Robust Cache-Aware Quantum Processor Layout

Travis LeCompte, Fang Qi, and Lu Peng
Division of Electrical and Computer Engineering
Louisiana State University
Baton Rouge, LA, 70803, USA
{tlecom3, fqi1, lpeng}@lsu.edu

***Abstract*—Quantum computation has taken over as one of the largest current research areas in computer architecture and information theory. With the potential to make a large number of factorization-based encryption methods obsolete, companies and governments around the globe are racing to build the first large-scale quantum computer. Currently, most quantum computers are noisy intermediate-scale quantum (NISQ), using a relatively small collection of unreliable qubits. While error correction methods exist, they require a large number of ancilla qubits to protect the data qubits which is not practical for use on current NISQ machines. However, following the Dowling-Neven Law, available qubits on a superconducting chip are growing at an exponential rate similar to Moore's Law. Looking toward larger scale quantum machines, we examine a method to increase usable qubit density of quantum machines implementing error correction by using quantum caches that utilize simpler error correction codes. Alternatively, this also allows for the design of reliable systems while meeting the performance and qubit requirements for quantum algorithms. We modify the Qiskit quantum simulation library to work with caches and investigate the effects of region size and topology on the swap characteristics of algorithm execution. We also present our results and discuss recommended topologies for each algorithm. Lastly, we present mix scale-out simulations to examine the impact of cache on future large-scale machines. The default central cache topology gains a maximum performance increase of 2.15 times compared to the worst topology, which creates a robust cache-aware quantum processor layout.**

***Index Terms*—Quantum Processor, Cache Layout, Robustness.**

I. Introduction

Interest in quantum computing and information technology has grown considerably in recent decades. Various companies including IBM, Google, and Microsoft, along with governments around the world, have been working to advance quantum technology. Currently, these quantum chips largely act as specialized hardware for efficiently executing quantum algorithms and physical simulations, while a general quantum computer is far in the distance. However, this does not diminish the importance of quantum technologies which already see use in quantum random number generators and magnetic imaging devices. Google and NASA recently claimed highly anticipated quantum supremacy [1], the realization of a chip that can do in minutes what would take classical computers thousands of years.

The state-of-the-art quantum processors are classified as noisy intermediate-scale quantum (NISQ) machines due to their relatively small number of error-prone qubits. While these NISQ devices are beneficial research tools, their practical applications are severely limited by their scale and unreliability. Traditional error correction methods used in classical computers via replication cannot apply to qubits due to the quantum no-cloning theorem, which does not allow for copying unknown qubit states. Quantum error correction schemes do exist, but carry an overhead that is not practical on NISQ machines. This directly connects both the scale and reliability problems – at an arbitrarily large scale, we can protect qubits to ensure reliable computation. However, just as Moore's Law is reaching its end, one can assume that an infinitely large quantum computer as predicted by the Dowling-Neven Law [2] is equally unrealistic. As such, there will always be a restriction on the number of high-fidelity qubits we can achieve in a quantum computer, and we would like to waste as few as possible on error correction.

We investigate the application of quantum caches to modern superconducting quantum computers in order to achieve functional error correction for robust quantum computing at increasingly smaller scales. Unlike classical caches that reduce execution time by reducing memory latency, these caches reduce the error correction overhead for protecting cache qubits by acting as a dedicated memory. By reducing the number of operations that take use these cache qubits, error probabilities are decreased and performance requirements are lowered, thus allowing for lighter error correction schemes. This results in more usable qubits as fewer are allocated for error correction, which in turn allows smaller scale devices to be robust to error while continuing to meet qubit and performance requirements. However, by restricting operational regions, we incur a cost as operations must avoid these cache regions. Through our experimentation we aim to minimize this overhead while maintaining the benefits of the caches.

To simulate these caches, we modify multiple parts of IBM's Qiskit quantum simulator [3]. Specifically, we target the virtual to physical qubit layout, the physical qubit coupling map, and the swap passes during compilation. This allows us to generate different cache layouts within the physical qubit map and ensure that the simulator can work with these caches to complete execution. For testing, we examine four cache layouts and two swap algorithms on five quantum algorithms and measure the overhead incurred. From our observations we provide a policy that performs well for all algorithms.

To ensure the validity of our results on large-scale systems that will implement error correction methods, we extend our

simulations to larger mesh networks. Qiskit is unable to simulate large meshes due to memory constraints during the computation of the quantum algorithm. As we are focused on minimizing the movement of qubits during an arbitrary program, we can simulate large scale algorithms by removing the computation component. This enables us to bypass memory limitations at the cost of algorithm execution. Therefore, combined with our small-scale observations, we can display both correct execution and scalability.

The contributions of this work can be listed as follows:

- Our design is the first work, to the best of our knowledge, to apply memory architecture to superconducting quantum technology.
- Our design explores the design space of possible quantum computer cache layout using five advanced quantum algorithms.
- Our design achieves a possible maximum performance increase at 2.15 times compare to the worst cases while keeping a robust system.
- Our design is the first work to explore cache architecture design space at large-scale quantum chip level using mixed scale-out algorithm.

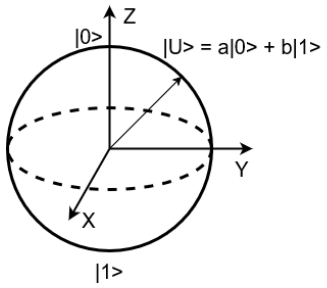

Fig. 1. Visualization of the Bloch Sphere

II. Background

A. Quantum Computation

Quantum information can be represented in multiple ways. One such representation for qubits, shown in Fig. 1, is the Bloch sphere [4]. The state of a qubit is represented by a vector in the sphere. The two states $|0\rangle$ and $|1\rangle$ form a basis set for the vector space of all possible states. If we restrict possible values to the basis set, we effectively have a classical bit. However, qubits can also take the state of any linear combination of the basis states with coefficients from the complex numbers, commonly known as a superposition of states. Operations analogous to classical gates can be represented as rotations of the state vector around the Bloch sphere. All operations can be reduced to a set of universal single- and double-qubit gates including the Pauli operators, the Hadamard gate, and the controlled-not (CNOT, also called the CX) gate.

Quantum algorithms perform computation by manipulating these states individually or in pairs. It is common to transform the data into a superposition with Hadamard gates, perform a series of operations, then transforming back using another series of Hadamard gates. This can be seen in both Fig. 3 and 8, where the square H gates are single-qubit Hadamards, and the circular gates are CNOTs. Each horizontal line represents a single qubit with gates applied sequentially from left to right. The final step of quantum computation is typically to measure the final states with respect to our basis set described above, returning to classical bits.

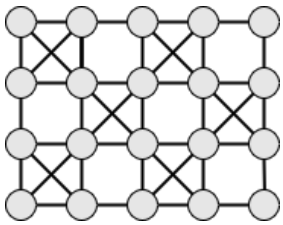

Fig. 2. Standard Mesh Network for Superconducting Quantum Computer

There are many different technologies to implement qubits and the associated gates. Ion trap computers construct qubits using suspended ions, where the basis states are the ground state and excited states of the ion [5]–[7]. Gates can be executed by applying lasers to excite the ions and measuring emitted photons. Ion trap systems have longer decay times and higher quality qubits [8] compared to some other implementations while also allowing for any-to-any entanglement between qubits. Optical quantum computers utilize photon polarizations as qubits, commonly with a basis of the horizontal and vertical polarizations [9]. Polarizers and beam splitters act as gates in these systems, using relatively well-known optical hardware. More exotic methods also exist, such as nitrogen-vacancy centers in diamonds that provide high quality qubits [10]. However, most current quantum chips use superconducting technology [11].

B. Superconducting Quantum Computers

Superconducting quantum computers are built from a collection of Josephson junctions. Each junction is composed of two superconducting materials separated by a thin barrier. This allows for the Josephson effect to take place, where electrons tunnel through the barrier with a phase shift, similar to an inductor. In combination with capacitors, this becomes an LC circuit, where the energy levels in the circuit (alternatively, the flux through the loop) act as the basis states for the qubits. Gates are performed by exposing the circuit to specific microwave pulses for certain durations depending on the rotations the gate carries out. The technology overlaps with traditional semiconductor production and has proven easier to produce functional machines. Both IBM and Google have invested into superconducting quantum systems and have proven successful in creating systems upwards of 70 qubits [12].

By comparison to ion trap systems, superconducting qubits are generally lower quality and are more vulnerable to error and decoherence. Since it relies on superconducting circuits, the chip must be cooled to extremely low temperatures to maintain the flow of supercurrent. Additionally, Josephson junctions cannot be connected in an all-to-all fashion to allow

for entanglement between any two physical qubits, as is possible with ion trap machines. Instead, the junctions are connected in a mesh network as shown in Fig. 2, where the connections identify which qubits can be entangled. This introduces multiple challenges to the processor design. If an algorithm requires for two qubits to be entangled, they must be moved to be adjacent to one another. Due to the quantum no-cloning theorem, states cannot be copied and thus must instead be moved into correct positions from physical qubit to physical qubit through the mesh by swapping adjacent qubits. Therefore, reducing the number of swaps necessary during computation becomes an optimization problem [13], [14].

IBM provides the Qiskit library [3] to simulate or connect to superconducting back-end devices for quantum computation research. It enables the creation of quantum circuits to execute algorithms and provides access to various back-ends for testing. Since it simulates superconducting chips, the abstract circuits must be compiled to match the physical mesh structure of the chips. There are three components of the compilation process that are important to note – the coupling map, the layout, and the swap algorithm. The coupling map defines the shape, size, and connectivity of the mesh. The layout is an initial map for virtual to physical qubits, and the swap algorithm decides how to move qubits to their necessary positions to perform entanglement during algorithm execution. In Section 4 we discuss the necessary modifications to these stages to enable cache-aware execution.

C. Quantum Error Correction

Error correction mechanisms are critical both in classical and quantum computing to ensure correct operation. Classically, errors occur most frequently in the communication of signals between machines. These errors are caused by environmental noise affecting the communication channel and typically appear as bit-flips present in the transmitted data [24], [25]. Some errors corrupt data into unreadable formats that make it very clear an error has occurred [26], [29]. Other errors may not be easily identifiable and can propagate into future computations [27], [28], [32]–[34]. Correcting these errors during transmission usually rely on a form of error correcting code. First, transmitted data is encoded in some form of error resilient representation [30], [31]. If any errors are detected on receiving the data, they can be corrected as a part of the decoding stage.

Most classical error correction codes are built on redundancy. Data is duplicated into multiple redundant copies and transmitted as a single logical bit. Errors can then show up as a bit-flip of an individual physical bit with probability p. A simple example is the triplication of data, where a 0 is encoded into three bits as 000 and a 1 as 111. Both classical and quantum error correction codes will be referred to as the ratio of data:ancilla (qu)bits, or 1:2 in this case for a total of 3 bits per logical bit. A majority-rules policy is implemented when decoding the data in an effort to correct any potential errors. If we receive 101 from the channel, we know at least a single error has occurred, which would imply the original value was a 1. Note that it is also possible two errors occurred in both the first and last bits, which cannot be resolved using this triplication code. However, provided that we are more likely to see single errors ($p < 0.5$), we see an improvement in total error rates. To handle multiple errors, one can simply extend the code from triplication to possibly 5 or 7 bits and maintain the majority-rules policy. The probability threshold then depends on the number of errors we hope to protect against and the total number of bits in the encoding. This also demonstrates that the minimum number of redundant bits necessary to protect against errors depends on the probability of error.

One may think we can merely apply the same classical replication techniques to qubits to achieve similar error detection and correction capabilities. However, qubits have two major features that complicate the error correction process. First is that qubits are not quantified into only two states like classical bits. One of the powerful features of quantum information is that a qubit can take one of an infinite number of states represented as a superposition of the two basis states. This impacts how errors can appear within a qubit. Qubits can experience two kinds of discrete errors known as bit and phase flips. These can be represented as the X and Z Pauli operators respectively, which flip the state of the qubit around the respective axis. In addition to these flips, in the worse case, an error on a qubit can turn the state $a|0\rangle + b|1\rangle$ into a completely random state $c|0\rangle + d|1\rangle$. Thus the form of quantum errors is more complex than classical bit-flips. The second major limitation is the quantum no-cloning theorem, which states than we cannot make copies of an arbitrary quantum state. This prevents us from duplicating the state for redundancy as in the previous error correction methods, nor can we measure the state to prepare a copy without collapsing the state and destroying the superposition of basis states.

For some time, it was argued that reliable quantum computing and quantum error correction may be an intractable problem. However, Shor's code [15] proved that it was possible to correct arbitrary quantum errors using ancilla qubits using a method built upon classical redundancy methods. Shor's code follows a similar idea to the classical error correction but avoids both copying and collapsing the arbitrary state. The code works in two stages to correct both the bit and phase-flip errors that can occur. First, to address bit-flip errors, we add two ancilla qubits for redundancy and entangle them with the data qubit by using CNOT gates. Note that this does not involve any copying or extraction of data from the initial state. However, the triplication of the basis states looks and acts similar to the classical error correction method. If an error occurs, it is likely to be a single qubit bit-flip, meaning it can be resolved with a majority-rules policy as in the classical setting. To detect any present errors, we can measure the qubits with a special set of measurements dubbed the error syndromes. These syndromes provide information of where the errors occurred without destroying the states, which allows one to undo the bit-flip error. A similar method exists for correcting phase-flips.

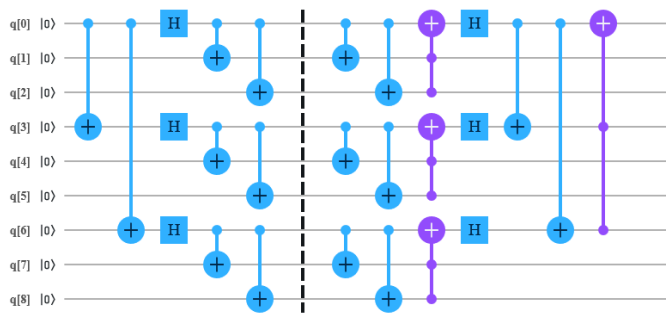

Fig. 3. Encoding and Decoding Circuit of the Shor Code

These two methods can be combined into a 1:8 error correction code that allows for the correction of arbitrary single-qubit errors. The circuit for realizing this error correction code is shown in Fig. 3. The vertical dashed line denotes the potential occurrence of an error during computation or transmission of data. Similar to the classical error correction methods, this provides a benefit in total error rates so long as the probability of error $p < 0.5$. It is possible to continue concatenating the codes to increase resilience and correct multi-qubit errors, though the number of qubits necessary quickly outgrow currently available technology.

There have since been many additional codes suggested for quantum error correction [16]–[18]. Some are modifications of Shor's code, such as the Steane code [19]. Others take a different approach, such as surface codes. These codes lay out the ancilla qubits on a surface (both 2D and toric codes are popular [20]) and take advantage of geometry to allow for error detection and correction. Logical zeros and ones are encoded by laying out patterns either in loops on the grid or in lines across or down the surface. In effect, this produces a situation where an error would have to manipulate many qubits to disrupt the logical value of the encoded qubit. Each of these codes require a considerable number of ancilla qubits which limits their applicability to current NISQ machines that operate with a smaller number of qubits.

D. Quantum Caches

In a previous work [21], Thaker et al. discuss the possibility of multiple levels of encoding at various overheads for computation, cache and memory regions in ion trap based quantum computers. While traditional cache designs provide performance increases by holding frequently used data in high-speed memory regions, the quantum cache design focuses on decreasing chip area when deploying error correction codes. By utilizing two error codes, they avoid wasting a large number of qubits for error correction on qubits that are less frequently used or less prone to error (the cache qubits). This allows for slower or less precise error correction methods in memory regions that are used less frequently, while deploying faster error correction in computation regions that must be applied after each operation. They also include code conversion circuits for transferring qubits in one encoding to another, allowing for fast transfer between cache and computation encodings.

An interesting effect of this design is that it also increases the number of usable qubits available if we are operating with a fixed number of physical qubits. Here we define usable qubits as the number of physical qubits in the system that are not allocated for error correction purposes. This is equivalent to the number of logical qubits the system can represent. For NISQ systems that operate with a limited number of qubits, this technique may enable smaller-scale systems to execute more demanding algorithms by providing a larger number of usable qubits.

III. Motivation

In the previously mentioned quantum caches, the authors implement 8:1 and 1:2 encodings in their compute and cache regions, providing a large improvement in the number of qubits necessary for error correction. The main source of this improvement is taking advantage of ion trap quantum computers' long coherence times, which can be on the order of multiple seconds and possibly even minutes in certain configurations [22]. As the previous proposed cache is designed for ion trap systems, they do not face the mesh connectivity problems that superconducting systems experience. This becomes the major design question we must address when porting this concept to superconducting technology.

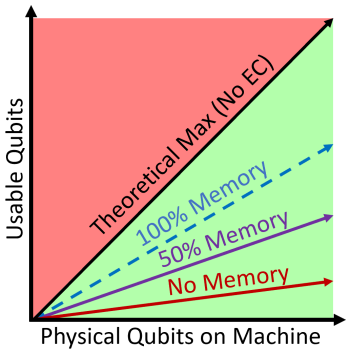

Fig. 4. Scaling of Usable Qubits with and without Cache

Superconducting technology cannot take advantage of coherence times when considering multiple error correction codes. Current superconducting coherence times are typically on the order of milliseconds. However, we can still take advantage of varying levels of encoding to save qubits where possible. Although we do not have the large coherence times, superconducting gate times, the time necessary to perform a gate operation on a qubit, are substantially shorter (order of nanoseconds) than ion trap machines (order of microseconds). This difference in gate time makes up for the difference in coherence times and results in comparable number of gate operations per coherence time. This allows us to follow a similar multi-level encoding structure. By having both "compute" and "cache" regions, we can also deploy faster but more costly

error correction in compute regions, and a more qubit-efficient encoding in the cache region. Two simple codes to choose to employ are the Shor code for compute regions (1:8) and the Steane code for cache regions (1:6). In total this is a reduction of 2 qubits per data qubit, or a $\frac{9-7}{9} = 22\%$ reduction for the cache regions.

Figure 4 shows the concept of saving qubits as we increase cache size. Here, usable qubits are defined as the sum of both computation and cache qubits (basically all non-ancilla qubits reserved for error correction). Naturally, the line $y = x$ acts as our absolute boundary. If we did not need to implement error correction at all, our device would lie along this line, though such a perfect machine does not currently exist. All other lines can be drawn following $N = \frac{pn}{c_1} + \frac{(1-p)n}{c_2}$ where N is the number of usable qubits, n is the number of physical qubits in the machine, p is the cache percentage, and c_1 and c_2 are the number of qubits necessary for the error correction codes for the cache and compute regions respectively. The dashed line represents marking every qubit as a cache qubit – this is impractical as we do not have any qubits for computation, but it provides an upper bound on our design as it would apply the less costly error correction code to the entire system. Similarly, the red line denotes having no memory, and thus having the most costly error correction code apply to all qubits. This results in the most qubits allocated for error correction. Therefore, when choosing a cache size, we fall somewhere between these two lines – shown here is an even split between cache and computation qubits. We can guide our choice of cache size by the number of qubits required for an algorithm. By maximizing the cache percentage p while maintaining enough usable qubits to execute the desired algorithms, we waste fewer qubits on error correction while meeting functional requirements.

An alternative perspective is to consider that we are enabling error correction on a system that cannot otherwise support error correction codes while meeting the performance and qubit requirements for a given algorithm. For example, if we wish to use Shor's code for an algorithm that requires n logical qubits, we would effectively need $9n$ physical qubits when adding the ancilla qubits for error correction. By comparison, using a 50% cache size with the Shor and Steane codes as discussed above, we can implement error correction with only $\frac{9n}{2} + \frac{7n}{2} = 8n$ qubits. While we could simply apply the Steane code to every qubit, there may be limitations that prevent this, such as performance requirements or differences in the reliability of individual qubits.

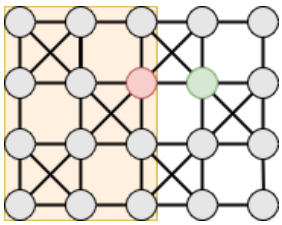

Fig. 5. Cache Forced Swaps

IV. Design

In order to simulate quantum caches on superconducting chips, we modify each of the three main parts of the Qiskit library – the coupling mesh, the initial layout, and the swap algorithms. We add a list to the mesh to keep track of the included cache nodes. This provides a base from which to enforce gate restrictions and make the execution cache-aware. The initial layout is modified to prioritize non-cache qubits to avoid unnecessary swaps. The swap algorithms must be aware of the cache qubits to enforce swaps in and out of the cache in addition to its traditional job of ensuring that qubits are adjacent for entanglement.

Beyond simulating on small scale superconducting chips around 20 qubits, we also extend Qiskit to perform large-scale quantum circuits on large-scale superconducting chips, around 100 qubits level. To perform this task, we modify the Qiskit library to allow large-scale circuit compilation and propose a scale-out algorithm to generate these circuits. We modify the simulation process to only perform swap mapping while discarding data operations such as single-qubit gates. This allows us to circumvent the library restrictions and makes simulation feasible. For the scale-out algorithm, we extract circuit characteristics directly from real quantum applications and feed them into the mixed scale-out algorithm, which aims to run on classical computers with high fidelity relative to the real large-scale circuit.

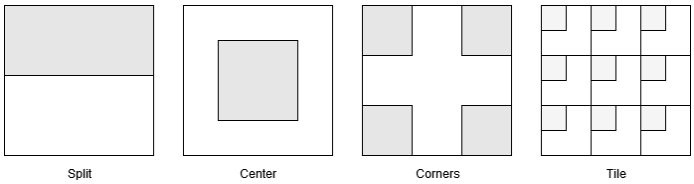

Fig. 6. Four Basic Cache Topologies

A. Coupling Mesh and Cache Topology

The first necessary step is to add the concept of a cache to the simulator. We add a list to hold all nodes that are in the cache. This is used by the layout and swap algorithms to identify which nodes should be prioritized for mapping and swapping. We additionally construct a separate mesh with cache nodes removed to allow for easy swap path identification. The swap algorithms rely heavily on shortest path algorithms, so it is beneficial to have a pre-made separate graph limited to only data qubits to prioritize non-cache swapping where beneficial. Lastly, we add functions to generate the various cache topologies used in our experiments.

In the previous work examining the quantum memory hierarchy, the system is implemented using ion trap technology [21]. Due to its arbitrary qubit entanglement capabilities, there is little distinction between data and cache qubits. However, there is a major difference when using superconducting qubits because the connectivity between qubits is limited, and we cannot simply operate on any two qubits at will. Choosing

which qubits to place in the cache thus has an impact on the swaps the algorithm must perform to complete the algorithm. As shown in Fig. 5, the two marked qubits are adjacent and therefore should be available for operation. However, with the highlighted cache placement, one of the qubits falls within the cache and thus is not valid for use. To complete the operation and continue with algorithm execution, the qubit must be moved out of the cache, creating more swaps.

Given n physical qubits there are 2^n potential cache layouts to consider. In order to avoid searching this whole space, we instead focus on four different topologies, two contiguous and two distributed, as shown in Fig. 6. The two contiguous methods, a straight split and a central cache, allow for larger contiguous cache and computation areas. The two distributed methods are the four corners of the mesh and a tiled version that spreads the cache equally throughout the mesh. The corner topology also allows for a large contiguous region of computation but divides the cache into parts, while the tile method instead opts to intersperse both cache and compute qubits. These four methods together provide a variety of options to consider for cache design based on algorithm characteristics.

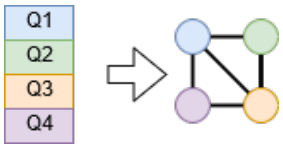

Fig. 7. Four Qubit Initial Layout

B. Layout

The initial layout of the algorithm maps the required virtual qubits to the physical qubits in the mesh, as shown in Fig. 7. Similar to the previous discussion about cache topology, a number of compiler passes directly involving the physical qubits need to be modified to account for the cache, including this initial layout. Poor initial qubit placement can result in additional swaps. It is not as critical as the swap algorithm since it is only the initial state of the system, but it can have considerable impact on shorter algorithms.

Typical layout passes examine both the mesh and the algorithm to decide on qubit placement. First, identify the most and least heavily connected qubits in the mesh. Second, look ahead through the gates that must be executed to identify the order in which qubits must be operated on. Using this information one can map the qubits that are used together to be nearby in the mesh. However, adding the concept of a cache changes this process, as an unaware layout may map virtual qubits to cache qubits and add unnecessary swaps. We modify this process to also reference the list of cache qubits in the mesh and prioritize non-cache qubits before cache qubits. This allows us to retain the benefits of the pre-existing layouts while avoiding as many unnecessary swaps due to the cache as possible.

C. Swap Algorithms

The last major part of the compilation process that we must modify are the swap algorithms. These algorithms are responsible for ensuring that qubits are adjacent to each other whenever two qubits are operated on together. As discussed previously, it is critical for the swap algorithm to be cache-aware for the algorithm to complete execution. Otherwise, the simulation would attempt to operate on qubits when it cannot due to the cache, leading to failures. In this sense it is also the responsibility of the swap algorithm to enforce these additional restrictions on the qubits' positioning within the mesh. While there are many different swap algorithms that have been studied to minimize swaps or maximize reliability, we present two different cache-aware swap algorithms and compare their properties and behaviors.

In general, the swap algorithms examine the list of operations that must be executed and insert swap operations that move the qubits to their necessary positions in the mesh. With no cache, single qubit operations do not require any swaps as they can be executed locally at any qubit position. The only operations that force qubit movement are two qubit operations such as the CNOT. Three or more qubit operations do exist, but can be unrolled into a combination of one and two qubit operations in the basis set. Upon locating a two-qubit operation, the swap algorithm checks whether they are adjacent. If not, it finds a shortest path from one qubit to the other and inserts swaps along that path. It is possible for the movement of one qubit to possibly move other qubits further from their goal locations. Some algorithms implement look-ahead mechanisms to address this problem and increase system-wide efficiency, while others implement probabilistic methods to avoid interference.

Algorithm 1 Baseline Swap Algorithm

Input: A list L of all gates and their operating qubits q, v

```
for all gates in L do
    if single qubit gate and q in cache then
        swap along shortest path to nearest available non-cache qubit
    end if
    if two qubit gate then
        if both q and v in cache then
            swap q out to nearest available non-cache qubit
            swap v to nearest available non-cache neighbor of q
        else if only one of q and v in cache then
            swap along shortest path to non-cache neighbor of v or q respectively
        else if neither q or v in cache and they are not adjacent then
            swap v to nearest available non-cache neighbor of q
        end if
    end if
end for
```

When incorporating a cache, the first modification involves single-qubit gates as they can no longer be executed on any qubit in the mesh. Instead, they are now capable of forcing movement if a single-qubit operation is set to take place on a cache qubit. By definition of the cache, operations should not act on cache qubits wherever possible to increase system reliability. As such, even single-qubit gates may require moving out of the cache to a non-cache region. For two qubit operations, as previously discussed, both qubits must be out of the cache and adjacent with each other for the operations to be successful. Both algorithms presented follow Algorithm 1, but act differently when selecting the paths to take to move qubits together. The first algorithm acts as a baseline, here referred to as the BaselineSwap (BSwap). It finds the direct shortest path between two qubits and routes them together, ensuring their final positions are not within cache qubits. This ensures correct execution of the algorithm, but allows for swaps through the cache. The second swap algorithm aims to minimize the number of swaps occurring within a cache, here referred to as the NoCache Swap (NCSwap). By utilizing the previously mentioned reduced mesh that does not contain cache qubits, the algorithm can easily find the direct shortest path using only non-cache qubits. In the case that either qubit is in the cache itself, it first moves them out to a non-cache region, then routes them together avoiding cache qubits. The only modification necessary is to use this modified mesh that does not contain the cache qubits when finding the shortest paths in lines 3, 8, 10 and 12. This algorithm does not work if the compute region is not contiguous, as the mesh then becomes disconnected, though it can be modified to simply fall back to BSwap in these circumstances.

Impact on Performance and Reliability. Implementing the cache adds data movement as we must swap qubits in and out of the cache and avoid transferring through cache regions where possible. The cache shape, size and swap algorithms can effect this performance overhead. Our design aims to minimize the number of swaps at various cache shapes and sizes, which we treat as our main performance metric in the following evaluations. Reducing the number of added swaps reduces execution time and increases reliability by reducing the number of total gate operations on the qubits, in addition to enabling the error correction codes at smaller scales.

D. Large-Scale Implementation

Due to the inherent exponential growth of quantum algorithms, using classical computers to simulate complete quantum circuit generation and computation at large scale is infeasible and would otherwise contradict quantum supremacy. However, observing the behavior of a real application on a large-scale circuit is one of the critical components for assisting large-scale quantum computer design. Therefore, it is necessary to find a feasible solution which generates a large-scale quantum circuit based on the small-scale algorithm using a classical computer with high fidelity compared to the real large-scale circuit.

By inspecting the circuit growth of real applications on the small-scale Deutsch-Jozsa algorithm, shown as Fig. 9, we observe the number of total gates, CX gates, and CX gates per qubit are growing at consistent exponential rates. After studying this growing trend, we have found an inherent growth behavior of a quantum algorithm that within one quantum algorithm, different stages follow a strict sequence order, and some stages grow at an exponential rate while others only grow linearly. A simple Quantum Fourier Transform (QFT) circuit, shown as Fig. 8, will be used to illustrate this behavior, with circuit implementation from [23]. As Fig. 8 (a) shows, the QFT algorithm can be separated into sequences of stages based on its functionality, which is true for many algorithms. The stage 1 boxed in blue performs on every qubit a sequence of one Hadamard gate followed by a series of UROT gates applied to all higher indexed qubits. Similar to the CX gate, the UROT gate is a two-qubit controlled rotation gate that requires target and control qubits to be adjacent in the mesh. Stage two and three are swap and measurement stages, which request three CX gates and one measurement gate for each operation respectively. Based on the behavior of each stage shown as the table of Fig. 8 (b), it is clear that stage 1 has an exponential scaling n^2, but stage 2 and stage 3 will have a linear scaling with $\frac{n}{2}$ and n respectively.

Therefore, as shown in Fig. 8, a large-scale implementation of QFT would follow the same sequence of execution stages as small-scale while using more qubits. The number of gates necessary for some stages may scale linearly with the number of qubits, such as stage 2, while others may scale exponentially. This general trend is proved in Fig. 9. This exponential growth quickly becomes impossible to compile and simulate on a classical computer, preventing direct scaling. However, it is still possible to approach similar qubit movement behavior using the Mixed Scale-out Algorithm presented next.

1) Mixed Scale-out Algorithm: A mixed scale-out algorithm will enlarge each stage of a real small-scale algorithm with a constant rate of n copies of gates from the target stage using a random mixed fashion. Following this rule, the circuit produced by a mixed scale-out algorithm will have the same stage sequence as the real large-scale quantum algorithm. For the gate number difference, the resulting circuit will only be different at stages that require exponential scaling, which are projected to a constant rate n. Since this paper focuses only on the swap mapping and swap count difference between different cache layouts, only 2-qubit gates will be extracted and used. Using the QFT circuit as an example, assume that the small-scale circuit has 6 qubits and that the target large-scale circuit has 120 qubits. A mixed scale-out algorithm will make a copy of the small-scale algorithm 20 times, and aggregate all the gates into corresponding stages. For stage 2 and 3, this enlarging process will be very close to the real large application. Stage 1 will have $6^2 * 20 = 1,240$ swap paths generated by the mixed scale-out algorithm, while the real large-scale circuit will have $120^2 = 14,400$ swap paths. The mixed scale-out algorithm successfully reduces the complexity of the swap path generation from $O(n^2)$ to $O(n)$

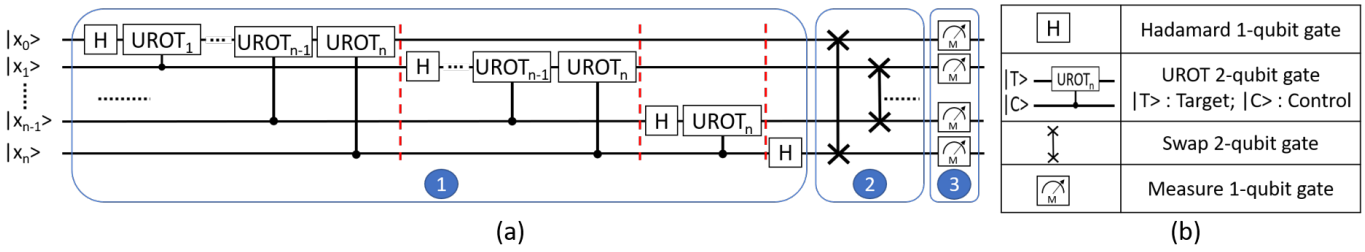

Fig. 8. n-qubit Quantum Fourier Transform(QFT) circuit. (a) QFT circuit separated by stage. (b) Table of gate diagram with the number of qubits involved.

with all the created swap paths belonging to the real large-scale circuit. Therefore the mixed scale-out algorithm perfectly suits the purpose of exploring the path behavior in the large-scale quantum chip with a balance between feasibility and fidelity.

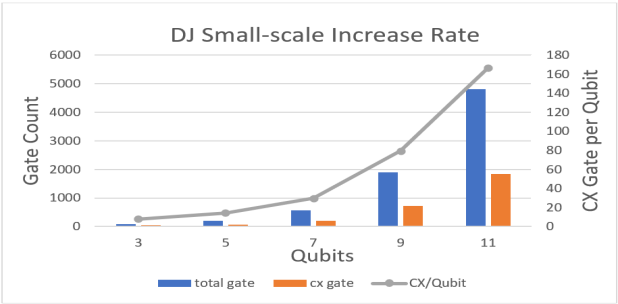

Fig. 9. DJ gate growing rate

Algorithm 2 Mixed Scale-out Algorithm

Input: A list of Quantum Gates C, scale-out ration n
Output: A list of Quantum Gates C'
  for each gate $0 \leq g \leq |C|$ **do**
    if $gate != measurement$ && $gate! = barrier$ **then**
      add the gate in C'
    end if
  end for
  for each gate $0 \leq g \leq |C|$ **do**
    Duplicate each gate n times
    Rename every Quantum Register of each new gates
    Save the new gates into C'
  end for
  enable random qubit allocation

2) Enabling Large-scale Qiskit Library for Mixed scale-out Algorithm: The large-scale Qiskit library inherits features of the three implementations in the preceding subsections. Compared with the original Qiskit library, the large-scale Qiskit library only performs the circuit construction and compiling processes while discarding the simulation part for computation. Meanwhile, the memory defined limitation of the largest number of physical qubits has been removed to allow mapping of the virtual qubits to physical qubits with any given size. This isolated process releases the potential of the library to be able to measure the swap count with any given input size. The proposed mixed scale-out algorithm shown as Algorithm 2, aims to generate circuits on a large scale and keep the balance between feasibility and fidelity. The algorithm extracts the circuit from a small algorithm and feeds it into the filters that filter out the barriers and measurement operations that influence combination with another circuit. The duplication procedure will rename the quantum registers to avoid mismatching. For executing each gate within the target stage from different small circuits, the algorithm adopts the round-robin policy to execute each gate of the circuits. After the generation of the circuit, the algorithm will enable the random qubits allocation procedure to avoid the qubits from the same copy to aggregate.

V. Results

For our experimentation, we test five quantum algorithms: Shor's factorization algorithm [35], Grover's search algorithm [36], Simon's algorithm [37], the Deutsch-Josza (DJ) algorithm [38], and the Bernstein-Vazirani (BV) algorithm [39]. These algorithms cover a variety of quantum computation tasks. BV, DJ and Simon's algorithm are three of the first quantum algorithms used to demonstrate the benefit of quantum computers over classical computers. Grover's search algorithm searches a set of quantum data to identify matching queries. Shor's algorithm, likely the most well known of the set, is a polynomial time factoring algorithm whose security concerns interest governments worldwide. Each of these algorithms are tested on all four cache topologies at various cache sizes. The number of swaps are recorded to show the swaps incurred by each cache topology. The overhead in the number of swaps can be used to approximate the overhead in computation time following the function $f(n_s) = 3 * n_s * t_{cnot}$, where n_s is the swap overhead and t_{cnot} is the average time necessary to execute a single CNOT gate. Results for both the BSwap and NCSwap algorithms are shown to provide comparisons under all settings.

A. Small Scale

For these small scale simulations, we implement the cache and swap algorithms on sizes that can be executed with current technology without error correction enabled. We cannot actually implement error correction for testing purposes, as it would require more qubits than can be feasibly simulated. Instead, we are working directly with the qubits and assuming that error correction would be implemented at a larger scale.

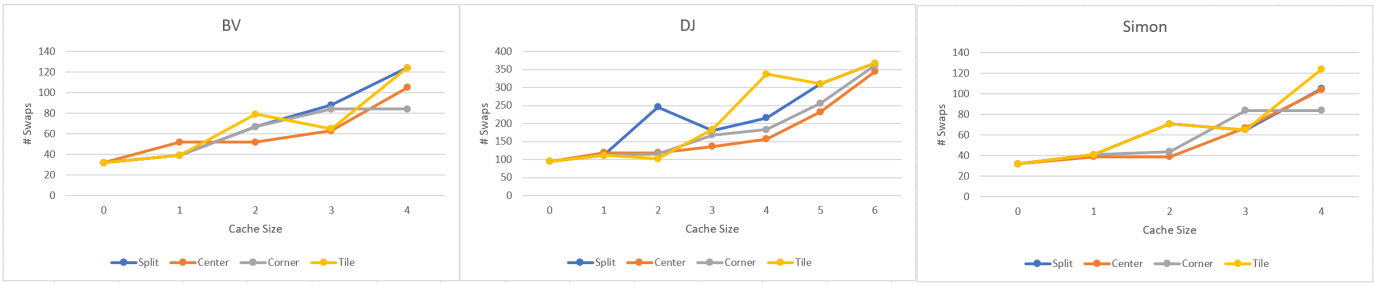

Fig. 10. Baseline Swap Results, Small Algorithms

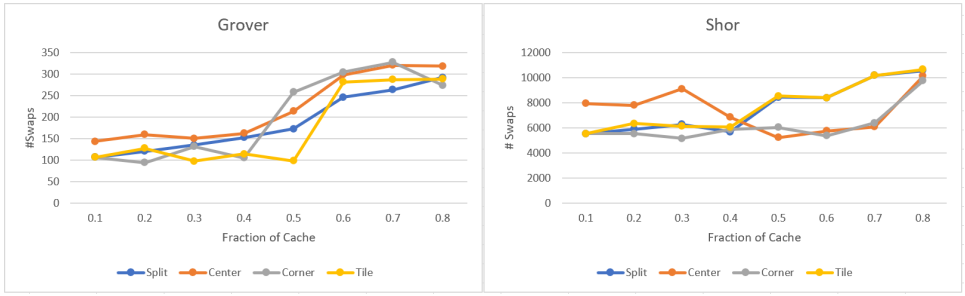

Fig. 11. Baseline Swap Results, Large Algorithms

Baseline Swap Results. The results for the BSwap algorithm are split into two figures, Fig. 10 and 11. The larger algorithms (Shor and Grover) are separated from the smaller algorithms (BV, DJ, Simon) because we use two separate methods for manipulating the cache size. For the larger algorithms, taking a percentage of the total mesh size as cache works with no problem, as there are enough qubits to select from. However, the smaller algorithms use a smaller mesh, resulting in rounding issues when using cache percentages. To provide more clear results about the effects of increasing cache size, we instead directly increase the number of cache qubits rather than relying on a percentage.

The smaller algorithms exhibit relatively similar behavior for all of the cache topologies. At most cache sizes, either the corner or center topology performs best depending on the algorithm, though the tile and split topologies do not perform much worse. As expected, the number of swaps trends upwards as cache size increases, as more qubits must be moved out of the cache.

The larger algorithms also show a general trend upwards in swaps as cache size increases, though Shor's algorithm shows somewhat more complex behavior. Grover's algorithm stands out as the only one that benefits mostly from the tile topology, and the split topology at higher cache sizes. By comparison, Shor's algorithm is has consistently low number of swaps with the central topology at all cache sizes except for the largest cache size. We will discuss why we believe the algorithms display these behaviors in the following discussion section.

Best and Worst-Case Overhead. In order to underscore the importance of choosing the correct cache topology, we identify the best and worst cache topology at each cache size for each benchmark and calculate the overhead difference between the best and worst topologies. Based on the previous observations in Figs. 10 and 11, we choose to treat the central cache as our default choice as it consistently performs best on the smaller algorithms, and the larger cache sizes for Shor's algorithm. Fig. 13 displays the minimum, maximum, and mean overhead figures for each benchmark. As shown, the difference between minimum and maximum ratios can be very large, with the greatest difference being roughly 147% for Grover's algorithm. When examining only the default central topology, we similarly find the greatest difference of 115%, or a 2.15x improvement. There is also typically a sizable difference between the average and minimum overhead, ranging from roughly 9-30% overhead over the best choice of topology. The large differences between the maximum and mean topologies indicate the impact of cache topology on performance. If one were to blindly choose a cache topology for a given algorithm and cache size, they could suffer these large increases in the number of swaps and therefore execution time. As such, our recommendation is to profile a given test program to identify which cache topology is optimal, but the central cache is a consistent choice for all of our tested algorithms.

No-Cache Swap Results. In addition to the previous results for the baseline swap algorithm, we also include results for the second NCSswap algorithm. This algorithm was created to minimize the number of cache-influenced swaps (a swap where one or both of the qubits were in the cache) during execution. The swap results are shown in Fig. 12. For brevity, only DJ is shown from the smaller class of algorithms, along with Shor's and Grover's algorithm. We chose these three algorithms as our sample because DJ, BV and Simon's algorithm show fairly similar behavior and can be represented

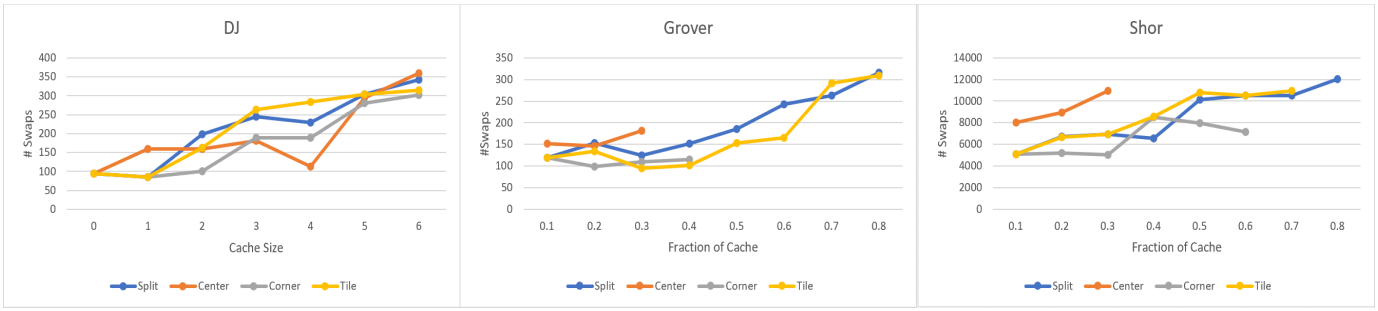

Fig. 12. Nocache Swap Results

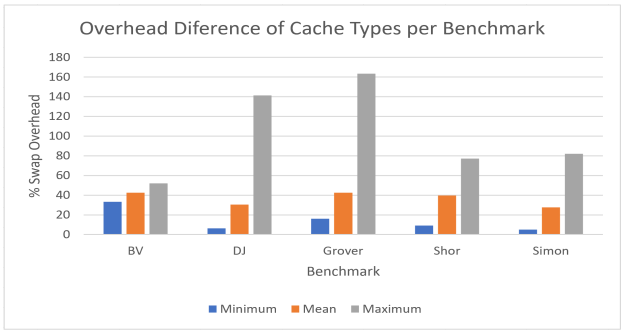

Fig. 13. Best, Mean and Worst case performance between topologies of each benchmark at all cache sizes

by one representative.

As seen with BSwap, for DJ the center and corner topologies typically perform better. However, Shor and Grover show very different behavior. The most prevalent observation is that not all cache topologies can be run at higher cache sizes, namely center and corner, which had performed best with the baseline swap algorithm. The cause of this is the requirement that the algorithm must not swap through the cache, except when one or both qubits originate in the cache. If distinct compute regions are isolated from one another by a cache region, qubits cannot move across the boundary and the computation cannot finish. This explains why the center topology is the first to fail, as it grows to divide the mesh in half. It is possible in this case to fall back on the baseline swap algorithm, but it is interesting to observe at which point the topologies begin to fail. Only the split topology is capable of completing both algorithms, which is sensible as it results in a large contiguous compute region. The tile configuration also performs well, and for the Grover algorithm also has the minimum number of swaps.

Cache-Influenced Swaps. To provide more insight into the impact of the no-cache swap algorithm, we measure how many swaps occur where at least one of the two qubits are present in the cache. We then calculate the difference between the percentage of these swaps for the baseline and no-cache swap algorithms. We expect that no-cache swap will display a substantial reduction in the frequency of these cache involved swaps as we actively avoid swapping through the cache where possible. We again calculate the geometric mean over the various cache sizes and topologies to present the total effect of the no-cache swap algorithm regardless of chosen topology or cache size, though it is worth noting that both algorithms have similar performance at the largest cache sizes as it becomes impossible to avoid swapping through the cache.

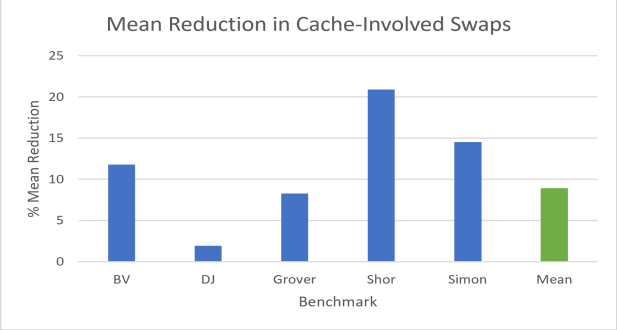

Fig. 14. Reduced Percentage of Cache-Involved Swaps

The results of these measurements for each benchmark are shown in Fig. 14. As expected, the no-cache swap algorithm reduces the number of cache involved swaps in every benchmark regardless of cache size or topology. Shor's algorithm shows the greatest improvement at nearly 21%, with a minimal improvement of roughly 2% on DJ. Across all five benchmarks, the no-cache swap algorithm provides a mean 9.85% reduction in the number of cache-involved swaps. Note that this reduction does come at an overhead in total number of swaps, as can be seen by comparing Figs. 10, 11 and 12. However, reducing the number of swap operations involving the cache further reduces the number of error correction operations that would be necessary in the cache.

B. Large Scale

As Fig. 15 showed, the results come from feeding DJ benchmark into the large-scale Qiskit with baseline swap using 96 physical qubits. After comparing with its smaller version, the overall behavior of the different topologies is similar. The rankings of the layouts at large-scale are stable that centers perform best, and split as the worst. One of the reasons that split takes the most swaps to execute might be that split has the longest path of the max possible distance between one

pair of cache and non-cache qubits, which becomes even more pronounced as the mesh increases in size. This demonstrates that the policy will be applicable for scale-out large quantum algorithm. As shown in the figures, for different layouts at a different size, a wise policy can reduce the number of swap operations by one order of magnitude for switching between the optimal layout in different cases. The large scale results also support the assumption that with a very aggressive memory-dense design as 80%, the policy can keep the extra swap overhead within 2 times range and achieve a 3 times reduction on quantum chip size.

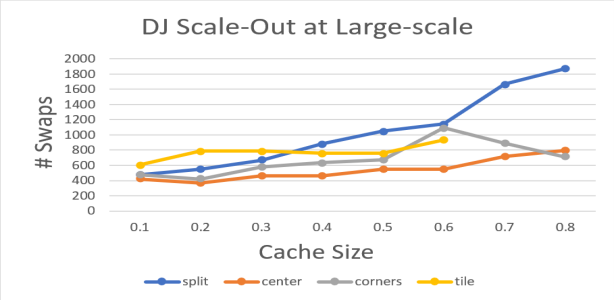

Fig. 15. Large-Scale Results of the DJ Benchmark

C. Discussion

In order to provide insight into the behavior of the algorithms we tested, we present here a small discussion on their properties and how they influence our observations. First, each of the smaller algorithms are relatively similar. They begin with a set of Hadamard gates to create superpositions of the qubits, perform a sequence of operations that depend on the given oracle the circuit is made to execute, then end with another set of Hadamard gates and measurements to extract the results. The cache mostly impacts multi-qubit gates as qubits must be moved to adjacent positions. These three algorithms benefit from larger contiguous compute regions, though not extremely as they do not implement many multi-qubit gates depending on the oracle.

By comparison, both Shor's and Grover's algorithm are more complex and rely on a larger number of multi-qubit gates. Shor's algorithm in particular shows a considerable difference between cache topologies at large cache sizes likely due to the large number of multi-qubit gates. Center and corner cache topologies provide contiguous compute regions without having to move as across the entire graph as often to meet adjacency requirements. Grover's algorithm stands apart from all of the others, actually benefiting most commonly from the tile topology. This is likely due to the implementation of the input oracle, which happens to execute in a way that benefits from the tile topology.

VI. Conclusion

In order to execute important quantum algorithms, it is necessary to increase the number of available qubits in a quantum computer. Quantum caches are one such method by reducing the number of ancilla qubits necessary for implementing quantum error correction codes. We have extended this concept from ion trap computers to superconducting meshes and modified the Qiskit quantum simulator to accommodate quantum caches. With these modifications, we have examined various cache sizes and topologies on five different quantum algorithms. Our observations show that central caches typically minimize the number of swaps added by the cache during algorithm execution, but it is optimal to profile each individual algorithm. We proposed an alternative cache-aware swap algorithm that reduces the cache disturbance caused by swapping qubits, further reducing cache operations and increasing reliability. In combination, these methods will increase the number of usable qubits on systems that implement error correction.

VII. Acknowledgement

We appreciate the invaluable comments from the shepherd Marko Vukolic and anonymous reviewers who help us finalize the paper. This work is supported in part by NSF Grants 1422408, 1527318, 1946626, and 2020446. Travis LeCompte is supported by a Louisiana Board of Regent Fellowship.

References

[1] Arute, Frank, et al. "Quantum supremacy using a programmable superconducting processor." Nature 574.7779 (2019): 505-510.
[2] Dowling, Jonathan P. "On The Dowling-'Neven' Law." Quantum Pundit. 11 July 2019. http://quantumpundit.blogspot.com/2019/07/on-dowling-neven-law.html
[3] Cross, Andrew. "The IBM Q experience and QISKit open-source quantum computing software", APS Meeting Abstracts, 2018.
[4] Goyal, Sandeep K., et al. "Geometry of the generalized Bloch sphere for qutrits." Journal of Physics A: Mathematical and Theoretical 49.16 (2016): 165203.
[5] Kielpinski, David, Chris Monroe, and David J. Wineland. "Architecture for a large-scale ion-trap quantum computer." Nature 417.6890 (2002): 709.
[6] Monroe, Christopher, and Jungsang Kim. "Scaling the ion trap quantum processor." Science 339.6124 (2013): 1164-1169.
[7] Kreger-Stickles, Lucas, and Mark Oskin. "Microcoded architectures for ion-tap quantum computers." 2008 International Symposium on Computer Architecture. IEEE, 2008.
[8] Linke, Norbert M., et al. "Experimental comparison of two quantum computing architectures." Proceedings of the National Academy of Sciences 114.13 (2017): 3305-3310.
[9] Li, Ying, et al. "Resource costs for fault-tolerant linear optical quantum computing." Physical Review X 5.4 (2015): 041007.
[10] Marcos, D., et al. "Coupling nitrogen-vacancy centers in diamond to superconducting flux qubits." Physical review letters 105.21 (2010): 210501.
[11] Fu, Xiang, et al. "An experimental microarchitecture for a superconducting quantum processor." Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture. ACM, 2017.
[12] Kelly, Julian. "A preview of Bristlecone, Google's Quantum Processor." Google AI Blog. 5 March 2018. https://ai.googleblog.com/2018/03/a-preview-of-bristlecone-googles-new.html
[13] Tannu, Swamit S., and Moinuddin K. Qureshi. "Not all qubits are created equal: A case for variability-aware policies for nisq-era quantum computers." arXiv preprint arXiv:1805.10224 (2018).
[14] Lye, Aaron, Robert Wille, and Rolf Drechsler. "Determining the minimal number of swap gates for multi-dimensional nearest neighbor quantum circuits." The 20th Asia and South Pacific Design Automation Conference. IEEE, 2015.
[15] Shor, Peter W. "Scheme for reducing decoherence in quantum computer memory." Physical review A 52.4 (1995): R2493.
[16] Michael, Marios H., et al. "New class of quantum error-correcting codes for a bosonic mode." Physical Review X 6.3 (2016): 031006.

[17] Guenda, Kenza, Somphong Jitman, and T. Aaron Gulliver. "Constructions of good entanglement-assisted quantum error correcting codes." Designs, Codes and Cryptography 86.1 (2018): 121-136.

[18] Chao, Rui, and Ben W. Reichardt. "Quantum error correction with only two extra qubits." Physical review letters 121.5 (2018): 050502.

[19] Steane, Andrew. "Multiple-particle interference and quantum error correction." Proceedings of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences 452.1954 (1996): 2551-2577.

[20] Barends, R., Kelly, J., Megrant, A. et al. Superconducting quantum circuits at the surface code threshold for fault tolerance. Nature 508, 500–503 (2014). https://doi.org/10.1038/nature13171

[21] Thaker, Darshan D., et al. "Quantum memory hierarchies: Efficient designs to match available parallelism in quantum computing." ACM SIGARCH Computer Architecture News 34.2 (2006): 378-390.

[22] Wang, Y., Um, M., Zhang, J. et al. Single-qubit quantum memory exceeding ten-minute coherence time. Nature Photon 11, 646–650 (2017). https://doi.org/10.1038/s41566-017-0007-1

[23] Quantum Fourier Transform. 27 July 2020, qiskit.org/textbook/ch-algorithms/quantum-fourier-transform.html

[24] L. Duan, et al. "Versatile Prediction and Fast Estimation of Architectural Vulnerability Factor from Processor Performance Metrics," In Proceedings of the 15th IEEE International Symposium on High-Performance Computer Architecture (HPCA-15), Raleigh, NC, Feb. 2009.

[25] L. Duan, et al. "Universal Rules Guided Design Parameter Selection for Soft Error Resilient Processors," In Proceedings of The 2011 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), Austin, TX, Apr. 2011.

[26] T. LeCompte, et al. "Soft Error Resilience of Big Data Kernels through Algorithmic Approaches," Springer Journal of Supercomputing, Vol. 73, pp. 4739–4772, Nov. 2017.

[27] L. Duan, et al. "Comprehensive and Efficient Design Parameter Selection for Soft Error Resilient Processors via Universal Rules," In IEEE Transactions on Computers, Volume 63, Issue 9, pages 2201 – 2214, Sep. 2014.

[28] L. Duan,et al. "Predicting Architectural Vulnerability on Multi-Threaded Processors under Resource Contention and Sharing," In IEEE Transactions on Dependable and Secure Computing, Vol. 10(2), pages 114-127, Mar.-Apr. 2013.

[29] Y. Zhang, et al. "Design Configuration Selection for Hard-error Reliable Processors via Statistical Rules", In Journal of Microprocessors and Microsystems, Volume 38, Issue 1, Feb. 2014, pages 22–30.

[30] S. Lim, T. Coy, Z. Lu, B. Ren, and X. Zhang, "NVGRAPH: Enforcing Crash Consistency of Evolving Network Analytics in NVMM Systems", IEEE Transactions on Parallel and Distributed Systems (TPDS), 2020.

[31] S. Lim, Z. Lu, B. Ren, X. Zhang, "Enforcing Crash Consistency of Evolving Network Analytics in Non-Volatile Main Memory Systems", In Proceedings of the 28th International Conference on Parallel Architectures and Compilation Techniques (PACT'19), Seattle, WA, September 2019.

[32] B. Li, et al. "Efficient Microarchitectural Vulnerabilities Prediction Using Boosted Regression Trees and Patient Rule Inductions," IEEE Transactions on Computers, Vol 59(5), pp. 593-607, May 2010.

[33] S. Chen, et al. "Soft Error Resilience in Big Data Kernels through Modular Analysis," in Springer Journal of Supercomputing, Vol. 72, Issue 4, Apr. 2016.

[34] S. Chen, et al. "A Framework For Evaluating Comprehensive Fault Tolerance Mechanisms In Numerical Programs," In Springer Journal of Supercomputing. Aug. 2015.

[35] Shor, Peter W. "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer." SIAM review 41.2 (1999): 303-332.

[36] Grover, Lov K. "A fast quantum mechanical algorithm for database search." Proceedings of the twenty-eighth annual ACM symposium on theory of computing. 1996.

[37] Simon, Daniel R. "On the power of quantum computation." SIAM journal on computing 26.5 (1997): 1474-1483.

[38] Deutsch, David, and Richard Jozsa. "Rapid solution of problems by quantum computation." Proceedings of the Royal Society of London. Series A: Mathematical and Physical Sciences 439.1907 (1992): 553-558.

[39] Bernstein, Ethan, and Umesh Vazirani. "Quantum complexity theory." SIAM Journal on computing 26.5 (1997): 1411-1473.