication, but it can be compensated if the target is frequently acted on. Although not implemented in QuEST, cache-blocking is already a part of some frameworks, such as Qiskit, where it serves as a means of distribution to multiple processes [3], and cuQuantum for multi-GPU support [4]. However, it is the first time this optimisation is used at such a scale and for energy reasons.

## 2.3 Circuits

The main circuit used in this paper is the *Quantum Fourier Transform (QFT)*, shown on fig. 1a. It is a common subroutine of larger quantum algorithms, like Quantum Phase Estimation.

The QFT is straightforward to cache-block without adding new gates, as it already features SWAP gates at the end. By shifting them to the left, we can guarantee that all Hadamard gates are applied to local qubits. Gates to the right of the swaps need to be vertically flipped. As a result, the only communication remains in the distributed SWAP gates. An example is shown in fig. 1b, where the last two Hadamard gates were made local.

To test the effects of communication, two other benchmarking circuits were designed – the Hadamard gate benchmark and the SWAP gate benchmark. Their structure is simple, consisting of $k$ gates applied sequentially to the same target qubits. Despite lack of practical use, they give insight into properties of statevector simulations. Importantly, a Hadamard benchmark on the last qubit of the system should be the worst-case simulation scenario, since all gates in the circuit are guaranteed to be distributed (provided the simulation spans multiple nodes).



**(a) Standard.**

**(b) Cache-blocked.**

**Figure 1: Standard and cache-blocked QFT circuit.**

## 2.4 Energy Measurements and Profiling

The energy consumption of statevector simulations can be retrieved by querying SLURM on ARCHER2, which uses power counters on the nodes to determine energy usage. However, this is missing the energy used by the network, which we estimated with the following formula: $E_{net} = n_s \cdot \bar{P}_s \cdot \Delta t$. The value $n_s$ represents the number of switches in the utilised network – 1 switch per 8 nodes on ARCHER2. The typical average power of a switch under load on ARCHER2 $\bar{P}_s$ is equal to 235 W. Finally, $\Delta t$ is the total runtime of the circuit. The estimated switch energy and the energy extracted from SLURM are added together to calculate the total energy consumed by the jobs being studied. We have omitted other sources of energy consumption during the job, as we do not have values for other rack level overheads nor do we include energy consumed in cooling the system.

# 3 RESULTS

We ran a QFT circuit at register sizes from 33 to 44 qubits, using the minimum possible number of nodes to fit the statevector, as well as the Hadamard and SWAP circuits to calculate the cost of distributed gates[4].

## 3.1 Clock Frequency and Node Memory

To benchmark the most efficient combination of the CPU frequency and available memory, we ran the QFT circuit. Results are shown in fig. 2. Additional buffers are required in the MPI implementation, doubling the overall memory requirement. As a result, 33 qubits will fit on a standard node, but 4 nodes are required for a 34 qubit

[3] https://docs.archer2.ac.uk/user-guide/energy/#controlling-cpu-frequency.

[4] Resources used are available at https://github.com/jjacobx/arb23.

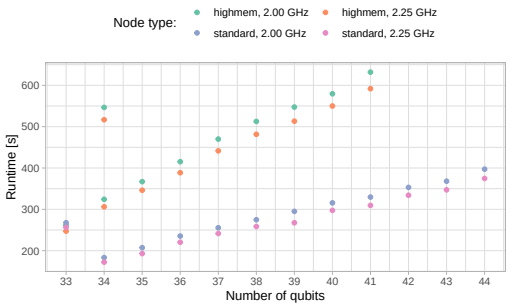



**Figure 2: Runtimes of the QFT circuit simulations with different register size. Standard nodes have 256 GB, highmem nodes have 512 GB.**

simulation. The 33 qubit standard node results, and slower of the two 34 qubit high memory node results are single node runs.

QFT runtimes scale linearly, due to the number of distributed gates rising linearly. The actual number of gates grows quadratically. High memory nodes are slower, but less than twice as slow as the standard nodes. Higher memory means twice as much data in the local memory needs to be recomputed at each gate application. A maximum of 41 qubits could be simulated on 256 high memory nodes, and 44 qubits on 4,096 standard nodes.

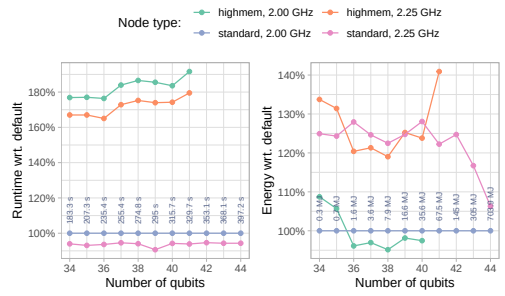



**Figure 3: Comparison of the default medium frequency, standard nodes to other setups.**

Since the standard node medium frequency setup is currently the default on ARCHER2, we plotted its fractional comparison to other setups on fig. 3, in terms of runtime and energy consumption. By interpreting both plots together, it is possible to assess which configuration is most optimal. The standard high frequency setup is consistently 5% to 10% faster than the default, but it uses around 25% more energy, up to 43 qubits, where the consumption significantly decreases. Therefore, we conclude that the defaults are appropriate for most simulations.

The use of high memory nodes drastically increases the runtime, however the CU cost of high memory simulations is lower than for standard memory. High frequency on high memory nodes required from 20% to over 40% more energy. We found that the lowest frequency available on ARCHER2 (1.5 GHz) was not of benefit in either case due to a large increase in runtime, and so these runs are omitted from our figures.

## 3.2 Cache Blocking

The cache blocking approach was described in section 2.2. In addition to the QFT, we ran the two other benchmarks from section 2.3, with 50 gates in each circuit. During the investigation, we noted that QuEST uses a series of blocking communications. We rewrote the message passing function to use non-blocking communication, which allows multiple messages to be sent and received in parallel when using an interconnect with high bandwidth. Each of our experiments here includes results from both the default QuEST implementation and our modified version.

| Qubit Index | Blocking | | Non-blocking | |
|---|---|---|---|---|
| | Time | Energy | Time | Energy |
| 29 | 0.51 s | 15.3 kJ | 0.53 s | 15.0 kJ |
| 30 | 0.59 s | 15.7 kJ | 0.74 s | 18.7 kJ |
| 31 | 0.80 s | 20.8 kJ | 0.97 s | 24.2 kJ |
| 32 | 9.63 s | 191 kJ | 8.82 s | 179 kJ |

**Table 1: Time/energy per gate in the Hadamard benchmark on qubits 29–32 using blocking/non-blocking MPI.**

Fifty Hadamard gates were applied to a specific qubit, from 0 to 37, on 64 nodes and per-gate values calculated. Table 1 presents the results on qubits 29–32. Up until qubit 29 the time per gate is roughly constant at 0.5 s, and the energy is approximately 15 kJ. At qubits 30 and 31 both values slightly increase, towards 1.0 s and 25 kJ due to data dependencies spreading to multiple NUMA regions. At qubit 32 and above, the runtime drastically rises to almost 10 s per gate for blocking communication, and 9 s for our non-blocking version, similarly, the energy shoots up to 190 kJ and 180 kJ respectively. The twenty-fold increase in runtime is caused by MPI, which now needs to communicate all the local elements of the statevector to another process. It is clear that non-blocking messages mitigate some of the communication costs.


**Figure 4: Energy consumption of the SWAP benchmark.**

Next, a similar benchmark with 50 SWAP gates was run. The SWAP gates require two targets, and as long as one is in the distributed regime, the operation entails communication. Since it is expensive to try all possible combinations of targets, we instead selected 5 local targets $[0, 4, 8, 12, 16]$, and 3 distributed targets $[35, 36, 36]$. The energy results are shown on fig. 4. Again, the non-blocking communication visibly improved the performance – with

blocking communication the time per gate ranges from 9.0 s to 9.75 s and the energy consumption is from 180 kJ to 195 kJ; meanwhile, our non-blocking implementation requires 8.25 s to 9.0 s per gate and 160 kJ to 180 kJ.

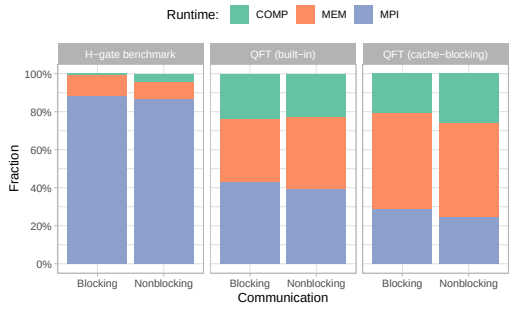



**Figure 5: Profiles of the Hadamard and QFT benchmarks.**

Lastly, we profiled the different QFT implementations and compared them to the worst-case communication scenario (last qubit Hadamard benchmark). The built-in QFT in QuEST implements the circuit shown in fig. 1a, but the controlled phase gates are applied more efficiently. The cache blocking circuit is as fig. 1b, but the swaps are done after the 30th Hadamard gate to prevent any increase in gate execution time, and the controlled phase gates used the optimised QuEST implementation.

The profiles of each run are presented on fig. 5. In the Hadamard benchmark MPI completely dominates the runtime. The QFT gates are mostly local, so communication only takes up to 43% of runtime, and the rest is split roughly 2:1 between memory access and computation. By applying our optimisation, we managed to reduce communication to 25%.

### 3.3 Best QFT Performance

Our final experiments were runs designed to use a significant share of ARCHER2 (which has 5,860 nodes), with and without optimisations. We tested the QFT on 43 and 44 qubits, using 2,048 and 4,096 nodes respectively. The 'Fast' version uses cache-blocking to move all the Hadamard gates out of the distributed regime, and implements non-blocking communication for the SWAP gates. Results are shown in table 2.

| Experiment | Built-in | Fast | Built-in | Fast |
|---|---|---|---|---|
| Qubits | 43 | 43 | 44 | 44 |
| Nodes | 2048 | 2048 | 4096 | 4096 |
| Runtime | 417 s | 270 s | 476 s | 285 s |
| Energy | 294 MJ | 206 MJ | 664 MJ | 431 MJ |

**Table 2: Runtime and energy consumption of large QFT runs on ARCHER2. 'Built-in' is using the standard QuEST QFT, 'Fast' is our modified version.**

We achieved 35% and 40% improvements in runtime, along with 30% and 35% reductions in energy for the 2,048-node and 4,096-node runs respectively. The biggest energy improvement was 233 MJ, which is around 65 kWh saved in a little more than 3 minutes.

## 4 CONCLUSIONS AND FUTURE WORK

We explored multiple ways to mitigate the energy consumption on ARCHER2 with QuEST framework. First, we highlighted the importance of using moderate CPU frequencies, which reduces the energy consumption with only a small runtime penalty. We note that the benefits end at 2.00 GHz – any further decrease worsens the runtime while keeping the energy usage fixed. Second, we investigated whether high memory nodes are beneficial. Results were mixed, although high memory nodes are more efficient for the CU budget, the energy consumption is sometimes slightly higher and other times slightly lower than for the standard nodes. In particular, due to memory bandwidth being a limiting factor, we do not recommend specifying high-memory nodes over more balanced nodes if selecting hardware for this application. Last, we explored transpiling the simulated circuit to be *cache-blocking*, which can reduce the number of distributed operations. The QFT is a perfect circuit for cache blocking, as SWAP gates are already a part of it. We also found that communication in QuEST could be improved on high-end networks using non-blocking communication. Applying all these optimisations, we managed to simulate 44 qubit QFT 40% faster, saving 35% energy. This work provides energy benchmarks both for both classical simulation codes and real quantum hardware to consider.

In future work we will further explore cache-blocking. If SWAP gates are the only distributed operations, communication could potentially be halved, as swapping only modifies half of the statevector. With this improvement, ARCHER2 could possibly simulate up to 45 qubits. It would also be useful to implement a cache-blocking transpiler, for example as an LLVM optimisation pass using the newly developed QIR [6]. We believe it would be beneficial to reimplement QuEST's core data-structures using a complex data type rather than separate real and imaginary arrays, in order to improve data locality. Finally, we will explore the impact on performance and energy usage of porting QuEST to multiple GPUs.

## ACKNOWLEDGMENTS

This work used the ARCHER2 UK National Supercomputing Service (https://www.archer2.ac.uk).

## REFERENCES

[1] Scott Aaronson and Lijie Chen. 2017. Complexity-Theoretic Foundations of Quantum Supremacy Experiments. In *Proceedings of the 32nd Computational Complexity Conference* (Riga, Latvia) *(CCC '17)*. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, DEU, Article 22, 67 pages.

[2] Frank Arute et al. 2019. Quantum supremacy using a programmable superconducting processor. *Nature* 574, 7779 (01 Oct 2019), 505–510. https://doi.org/10.1038/s41586-019-1666-5

[3] Jun Doi and Hiroshi Horii. 2020. Cache Blocking Technique to Large Scale Quantum Computing Simulation on Supercomputers. In *2020 IEEE International Conference on Quantum Computing and Engineering (QCE)*. Institute of Electrical and Electronics Engineers Inc., Denver, CO, USA, 212–222. https://doi.org/10.1109/QCE49297.2020.00035

[4] Jennifer Faj, Ivy Peng, Jacob Wahlgren, and Stefano Markidis. 2023. Quantum Computer Simulations at Warp Speed: Assessing the Impact of GPU Acceleration. arXiv:2307.14860 [cs.PF]

[5] Tyson Jones, Anna Brown, Ian Bush, and Simon C. Benjamin. 2019. QuEST and High Performance Simulation of Quantum Computers. *Scientific Reports* 9 (12 2019), 10736. Issue 1. https://doi.org/10.1038/s41598-019-47174-9

[6] Thomas Lubinski, Cassandra Granade, Amos Anderson, Alan Geller, Martin Roetteler, Andrei Petrenko, and Bettina Heim. 2022. Advancing hybrid quantum–classical computation with real-time execution. *Frontiers in Physics* 10 (2022). https://doi.org/10.3389/fphy.2022.940293

[7] Feng Pan and Pan Zhang. 2022. Simulation of Quantum Circuits Using the Big-Batch Tensor Network Method. *Physical Review Letters* 128 (1 2022), 030501. Issue 3. https://doi.org/10.1103/PhysRevLett.128.030501