

Quantum factorization simulation as a benchmark for HPC

Santiago Ignacio Betelu

UNT Mathematics Department adjunct professor

Data Vortex Technologies chief scientist



Why quantum factorization as a benchmark

- Quantum factorization a widely known & relevant problem
- Easy to validate and understand
- Each additional qubit doubles RAM usage, CPU power and internode communication: good to test large machines
- Runs in a reasonable time: 1-3 hours
- Portable with less than 300 lines of C and MPI
- It just runs: no input or special knowledge from user
- Runs from a laptop to a large supercomputer

Summary of the benchmark

- Simulates a quantum computer with state $|\psi\rangle = \sum_{x=0}^{(2^Q-1)} c_x |x\rangle$
- The test consists on running a simplified Shor's algorithm* with increasing number of Q qubits until resources are exhausted.
- Only timing of Fourier Transform AQFT, not the modular exponentiation
- For each Q run the simplified Shor's algorithm to factorize an integer n

$$n = p \cdot q$$

- p and q are chosen to maximize the Euler's totient function

$$\varphi = (p - 1) \cdot (q - 1)$$

with the constraint $n^2 \leq 2^Q < 2n^2$

- Then verify that the total probability under peaks is larger than $1/2$

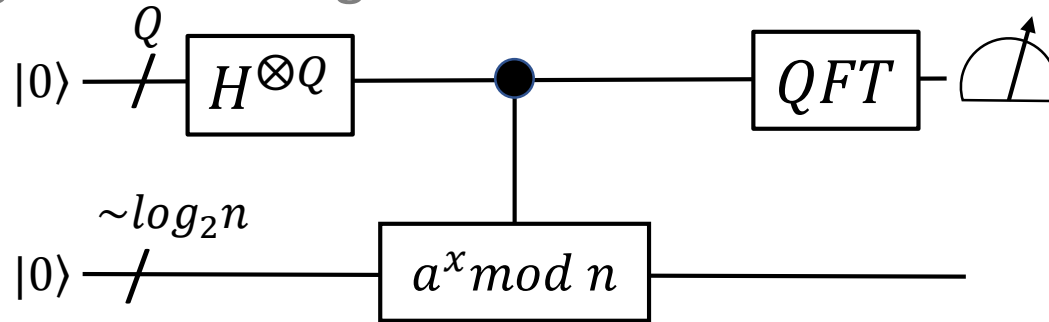
$$P = \sum_{peaks} |c_x|^2 > \frac{1}{2}$$

- Peaks located at $x = \frac{2^Q}{(p-1)(q-1)} \times \text{integer}$

* PW Shor, "Polynomial-Time algorithms for prime Factorization and Discrete Logarithms on a Quantum Computer", SIAM J. Comp 1997

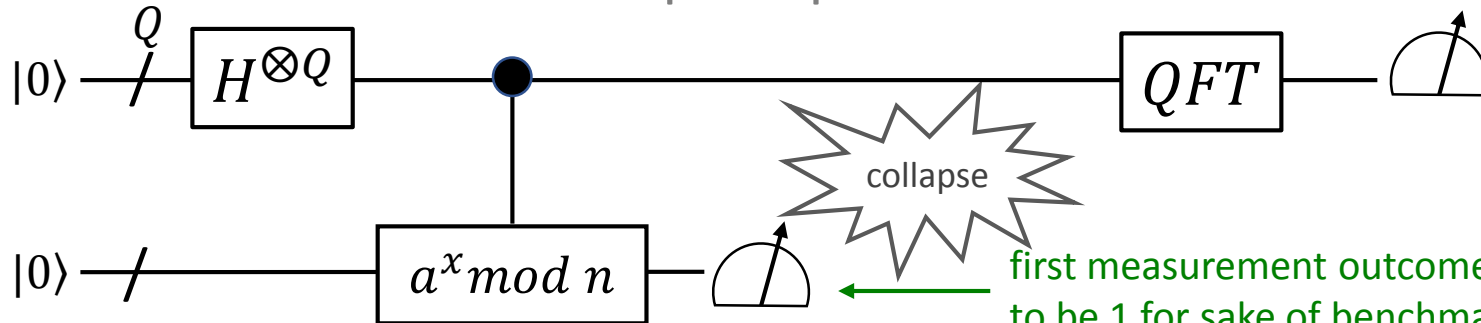
Use deferred measurement principle to save qubits

Original Shor's algorithm



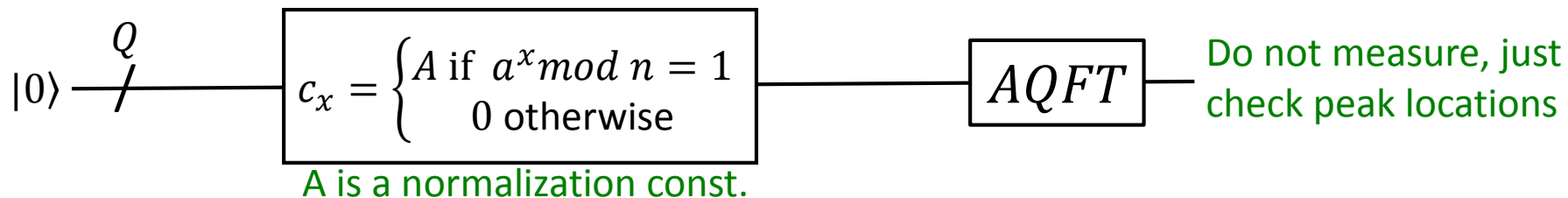
This measurement is inconvenient in a benchmark

With deferred measurement principle

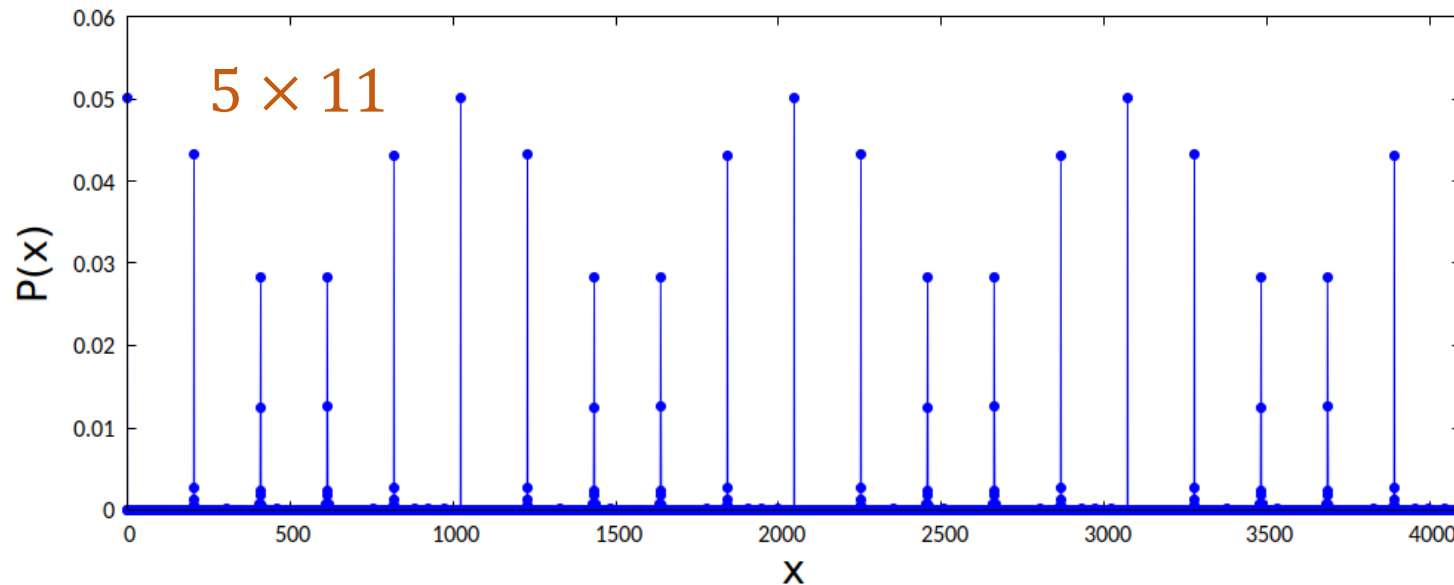
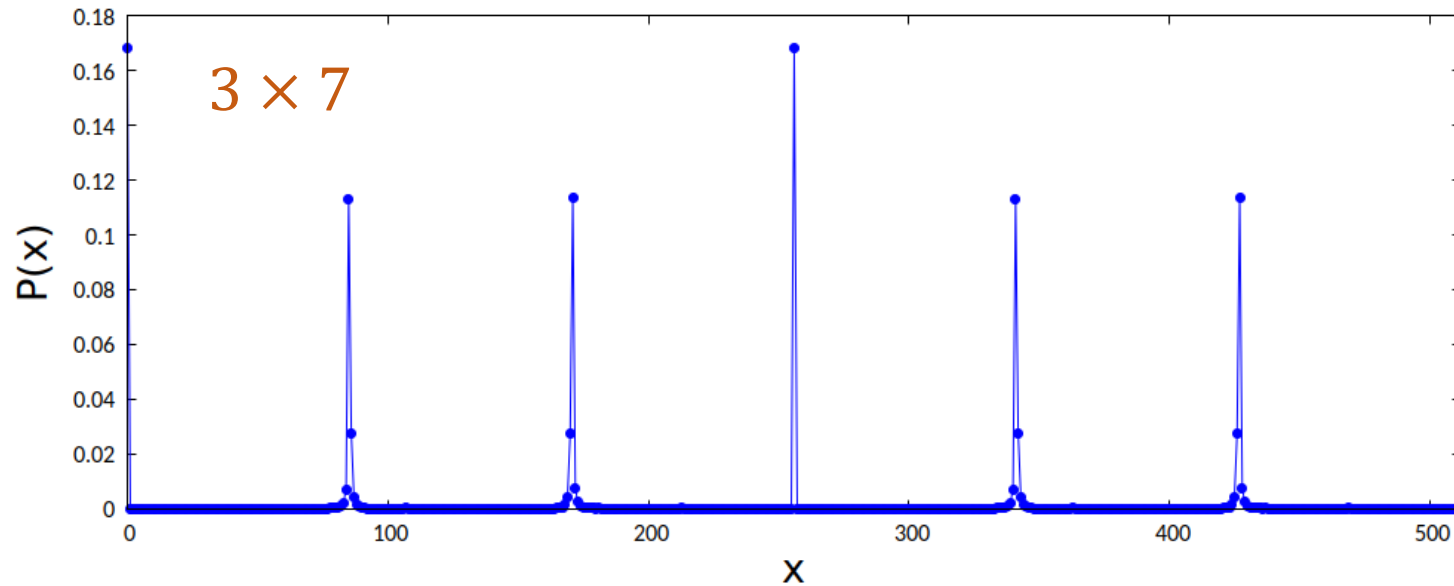


first measurement outcome chosen to be 1 for sake of benchmarking

This benchmark saves $\log_2 n$ qubits, only AQFT is timed



Verify location of peaks for each $n = p \cdot q$



List of factorizations used in the test

$n = p \cdot q$ chosen to maximize $(p - 1) \cdot (q - 1)$, $n^2 \leq 2^Q < 2n^2$
 $p < q$, this way the period of $2^x \bmod n$ is maximized

Q	$p \times q$
9	3×7
10	3×7
11	3×13
12	5×11
13	7×11
14	7×71
15	7×23
16	11×23
17	11×31
18	17×29
19	23×31
20	19×53
21	23×61

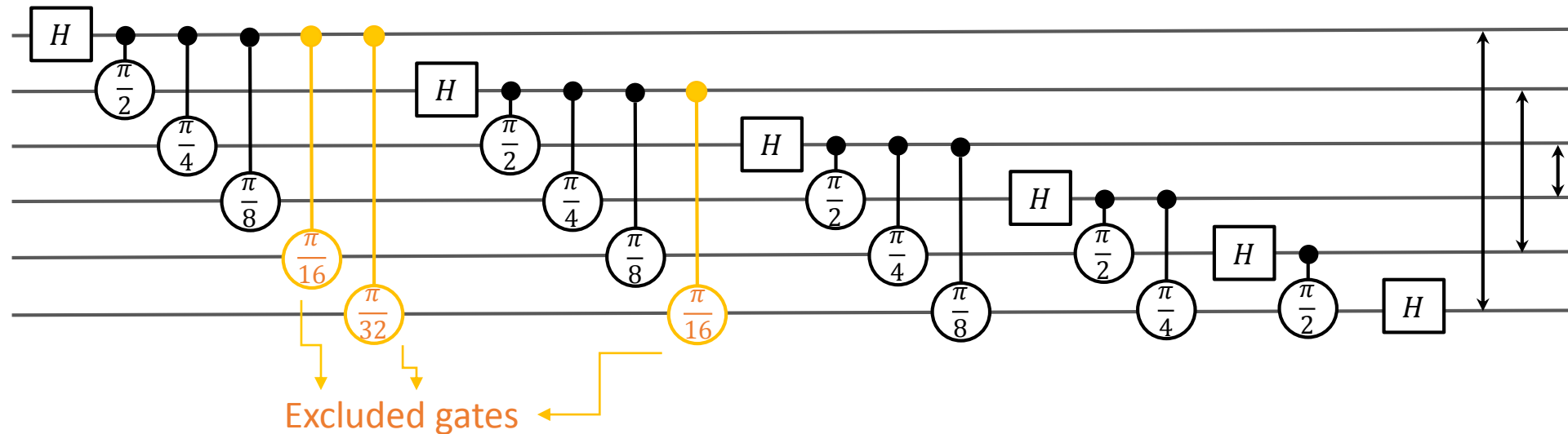
Q	$p \times q$
22	23×89
23	43×67
24	61×67
25	53×109
26	79×103
27	71×163
28	83×197
29	101×229
30	137×239
31	149×311
32	233×281
33	211×439
34	283×463

Q	$p \times q$
35	241×769
36	503×521
37	389×953
38	557×941
39	859×863
40	911×1151
41	1039×1427
42	1399×1499
43	1669×1777
44	1787×2347
45	2039×2909
46	2357×3559
47	2609×4547

Q	$p \times q$
48	4093×4099
49	3709×6397
50	5471×6133
51	5503×8623
52	8011×8377
53	8537×11117
54	11119×12071
55	12757×14879
56	12941×20743
57	17837×21283
58	22717×23633
59	24847×30557
60	28579×37571

Use AQFT* because it is faster than QFT


- For each H gate we compute $1 + \log_2 Q$ phase gates
- Phase gates computed with one memory access per state
- Complexity $O(Q \log Q)$ versus $O(Q^2)$



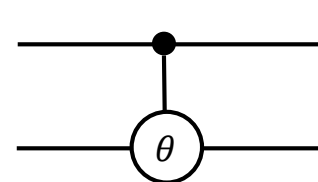
Results in the approximation $f_y \cong \frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} c_x e^{2\pi i x y / N}$

* D Coppersmith, An Approximate Fourier Transform Useful in Quantum Factoring, IBM report 1994
A Barenco et al, Approximate Quantum Fourier Transform and Decoherence, 1996

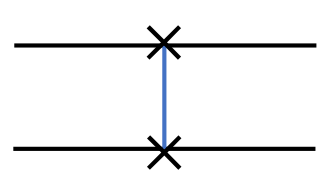
AQFT uses 3 types of gates


$$= \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

Hadamard


$$= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & e^{i\theta} \end{bmatrix}$$

Controlled phase


$$= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Swap qubits

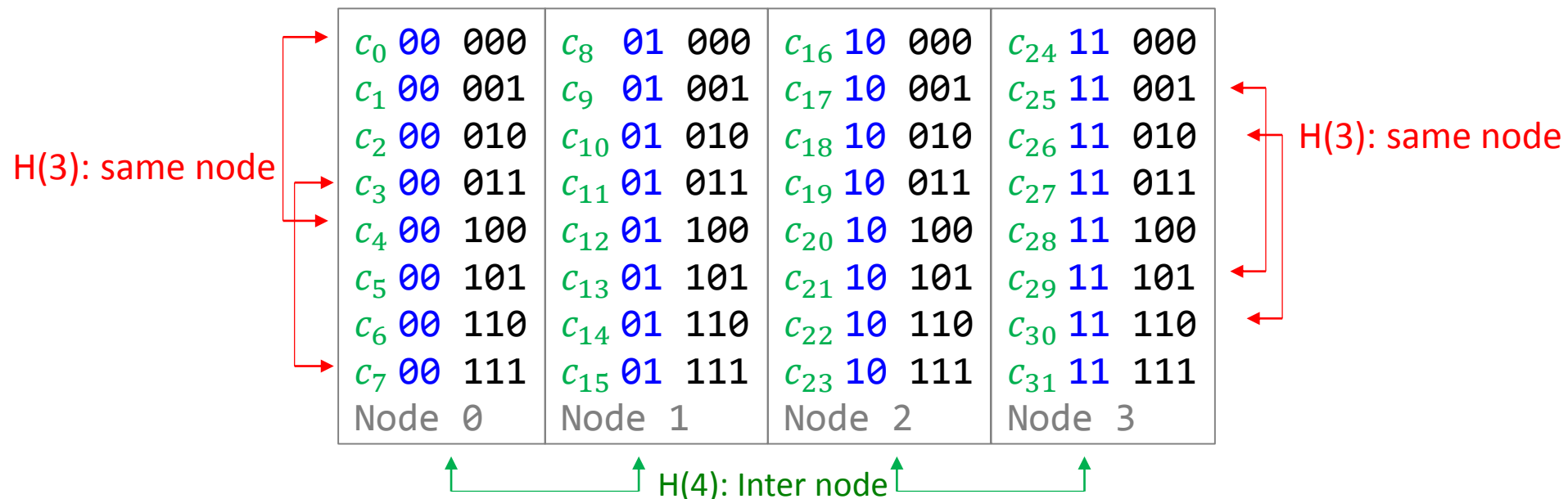
- The essentially heavy part of computation is the Approximate Fourier Transform, computed simulating gates
- Modular exponentiation $a^x \bmod n$ is not implemented gate-by-gate and it is not timed. We compute directly in C, and the computation time is negligible.
- Exponentiation could be implemented with stabilizer-group gates, thus simulated efficiently, not in this benchmark

How are the coefficients distributed between nodes

- The state of the quantum registers $|\psi\rangle = \sum_{x=0}^{(2^Q-1)} c_x |x\rangle$ is distributed among nodes
- A basis state is described in binary as

$$|x\rangle = |b_Q b_{Q-1} \dots b_{Q-M+1}\rangle \otimes |b_L b_{L-1} \dots b_2 b_1\rangle$$
- The black digits denote indices of amplitudes within each node.
- The blue digits denote node index.
- The number of nodes is $p = 2^M$ $Q = L + M$
- For example the Hadamard gate at qubit q combines the amplitudes c_x and c_y where x and y differ only in bit q .
- Depending on the value of q , we may be combining data between nodes or within node

Example H gate with $Q=5$, $L=3$, $M=2$



How to use the benchmark (MPI version)

First prepare a batch file with number of nodes and number of tasks equal to **a power of 2**:

```
#SBATCH -o output-8nodesx64cores
#SBATCH --nodes=8      # 8 nodes
#SBATCH -n 256         # 64 cores per node
#SBATCH -p normal      # for KNL processors
#SBATCH -t 02:00:00    # usually less than 2 hours
ibrun ./quansimbench
```

```
> mpicc -Ofast quansimbench.c -o quansimbench -lm -Wall
> sbatch quansimbench.batch
```

May need to add processor specific options such as

- xCORE-AVX512 for skx

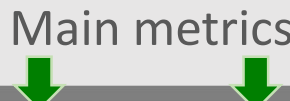
- xMIC-AVX512 for knl

```
cksum quansimbench.c = 219669803 14629
```

Typical output

- This example runs with 16384 KNL cores in 256 nodes
- Memory limited
- ~ 1 hour run cumulative

Memory exhausted at 41 qubits

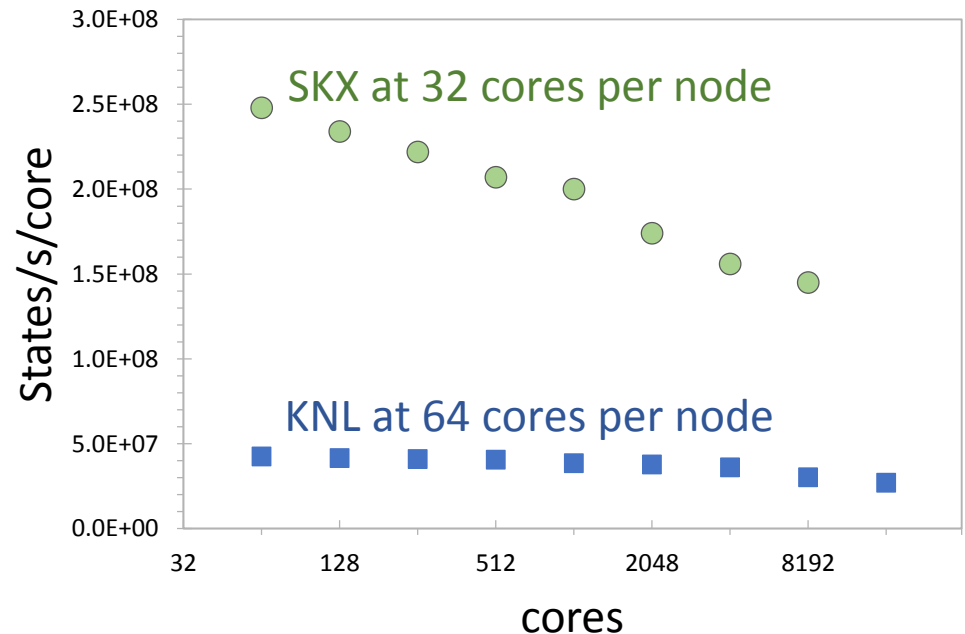
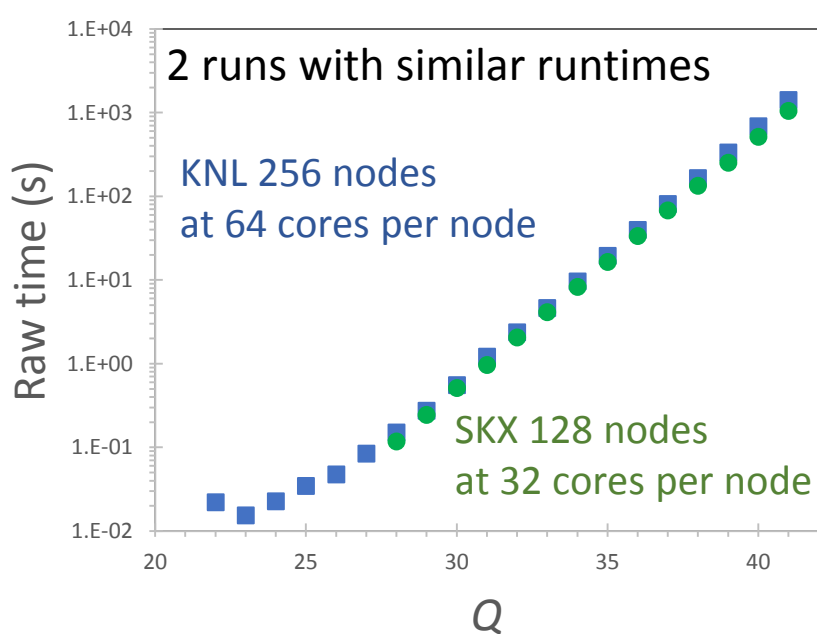


Qubits	Factors	probability	rawtime	states/s/core	pass
14	7*17	0.743078	2.6930e-01	2.4879e+02	yes!
15	7*23	0.727522	1.0310e-02	1.3967e+04	yes!
16	11*23	0.760696	1.4716e-02	2.4191e+04	yes!
17	11*31	0.675652	9.7967e-03	7.7577e+04	yes!
18	17*29	0.756086	1.9258e-02	8.4743e+04	yes!
19	23*31	0.749676	1.8767e-02	1.8415e+05	yes!
20	19*53	0.765709	1.4059e-02	5.2350e+05	yes!
21	23*61	0.765677	1.5442e-02	1.0030e+06	yes!
22	23*89	0.678047	2.2167e-02	1.4782e+06	yes!
23	43*67	0.763136	1.6865e-02	4.0682e+06	yes!
24	61*67	0.763811	2.2675e-02	6.3676e+06	yes!
25	53*109	0.763169	3.4554e-02	8.7127e+06	yes!
26	79*103	0.761923	4.7436e-02	1.3298e+07	yes!
27	71*163	0.762637	8.4091e-02	1.5587e+07	yes!
28	83*197	0.762689	1.5091e-01	1.8131e+07	yes!
29	101*229	0.762295	2.7516e-01	2.0602e+07	yes!
30	137*239	0.76021	5.5607e-01	2.1214e+07	yes!
31	149*311	0.760909	1.2061e+00	2.0213e+07	yes!
32	233*281	0.770465	2.3610e+00	2.4316e+07	yes!
33	211*439	0.770119	4.6289e+00	2.5598e+07	yes!
34	283*463	0.769982	9.5839e+00	2.5602e+07	yes!
35	241*769	0.785465	1.9453e+01	2.5982e+07	yes!
36	503*521	0.770013	3.9763e+01	2.6265e+07	yes!
37	389*953	0.769878	8.0994e+01	2.6514e+07	yes!
38	557*941	0.769711	1.6589e+02	2.6700e+07	yes!
39	859*863	0.769253	3.3575e+02	2.7084e+07	yes!
40	911*1151	0.769082	6.8747e+02	2.7235e+07	yes!
41	1039*1427	0.768945	1.4074e+03	2.7275e+07	yes!

A typical comparison in TACC's Stampede 2

- Compare KNL Xeon Phi 7250 vs. SKX Xeon 8160 nodes
- From 1 to 256 nodes
- Graph shows multilevel switch slowdown in per core/state basis
- Exponential growth of rawtime

$$\text{Normalized performance} = \frac{\text{gates} \times 2^Q}{\text{Rawtime} \times \text{cores}}$$



Normalized performance

- Number of basis states each core can process in a second

$$States/(s \text{ core}) = \frac{gates \times 2^Q}{Rawtime \times cores}$$

- In an ideal computer that should be a constant
- In reality it will vary due to the network and memory access
- Useful to compare cores
- Raw time useful to compare whole systems

One node SKX Xeon 8160 and different core numbers, the metric is constant.

