Full-State Quantum Circuit Simulation by Using Data Compression

Xin-Chuan Wu
Department of Computer Science
University of Chicago
Chicago, Illinois
xinchuan@uchicago.edu

Sheng Di*
Mathematics and Computer Science Division
Argonne National Laboratory
Lemont, Illinois
sdi1@anl.gov

Emma Maitreyee Dasgupta
Department of Computer Science
University of Chicago
Chicago, Illinois
edasgupta@uchicago.edu

Franck Cappello
Mathematics and Computer Science Division
Argonne National Laboratory
Lemont, Illinois
cappello@mcs.anl.gov

Hal Finkel
Argonne Leadership Computing Facility
Argonne National Laboratory
Lemont, Illinois
hfinkel@anl.gov

Yuri Alexeev
Computational Science Division
Argonne National Laboratory
Lemont, Illinois
yuri@anl.gov

Frederic T. Chong
Department of Computer Science
University of Chicago
Chicago, Illinois
chong@cs.uchicago.edu

ABSTRACT

Quantum circuit simulations are critical for evaluating quantum algorithms and machines. However, the number of state amplitudes required for full simulation increases exponentially with the number of qubits. In this study, we leverage data compression to reduce memory requirements, trading computation time and fidelity for memory space. Specifically, we develop a hybrid solution by combining the lossless compression and our tailored lossy compression method with adaptive error bounds at each timestep of the simulation. Our approach optimizes for compression speed and makes sure that errors due to lossy compression are uncorrelated, an important property for comparing simulation output with physical machines. Experiments show that our approach reduces the memory requirement of simulating the 61-qubit Grover's search algorithm from 32 exabytes to 768 terabytes of memory on Argonne's Theta supercomputer using 4,096 nodes. The results suggest that our techniques can increase the simulation size by 2∼16 qubits for general quantum circuits.

*Corresponding author: Sheng Di, Mathematics and Computer Science Division, Argonne National Laboratory, Lemont, IL 60439.

SC '19, November 17–22, 2019, Denver, CO, USA

ACM ISBN 978-1-4503-6229-0/19/11…$15.00
https://doi.org/10.1145/3295500.3356155

CCS CONCEPTS

• **Computer systems organization → Quantum computing**.

ACM Reference Format:
Xin-Chuan Wu, Sheng Di, Emma Maitreyee Dasgupta, Franck Cappello, Hal Finkel, Yuri Alexeev, and Frederic T. Chong. 2019. Full-State Quantum Circuit Simulation by Using Data Compression. In *The International Conference for High Performance Computing, Networking, Storage, and Analysis (SC '19), November 17–22, 2019, Denver, CO, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3295500.3356155

1 INTRODUCTION

Classical simulation of quantum circuits is crucial for better understanding the operations and behaviors of quantum computation. Such simulations allow researchers and developers to evaluate the complexity of new quantum algorithms and validate quantum devices. The path toward building Noisy Intermediate-Scale Quantum (NISQ) [54] machines such as IBM's 50-qubit quantum computer and Google's 72-qubit quantum computer [41] will require intermediate-scale quantum circuit simulators to calibrate and verify the hardware.

Unfortunately, today's practical full-simulation limit is 47 qubits (Table 1) [61]. The reason is that the number of quantum state amplitudes required for the full simulation increases exponentially with the number of qubits, making physical memory the limiting factor. Given n quantum bits (qubits), we need 2^n amplitudes to describe the quantum system [51]. In order to describe the amplitudes precisely, double-precision complex numbers are used to represent the state of the quantum systems. As a result, the size of the quantum state in the simulation is 2^{n+4} bytes. Although 49-qubit simulations will become possible in the near future with the arrival

Table 1: Examples of supercomputers, their total memory capacity, and the maximum number of qubits they can simulate for arbitrary circuits.

System	Memory (PB)	Max Qubits
Summit	2.8	47
Sierra	1.38	46
Sunway TaihuLight	1.31	46
Theta	0.8	45

of exascale supercomputers [26], a gap still remains between the size of classical simulation and the size of NISQ machines.

Several simulation techniques related to Feynman paths [6] and tensor network contractions [10, 49, 53] have been proposed to trade time for space complexity. Different approaches have different benefits and disadvantages. For Feynman paths method, both time and space complexity grow exponentially with the circuit depth, and thus this technique can simulate only shallow circuits. For tensor network simulation methods, since the time complexity grows exponentially with the underlying graph treewidth, these simulation techniques can only simulate the circuits with low treewidth. Some approaches calculate only a single amplitude or a partial state vector in order to be less restrictive on computational resources [10, 15, 16, 53]. As for quantum software development, several types of quantum applications require intermediate measurement [4, 5, 12, 29]. In addition, recent studies focus on quantum software debugging by inserting assertions in the middle of quantum programs [34]. Tensor network simulation techniques do not effectively support intermediate measurement and full-state assertion checking for software debugging. At the same time, NISQ machines are evolving toward supporting deeper circuits with error mitigation techniques [11, 25, 40, 56], which may make full-state simulations of high-depth circuits more important. When trying to verify quantum hardware and software by using a full-state simulator, every qubit counts in maximum simulation size. Every qubit closer to physical machine sizes means 2X more state space that can be evaluated, or, conversely, 2X smaller gap in state space between the simulation and the physical hardware.

To simulate general circuits with higher qubit count and depth, we propose quantum circuit simulation techniques that can reduce the memory requirement of the full simulation by compressing quantum state amplitudes at runtime. Specifically, we apply data compression techniques to the quantum state vector during the simulation. Since we aim to simulate intermediate-scale general quantum circuits, we have to achieve a data compression ratio as high as possible, because the compression ratio is the key to increasing the number of qubits in the simulation. Our approach uses lossless compression, lossy compression, and adaptive error bounds to reduce the memory requirement of the simulation. In general, lossy compression algorithms lead to significantly higher compression ratios than do lossless compressors, while introducing errors to a certain extent. To minimize the error propagation and guarantee high-fidelity simulation results, we utilize both Zstandard lossless compressor [19] and an error-bounded tailored lossy compressor in our simulation framework.

Intuitively, one might be concerned that lossy compression would introduce correlated errors in our simulation output that would be very different from the kinds of errors that physical machines would experience. We will see, however, that our lossy compression is applied in an uncorrelated fashion. We shall also see that this has an added benefit of dramatically speeding up compression time.

Using our techniques, we are able to trade computation time and fidelity for memory space. We implement our techniques on Intel-QS, a full-state quantum circuit simulator developed by Intel [59]. Intel-QS is an MPI-based distributed high-performance quantum circuit simulator that can run on supercomputing systems. By orchestrating data compression techniques and full-state high-performance simulation techniques, our approach is capable of simulating intermediate-scale general quantum applications and hence obtain effective results of calibration, verification, and benchmarking for NISQ quantum machines.

Our approach integrates knowledge of quantum computation and data compression techniques to reduce the memory requirement of quantum circuit simulations such that our technique allows us to simulate a larger quantum system with the same memory capacity. We provide one more option in the set of tools to scale quantum circuit simulation. The main contributions of our work are as follows.

- We present a new technique to reduce memory requirements of full-state simulations of general quantum circuits by using data compression. Reducing memory requirements allows us to increase the number of qubits in our full-state simulations.
- We design a novel lossy compression method to optimize both compression ratios and compression speed for quantum circuit simulations. This lossy compression technique can be combined with several existing simulation techniques to further reduce the memory footprint.
- We implement our general quantum circuit simulation framework on the Theta supercomputer at Argonne National Laboratory (ANL).
- Our experimental results show that our approach reduces the memory requirement of simulating the 61-qubit Grover's search quantum circuit from 32 exabytes to 768 terabytes of memory. Based on the state-of-the-art simulation techniques, the results suggest that our technique can increase the simulation size by 2 to 16 qubits for general quantum applications with 0.976 simulation fidelity on average.

Our paper is organized as follows. In Section 2, we introduce the basics of quantum computation and simulation as well as the data compression techniques. In Section 3, we present our simulation design. In Section 4, we describe our lossy compression technique, and in Section 5 we evaluate our approach. We discuss future directions and provide conclusions in Section 6.

2 BACKGROUND AND RELATED WORK

In this section, we provide a brief overview of the quantum computation and discuss related work on quantum circuit simulations. We then present the relevant background on compression techniques.

2.1 Principles of Quantum Computation

A qubit is a two-level quantum system, and the state $|\psi\rangle$ can be expressed as

$$|\psi\rangle = a_0 |0\rangle + a_1 |1\rangle , \quad (1)$$

where a_0 and a_1 are complex amplitudes and $|a_0|^2 + |a_1|^2 = 1$. $|0\rangle$, and $|1\rangle$ are two computational orthonormal basis states. The quantum state can also be represented as follows.

$$|\psi\rangle = a_0 \begin{bmatrix} 1 \\ 0 \end{bmatrix} + a_1 \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} a_0 \\ a_1 \end{bmatrix} \quad (2)$$

More generally, the state of an n-qubit quantum system can be represented by using 2^n amplitudes, as follows.

$$|\psi\rangle = a_{0\ldots00} |0\ldots00\rangle + a_{0\ldots01} |0\ldots01\rangle + \ldots + a_{1\ldots11} |1\ldots11\rangle \quad (3)$$

The squared magnitudes have to sum up to 1, that is,

$$\sum_i |a_i|^2 = 1. \quad (4)$$

In quantum computation, quantum gates are applied to the quantum system. All gates are represented in matrix form. General single-qubit gates and two-qubit controlled gates are known to be universal [24]. Applying a single-qubit gate U to the kth qubit can be represented by a unitary transformation

$$A = I^{\otimes n-k-1} \otimes U \otimes I^{\otimes k}, \quad (5)$$

where I is a 2×2 identity matrix and U is a 2×2 unitary matrix.

2.2 Quantum Circuit Simulation

Previous studies have provided various types of simulators [55]. Generally, in order to simulate a quantum circuit with n qubits and depth d, there are several simulation approaches [1, 6, 49]. Different simulation approaches have different purposes and benefits.

Schrödinger algorithm. This strategy maintains the full-state vector in memory and updates the state vector in every time step [20, 59]. Since the space grows exponentially with the number of qubits, the physical memory limits the simulation size. The time complexity is polynomial with the number of gates (or circuit depth), and hence this approach is capable of simulating arbitrary depth of circuits. This simulation approach can simulate the supremacy circuits of 45 qubits on the Cori II supercomputing system using 0.5 petabytes [33], and Li et al. further optimize the simulator specifically for the 49-qubit supremacy circuits [43].

Feynman paths algorithm. This approach calculates the amplitude a_x for any n-bit string $x \in \{0, 1\}^n$ by following all the paths from a final state to the initial state. For the Feynman paths algorithm, this approach requires $O(2^{dn})$ time to perform the simulation [53], and hence this simulation technique is suitable only for low-depth quantum circuits.

Tensor network contractions. This approach uses tensor networks to represent quantum circuits [7, 49, 50]. The time and space cost for contracting such tensor networks is exponential with the treewidth of the underlying graphs. Therefore, this approach is impractical to simulate large quantum circuits. Several studies have proposed to use tensor network simulation technique to simulate low-depth supremacy circuits [10, 15, 53, 63]. To trade the simulation fidelity for computational resources, the *approximate* simulation is proposed [10, 48]. The previous studies of approximate simulations target the overall circuit fidelity at 0.005 [48, 63], but if we want to use the simulation results to help calibrate and validate the real machines, we might need higher circuit fidelity. As for quantum software development, several types of quantum applications require intermediate measurement [4, 5, 12, 29]. In addition, recent studies propose to insert assertions in quantum programs for software debugging by checking the full-state distribution [34]. Tensor network simulation techniques do not effectively support intermediate measurement and full-state assertion checking.

Other strategies for simulating quantum circuits also exist. The *Gottesman-Knill theorem* [30] states that circuits consisting of only Clifford gates can be efficiently simulated in polynomial time, hence there are simulation techniques for the circuits with a restricted gate library [2, 13, 28, 30]. In [70], the decision diagram is used to simulate circuits that consist of Clifford+T gates. This approach exploits redundancies in a quantum state to gain more compact representations. Approximate simulation techniques also have been proposed for circuits using only restricted gates [13, 69, 71].

2.3 Data Compression Techniques

With the vast volumes of data being produced by extreme-scale scientific research and applications, various data compression techniques have been developed for years. Basically, scientific researchers mainly adopt two types of compressors: lossless compressors or error-bounded lossy compressors. Lossless compressors usually adopt both variable-length encoding algorithms (such as Huffman encoding [35] and arithmetic encoding [65]) and dictionary coders such as LZ77/78 [67]. In most cases, however, lossless compressors such as Gzip [21], Zstd [19], and Blosc [8] cannot effectively compress scientific data because the ending mantissa bits of floating-point values are random such that it is hard to find exactly the same patterns in the data stream. Some studies [22] show that lossless compressors always suffer from low compression ratios (around 2:1 in most cases), which is far less than enough for today's extreme-scale high-performance computing (HPC) applications. Accordingly, error-bounded lossy compression has been widely treated as the best solution to such a big scientific data issue, because it not only can significantly reduce the data size but also can control the data distortion according to the user's requirements.

Error-bounded lossy compressors may have distinct designs and implementations, so selecting the most appropriate compression technique is critical to our research. All existing error-bounded lossy compressors can be categorized into two models: data-prediction based and domain-transform based, which are described below.

- *Data-prediction-based compression model.* This model tries to predict each data point as accurately as possible based on its neighborhood in spatial or temporal dimension and then shrinks the data size by some coding algorithm such as data quantization [62] and bit-plane truncation. A typical example compressor is SZ [44], which involves four compression steps: (1) data prediction, (2) linear-scaling quantization, (3) entropy-encoding, and (4) lossless compression. The errors

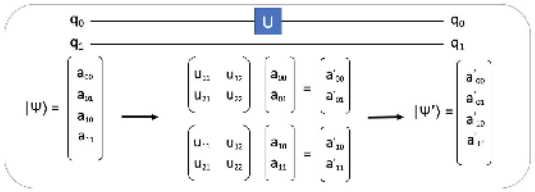


Figure 1: Example of two-qubit quantum state with single-qubit gate operations

are introduced and controlled at step (2). Other examples include ISABELA [42] and FPZIP [46].

- *Domain-transform-based compression model.* This model needs to transform all the original data values to another nonorthogonal coefficient domain for decorrelation and then shrink the data size by applying some optimized coding algorithms such as embedded coding [45]. A typical example compressor is ZFP [45], which performs the classic texture compression by leveraging three techniques in each 4^d block (where d is the number of dimensions): (1) exponent alignment, (2) (non)orthogonal block transform, and (3) embedded coding. The compression errors (or distortion of data) are introduced and controlled only at step (3). The other two techniques are VAPOR [18] and Sasaki et al.'s compressor [57], which both adopt the Wavelet transform in the domain transform step.

All the existing state-of-the-art error-bounded lossy compressors are designed or assessed mainly for visualization, so they are not optimized for the requirement of data fidelity and compression quality in the context of quantum computing simulation. In this sense, we first characterize the effectiveness of the existing state-of-the-art lossy compressors on quantum circuit simulation results and then exploit a fairly efficient compression method beyond the existing lossy compressors. In our study, we investigate two types of error controls, pointwise absolute error bound and pointwise relative error bound, because they have been supported by the existing state-of-the-art lossy compressors well.

- Absolute Error Bound (denoted by e). With this type of error control, the compression errors (defined as the difference between the original data value and its corresponding decompressed value) of all data points must be strictly limited in the required bound; in other words, d_i' must be in $[d_i - e, d_i + e]$, where d_i and d_i' refer to an original data value and the corresponding decompressed value, respectively.
- Relative Error Bound (denoted by ϵ). With this type of error control, the data distortion must respect the following inequality for each data point: $|d_i - d_i'| \leq \epsilon d_i$. Obviously, the smaller the original value is, the smaller the compression error it will get. The relative error bound is particularly useful to applications requiring multiple error controls depending on the data values.

3 SIMULATION DESIGN

We aim to simulate general quantum circuits with high fidelity. We integrate our compression techniques into the quantum circuit simulation such that the simulation scale can be increased with the same memory capacity. Our technique allows the Schrödinger-style simulation to trade the computation time and simulation accuracy for memory space by applying lossy compression techniques to state vectors. The lower compression error bound gives us the higher simulation fidelity, but the higher compression error bound will give us a higher compression ratio so that we can simulate the quantum circuits with larger numbers of qubits.

3.1 Overview of Our Simulation Flow

To demonstrate the practicality of our design, we integrate our compression techniques into Intel-QS [59], a distributed quantum circuit simulator on a classical computer.

As mentioned in Section 2, applying a single-qubit gate to the kth qubit can be represented by a unitary transformation $A = I^{\otimes n-k-1} \otimes U \otimes I^{\otimes k}$. In the simulation, however, we do not need to build the entire unitary matrix A to perform the gate operation. Figure 1 shows an example of applying a single-qubit gate to a two-qubit system. Applying a gate U to the first qubit corresponds to applying the 2×2 unitary matrix to every pair of amplitudes, whose subscript indices have 0 and 1 in the first bit and all remaining bits are the same. In the same way, performing a single-qubit gate to the second qubit is to apply the unitary to every pair of amplitudes whose subscript indices differ in the second bit. More extensively, applying a single-qubit gate to the k-th qubit of an n-qubit quantum system is to apply the unitary to every pair of amplitudes whose subscript indices have 0 and 1 in the k-th bit, while all other bits remain the same.

$$\begin{bmatrix} a'_{*\ldots*0_k*\ldots*} \\ a'_{*\ldots*1_k*\ldots*} \end{bmatrix} = \begin{bmatrix} u_{11} & u_{12} \\ u_{21} & u_{22} \end{bmatrix} \begin{bmatrix} a_{*\ldots*0_k*\ldots*} \\ a_{*\ldots*1_k*\ldots*} \end{bmatrix} \tag{6}$$

As for a generalized two-qubit controlled gate, the unitary is applied to a target qubit t if the control qubit c is set to $|1\rangle$; otherwise tth is unmodified.

$$\begin{bmatrix} a'_{*1_c\ldots*0_t*\ldots*} \\ a'_{*1_c\ldots*1_t*\ldots*} \end{bmatrix} = \begin{bmatrix} u_{11} & u_{12} \\ u_{21} & u_{22} \end{bmatrix} \begin{bmatrix} a_{*1_c\ldots*0_t*\ldots*} \\ a_{*1_c\ldots*1_t*\ldots*} \end{bmatrix} \tag{7}$$

Figure 2 shows an overview of our simulation design. The Message Passing Interface (MPI) [31] is used to execute the simulation in parallel.

Assuming we simulate n-qubit systems and have r ranks in total, the state vector is divided equally on r ranks. To reduce the memory requirement, we further divide the partial state vector into n_b blocks on each rank. Each block is stored in compressed format on the memory. To complete a gate operation, we need to apply matrix multiplication to the pair of amplitudes whose subindices are 0 and 1 at the target qubit position, so at most two blocks are decompressed, $Vector_x$ and $Vector_y$, and then update the state vectors. After all the amplitudes in $Vector_x$ and $Vector_y$ are updated, we compress the state vectors and move to the next two blocks. Once all the blocks have been updated, a gate operation is completed.

In this way, the total number of bytes required for the simulation of a rank is as follows:

$$nbBytes = \sum_i sizeof(CB_i) + 2(\frac{2^{n+4}}{r \times n_b}), \tag{8}$$

where CB_i is the ith compressed block.

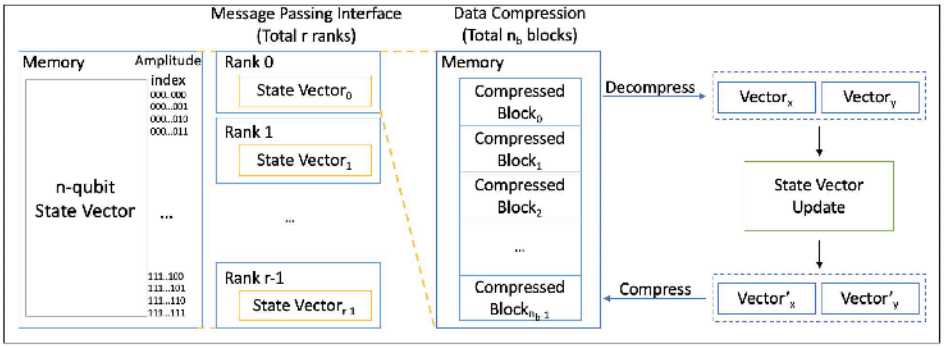


Figure 2: Simulation overview

3.2 MCDRAM Memory Configuration

Multichannel DRAM (MCDRAM) is a high-bandwidth, low-capacity memory, packaged with the Intel Xeon Phi processor [60]. Because several repeated compression and decompression operations are involved in the simulation, we can achieve the performance improvement by utilizing MCDRAM. Every time the block is decompressed, we decompress the state vectors to MCDRAM. To allocate memory in MCDRAM, we use *mkl_malloc* function to acquire memory space. This memory allocation strategy directly improves the performance of compression, gate operation, and decompression. To achieve this, we set the machine to *equal* mode, 50% cache and 50% flat.

3.3 Integration Details

We build our compressor as a C library and add it into the Intel-QS building process[1]. In the initialization of the simulation process, we create the state vectors in blocks, and compress them as compressed blocks. During the simulation, if the process needs to update the state vector, it calls our compressor library to decompress the block to the pre-allocated MCDRAM. After the state vector update, the block is compressed, and the process moves to the next block.

Since we allow decompression of only two blocks for each rank at the same time, we must select the corresponding blocks to be decompressed. For n-qubit systems, assuming we have r ranks and each block contains b amplitudes, the amplitude index string can be divided into three segments (Figure 3). When we execute a single-qubit gate computation on the target qubit position q, we find the corresponding blocks according to the segment q belongs to.

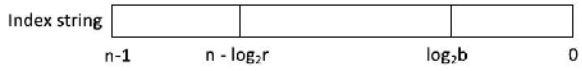


Figure 3: Amplitude index segments

- $q < log_2 b$: Both amplitudes are in the same block.
- $n - log_2 r \leq q \leq log_2 b$: Both amplitudes are in the same rank but with different blocks. We can use the number q to find the corresponding blocks in the rank.
- $q > n - log_2 r$: The pair of amplitudes are in the different ranks. The blocks have to be exchanged between different ranks.

[1]More integration details and the source code can be found at https://github.com/ryanxw/compressed_qcsim.

Two-qubit gate operations are similar to single-qubit gate operations, but we modify the amplitudes only when the control qubit is set to $|1\rangle$. According to the position of the control qubit c, we also have three cases to determine whether the amplitudes should be modified:

- $c < log_2 b$: If cth bit is 0, the amplitude is left unmodified.
- $n - log_2 r \leq c \leq log_2 b$: If cth bit is 0, the whole block is left unmodified.
- $c > n - log_2 r$: If cth bit is 0, the whole rank is left unmodified.

3.4 Compressed Block Cache

For most of the quantum circuits, the amplitudes may share the same value [70]. By exploiting redundancies in the quantum state, we can reduce the computation time significantly by constructing a compressed block cache.

Figure 4: Compressed block cache line

Figure 4 shows the contents of a compressed block cache line. The first element (OP) is the gate operation and the target qubit position. The second and third elements (CB_1, CB_2) are the compressed blocks before the gate operation. The rest of the elements (CB'_1, CB'_2) are the compressed blocks after the gate operation. Each rank maintains a memory space for the compressed block cache with 64 cache lines. When a gate operation is executed, our computation procedure first checks whether the pattern (OP, CB_1, CB_2) is in the cache. If there is a cache hit, then the computation is done by directly returning the blocks (CB'_1, CB'_2), and thus the performance is improved by reducing the compression, computation, and decompression time.

The cache replacement policy is least recently used. If there is no redundancy in the quantum state, the cache hit rate will be 0, and this will introduce the cache miss penalty in our simulation. Thus, our simulator will disable the compressed block cache if the cache hit rate is always zero.

3.5 Simulation Checkpoint

In general, most supercomputing systems have a 24-hour wall-time limit. This runtime constraint puts a circuit depth limit on the simulation. However, one can save the compressed blocks before terminating the job and then resume the task by loading the compressed blocks in the next job submission.

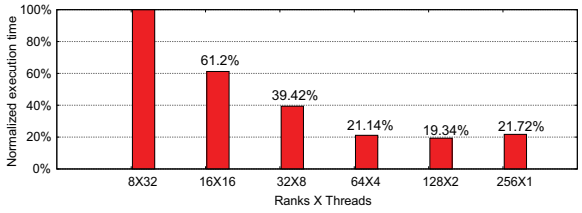


Figure 5: Normalized execution time for running 35-qubit random circuit simulations.

3.6 MPI Configuration

To understand the performance under different MPI rank configuration, we run the 35-qubit random circuit simulation with various ranks per node (Figure 5). On the KNL nodes, each node has 64 cores and 256 threads. We found that the setting of 128 ranks per node gives the best performance.

3.7 Variable Error Bound Compression

To keep the simulation accuracy, we use the lossless compression Zstd[19] in the beginning of the simulation when the compression ratio is still high enough to fit all compressed blocks in the memory space. During the simulation, the quantum state becomes more and more complicated, and hence the lossless compression ratio is lower. When the compression ratio is too low to fit all compressed blocks in the memory, our simulation will use lossy compression to compress the state vectors. To control the error, we use a pointwise relative error bound to compress the state vector. This compression mode guarantees that the decompressed data point $|D'|$ must be in the range of $(|D(1-\delta)|, |D|)$, where D is the original data and δ is the error bound. In this work, we have five different levels of error bounds: 1E-5, 1E-4, 1E-3, 1E-2, and 1E-1. Whenever the compression ratio is not enough, the error bound is relaxed to the next level (larger error). We give a detailed discussion about our lossy compression technique in the next section.

3.8 Lower Bounds on Simulation Accuracy

Since lossy compression is used in our simulations, information is lost with every lossy compression, causing a decrease in the simulation's overall accuracy. The accuracy of our simulation can be quantified by the state fidelity, a measure of the similarity of two quantum states [51].

The fidelity takes values between 0 and 1, with higher fidelity indicating greater similarity between the two states. A fidelity of 1 would indicate that the two quantum states are the same. The pure state fidelity between the ideal output state, ψ_{ideal}, and the output state from our simulation, ψ_{sim}, can be described by the following simplified equation [51].

$$F(ideal, sim) = |\langle \psi_{ideal} | \psi_{sim} \rangle| \tag{9}$$

Since the errors are bounded in our simulation, the fidelity can be estimated by propagating the maximum error bounds at each gate to calculate the maximum decrease in fidelity (from the ideal fidelity of 1) that comes from each lossy compression and then finding their combined impact on the overall maximum decrease in fidelity. Suppose that a gate has a percentage error of δ. Then if a complex amplitude a_i in the ideal state vector is represented as

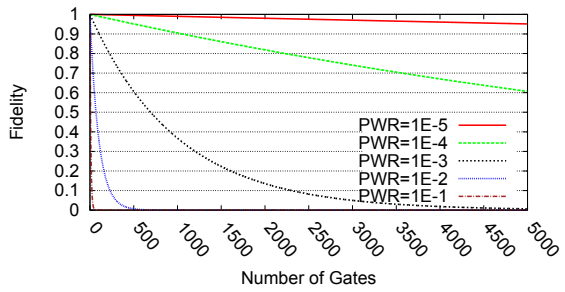


Figure 6: Minimum bounds for fidelity with an increasing number of gates at different error levels

$a + bi$, the corresponding amplitude after our lossy compression in the simulation can be represented by $a' + b'i$, where $|a'| \geq |a(1-\delta)|$ and $|b'| \geq |b(1-\delta)|$. Thus, we see that the state fidelity can be calculated as follows:

$$F(ideal, sim) = |\langle \psi_{ideal} | \psi_{sim} \rangle| \geq \sum_{i=0}^{2^n-1} a_i^2 (1-\delta) = 1 - \delta. \tag{10}$$

So we see that the minimum fidelity drops by a factor of $(1-\delta)$ after applying a lossy compression with a maximum percentage error bound of δ. Suppose that for the ith gate, the error bound is δ_i on the lossy compression. Then the lower bound on the fidelity after applying this compression will drop to $(1-\delta_i)$ times the lower bound on the fidelity calculated before lossy compression on this gate. Combining the contributions of all the gates in the simulation allows us to calculate the lower bound on the simulation fidelity as

$$F(ideal, sim) \geq \prod_i (1-\delta_i). \tag{11}$$

If all the δ_i were 0, the simulation fidelity would be 1, which makes sense because our simulation would then be identical to the simulation without lossy compression against which we are calculating the fidelity. As mentioned before, in our simulation with lossy compression, the error bounds δ_i can be set to 0 (lossless), 1E-5, 1E-4, 1E-3, 1E-2, and 1E-1. Figure 6 shows how fidelities change with the number of gates when different error levels are applied.

4 ADAPTIVE COMPRESSION OPTIMIZED FOR QUANTUM CIRCUIT SIMULATION

In this section, we propose a novel error-bounded compression method that can significantly control memory footprint for full-state quantum circuit simulations that were unreachable before.

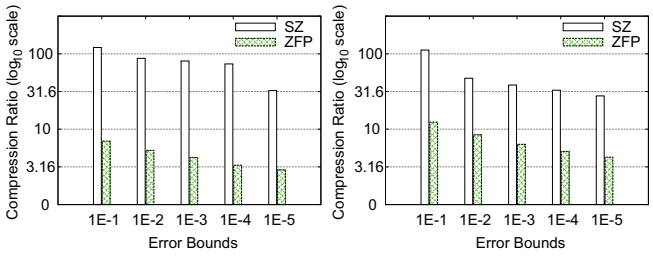
4.1 Assessment of Existing Error-Bounded Lossy Compressors

First, we explore the best solution from among the existing compressors for the quantum circuit simulations. We choose SZ [23, 68], FPZIP [46], and ZFP [45] in our exploration because they have been confirmed as the best error-bounded compressors on many scientific datasets [23, 47, 68].

We perform the assessment using two well-known quantum circuits: a quantum approximate optimization algorithm (QAOA) [27] and the random circuit proposed by Google to show quantum supremacy [9]. In this analysis, we run 36 qubits for both circuits and denote them as qaoa_36 and sup_36, respectively.

As discussed in Section 2.3, there are two types of error bounds, both of which have been widely used by scientific applications, so we evaluate the compression ratios for quantum circuit simulation data based on both types of errors. Since each rank involves hundreds or thousands of data blocks each having different value ranges, we perform the compression based on the absolute error bound in the regard of value range in our evaluation, without loss of generality. That is, we set the absolute error bound to a fixed percentage of the value range in each block. For instance, 1E-2 means 1% of the value range in the following figures.

Figure 7 presents the compression ratios of SZ and ZFP based on different absolute error bounds. FPZIP is missing in this figure because it does not support an absolute error bound. We can see that SZ always leads to one or two orders of magnitude higher compression ratios than ZFP does at every error bound. For instance, on qaoa_36, SZ can lead the compression ratios up to about 100:1, while ZFP's compression ratios are always less than 10:1. For the dataset sup_36, the compression ratios of SZ and ZFP are about 28~126 and 4.25~12.6, respectively.

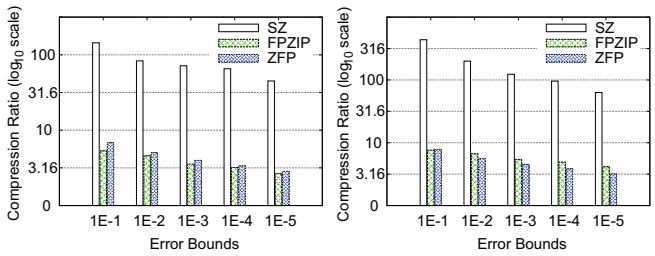


(a) qaoa_36 (b) sup_36

Figure 7: Compression ratio of SZ vs. ZFP (absolute error)

For the pointwise relative error bound, for ZFP, we transform the original data to the logarithm domain and then compress the transformed data with absolute error bounds, for fairness of the comparison. Such a log-preprocessing-based compression has been validated as the best way to do the pointwise relative-error-bounded compression [66]. As for FPZIP, it provides a so-called precision number (=4~64) to control the pointwise relative error bound. The larger the precision number is, the lower the pointwise relative error bound obtained. We set the precisions to 16, 18, 22, 24, and 28 for FPZIP in our experiments because they correspond to the pointwise relative error bounds of 1E-1, 1E-2, 1E-3, 1E-4, and 1E-5 approximately. Figure 8 clearly shows that SZ always leads to much higher compression ratios than do the other two compressors with the same pointwise relative error bounds.

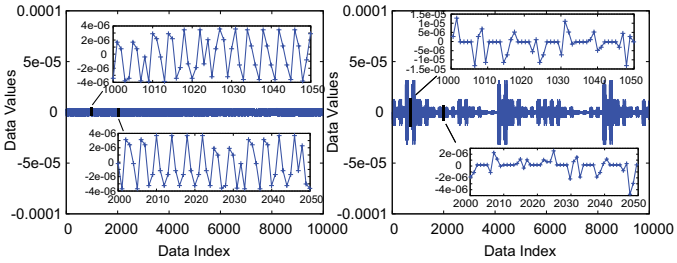
Based on our analysis, we conclude that SZ is superior to the other two compressors. The key reason is as follows. For ZFP, it substantially replies on the high smoothness of data, especially when the error bounds are set to a relatively low value. However, the quantum simulation data are not smooth at all, as illustrated in Figure 9, such that the domain-transform in ZFP would totally lose its effectiveness, leading to poor compression ratios. The key difference between FPZIP and SZ is that the former adopts a totally different encoding method, unlike SZ adopting linear-scaling quantization + Huffman encoding + Zstd. In what follows, we treat SZ



(a) qaoa_36 (b) sup_36

Figure 8: Compression ratio of SZ,FPZIP,ZFP (relative error)

as the baseline and propose a new lossy compression method that is more effective on the quantum circuit simulation data.



(a) qaoa_36 (b) sup_36

Figure 9: Illustration of Value changes of quantum circuit simulation data: the data exhibit a high spikiness such that existing lossy compressors cannot work effectively.

4.2 Optimizing the Compression Strategy

In our work, we adopt a hybrid, adaptive compression pipeline to control the memory footprint while keeping the high fidelity of the simulation results and relatively low time overhead. Specifically, we observe that at the early stage of the simulation, a large majority of the data are zero. In this situation, the lossless compressor Zstd can also get a satisfactory compression ratio, without any loss of data fidelity. Although SZ leads to much higher compression ratios in this situation, it may introduce data distortion, causing degraded fidelity unexpectedly. As the simulation goes on, however, the simulation data will become more and more complicated, such that Zstd will suffer from low compression ratios. In this situation, the error-bounded lossy compressor is helpful in controlling the memory footprint while keeping a high fidelity of simulation results. Thus, we adopt both Zstd and error-bounded lossy compression during each entire simulation. More details are in Section 3.7.

Developing a fast, effective error-bounded lossy compressor is critical to the overall simulation results (including fidelity and simulation time). In particular, we need to use a pointwise relative error bound to do the compression, as discussed in Section 3. To this end, we explored a battery of novel compression strategies to optimize both the compression ratios and compression speed for quantum circuit simulations, as described below.

Solution A (SZ 2.1): This solution is the state-of-the-art error-bounded lossy compressor - SZ [23, 62, 68]. In the context of quantum circuit simulations, the data can be treated only as a 1D array. In SZ 2.0, the pointwise relative-error-bounded compression involves the following steps: (1) logarithm data transform; (2) absolute-error-bounded compression on log-transformed data (Lorenzo prediction [36] + quantization [62]); (3) Huffman encoding; and (4) Zstd lossless compression. Since log-transform is expensive, the SZ development team developed SZ 2.1 leveraging a table lookup method to accelerate the compression significantly [68], without degrading the compression ratios. However, the compression/decompression speed still cannot meet the expected level for quantum circuit simulations (to be shown later), which motivates us to further explore a new, faster compression method instead.

Solution B (SZ 2.1 with complex type supported): Quantum state amplitudes are stored in the form of complex data type, in which the real numbers and imaginary numbers are stored alternatively, such that the prediction accuracy would be degraded to a certain extent, leading to limited compression ratios. Accordingly, Solution B improves the prediction accuracy by performing the prediction on the real numbers and imaginary numbers, respectively. In addition, we set the maximum number of quantization bins to 16,384 (unlike the default setting of 65,536 in SZ 2.1), which can significantly improve the compression/decompression rate (to be shown later).

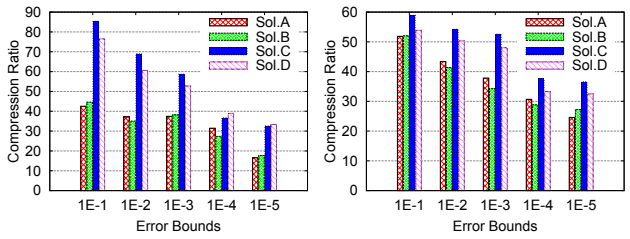
Solution C (XOR leading-zero data reduction + bit-plane truncation + Zstd): Solution B can improve the compression ratios in some cases, but its compression speed is still lower than expected such that the total simulation would slow significantly compared with the original compression-free execution. To reduce the compression/decompression time significantly, we developed Solution C, a simple yet efficient compression pipeline that is particularly suitable for quantum circuit simulation. This method involves three key steps. For each data point, it first leverages *XOR leading-zero data reduction method* [14, 23], which uses a two-bit code to record the number of exactly the same bytes between the current value and its preceding value. Then the algorithm truncates the insignificant bit-planes based on the required relative error bound ϵ. Specifically, the significant number of bits can be calculated as follows.

$$Sig\_Bit\_Count = Bit\_Count(Sign\&Exp) - EXP(\epsilon), \qquad (12)$$

where $EXP(\epsilon)$ refers to the exponent of the relative error bound ϵ (e.g., $EXP(0.01)$=−7) and $Bit\_count(Sign\&Exp)$ is the total number of bits used to represent sign and exponent in the IEEE 754 format (e.g., it is equal to 12 for double precision). We then adopt Zstd lossless compression to shrink the data significantly.

Solution D (Reshuffle + Solution C): Since the simulation data are stored in the complex data type (i.e., real number and imaginary number alternatively), one plausible idea is to reorganize the data into real numbers and imaginary numbers separately before compressing the data. Such a reshuffle step may help improve compression ratio especially when the real numbers and imaginary numbers are located in different non-overlapped value ranges. The reason is that in Solution C, the last step Zstd involves a dictionary-matching stage (LZ77 [67]) leveraging potential repeated patterns in the data streams and the reshuffling step that separates the real numbers and imaginary numbers may improve the pattern matching to a certain extent. Accordingly, we developed Solution D, which might have higher compression ratios with a slightly lower compression/decompression rate.

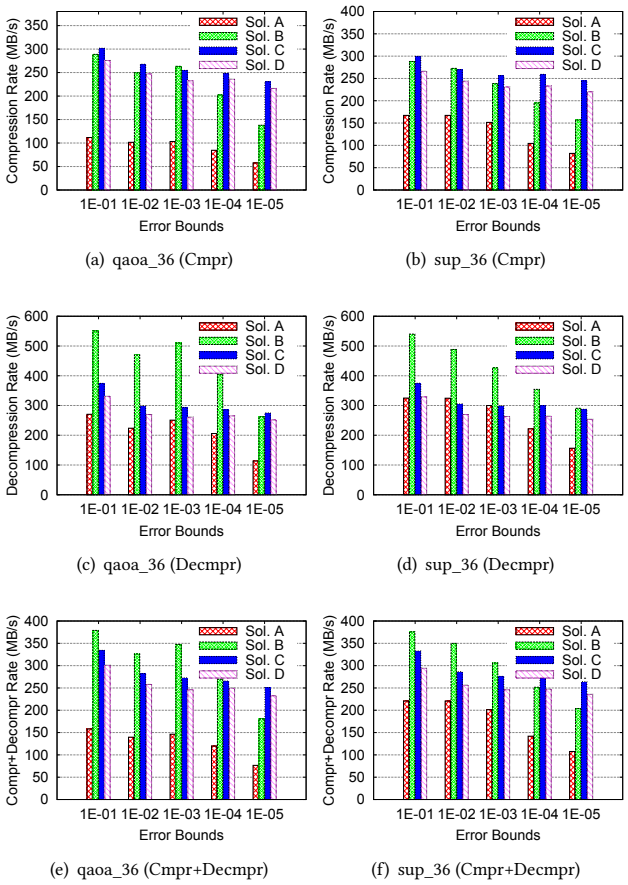
We evaluate all the four solutions by doing the compression and decompression with the two simulation datasets qaoa_36 and sup_36. Figure 10 presents the compression ratios of the four solutions. We can see that the classic compressor SZ 2.1, either supporting complex type (Solution B) or not (Solution A), suffers from about 30%∼50% lower compression ratios than do Solutions C and D. The likely reason is that the simulation data exhibit spiky changes (as illustrated in Figure 9) such that the SZ compressor always suffers from low prediction accuracy. Besides, we note that the solution C may lead to slightly higher compression ratios than does the solution D in some cases, which also makes sense as explained as follows. Note that the only difference between these two solutions is the extra reshuffle step in the solution D. This step might affect the compression ratio only because of the LZ77 stage in the last step Zstd of the two solutions, as analyzed previously. With this in mind, the compression ratio actually may not change significantly in between since the pattern matching efficiency for the two solutions could be very similar. On the one hand, Zstd improved LZ77 by adopting a pretty large window size. On the other, the real numbers and imaginary numbers are of the similar value ranges (as shown in Figure 9), such that the reshuffling step may not induce more regular data. In this sense, the solution C and D can be deemed having comparative compression ratios on the QC simulation datasets.



(a) qaoa_36 (b) sup_36

Figure 10: Compression ratio of 4 solutions (relative error)

We present the compression and decompression rate in Figure 11 (single-core performance). This figure clearly shows that Solutions B, C, and D have much higher compression rates and decompression rates than the classic SZ 2.1 (Solution A) does. The key reason Solution B is faster than Solution A is that it predicts the data in terms of the complex data type, which may get higher prediction accuracy, thus leading to faster encoding thereafter. Moreover, we set a lower maximum number of quantization bins in Solution B such that it keeps a relatively high compression rate in the case with low relative error bounds such as 1E-5. The reason the Solutions C and D run much faster than Solutions A and B do is that they totally remove the three costly steps—prediction, quantization, and Huffman encoding—in the compression. We can also observe that Solution C is slightly faster than Solution D because of the extra reshuffle step in Solution D.



(a) qaoa_36 (Cmpr) (b) sup_36 (Cmpr)

(c) qaoa_36 (Decmpr) (d) sup_36 (Decmpr)

(e) qaoa_36 (Cmpr+Decmpr) (f) sup_36 (Cmpr+Decmpr)

Figure 11: Compress/decompression rates (relative error)

We present in Figure 12 the distribution of the maximum compression error (pointwise relative errors) per data block for all four compression solutions on qaoa_36 and sup_36, respectively. Similar to the previous evaluations, the experimental dataset comes from one rank involving hundreds of data blocks (each with 1,048,576 complex-type data points, bringing a total of 16 MB per block).

From Figure 12 we observe that all four solutions respect the compression errors well in terms of different pointwise relative error bounds (1E-1∼1E-5). We note that the error distribution curves of Solutions C and D overlap, which is explained as follows. In fact, they both avoid the data prediction and quantization step such that the compression errors have nothing to do with the order of the data points. The only difference between them is the extra reshuffle step in Solution D. Thus they have exactly the same compression errors in each block.

Moreover, we note that Solutions C and D exhibit much lower compression errors than do Solutions A and B in general. The reason is as follows. As illustrated in Figure 13 (a), Solutions A and B both adopt the data prediction and linear-scaling quantization [62]. Specifically, they predict each data point and then approximate its value by the error-bound-quantized distance [62] between the true raw value and the predicted value. Note that the compression errors are determined by the difference between the decompressed values and the true values (as shown in Figure 13 (a)). Hence, if the error bound is relatively small, the quantization bin size would be small too, then the true values would be located at a rather random postion in a quantization bin because of the likely large distance between the predicted value and true value, thus leading to a uniform distribution. By comparison, Solutions C and D calculate the significant bit-planes based on user-required pointwise relative errors, and each bit-plane spans a certain value range. As illustrated in Figure 13 (b) (with a single-precision value 3.9921875 as an example), truncating different bit planes will lead to discrete decompressed values and relative errors. Suppose the relative error bound is set to 0.01, then we need to keep 15 leading bits and the decompressed value would be 3.96875 with a relative error of 0.005871, which is actually lower than the error bound 0.01. That is, the compression errors of the solutions C and D are generally somewhat lower than the desired error bound.

Such overpreservation of error can be observed more clearly by plotting the distribution of normalized compression errors compared with the error bounds for Solution C, as presented in Figure 14. We plot the cumulative distribution function (CDF) of the normalized pointwise relative errors compared with the corresponding error bounds for one random block of data, because too many data points are involved in the whole simulation and other blocks actually exhibit similar error distributions. In the figure we can observe that (1) all compression errors are indeed confined within the required relative error bound; (2) the compression errors follow a uniform distribution; and (3) most of the compression errors are actually much lower than the required error bound, bringing an extra benefit to the control of data distortion in the simulation.

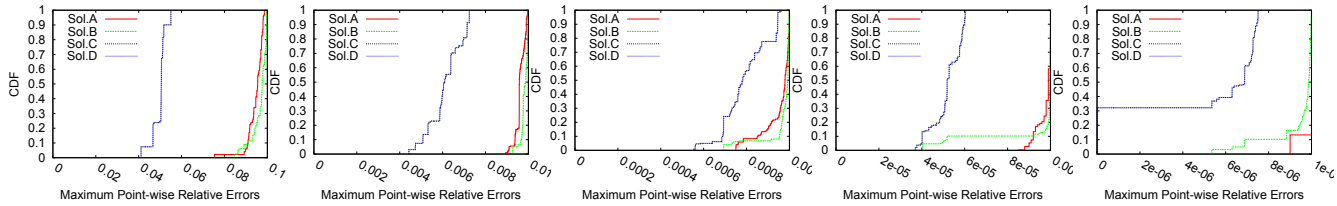
Moreover, the solution C leads to non-correlated compression errors (point-wise relative errors) for the quantum circuit simulation datasets. The reason is that the relative errors are determined by the high-order bit-values in the truncated insignificant bit-planes. Since the quantum circuit simulation data exhibit a rather high randomness (as presented in Figure 9), the truncation errors are supposed to be fairly random as well. We confirm this point by calculating the lag-1 autocorrelation coefficients of the compression errors. The autocorrelation value is always in [-1,1]; the closer to zero, the higher level of non-autocorrelation is. Our evaluation shows that the autocorrelation value often ranges in [-1E-4,1E-4] if a large majority of original raw data are non-zeros in the dataset, testifying the high non-correlation feature of the solution C.

Based on this analysis, we can conclude that Solution C is a fairly good tradeoff, which can obtain a high compression ratio, low compression time and decompression time, and low compression errors with non-correlation feature. Accordingly, Solution C is our final error-bounded lossy compressor to be used in our experiments.

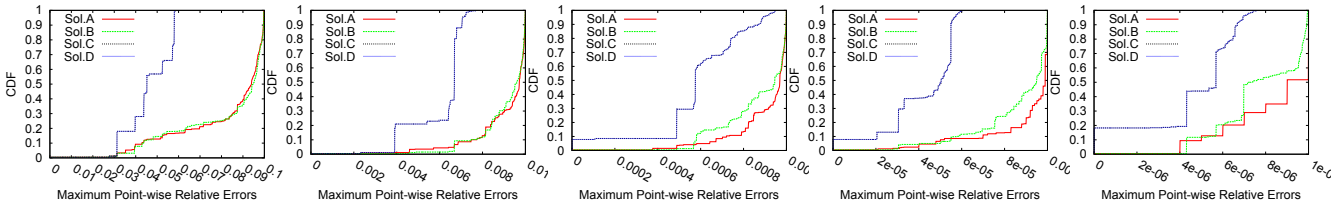
5 EVALUATION

5.1 Experimental Setup

For multinode evaluation, we performed our simulation on the Theta supercomputer at Argonne National Laboratory. Theta consists of 4,392 nodes, each node containing a 64-core Intel®Xeon Phi™ processor 7230 with 16 gigabytes of high-bandwidth in-package memory (MCDRAM) and 192 GB of DDR4 RAM [52]. The bandwidth of the MCDRAM is 400GB/s, and the average latency is 154 ns [3]. On Theta, there are wall-time limits for each jobs with

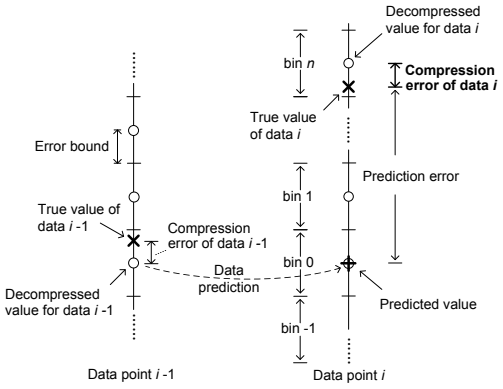


(a) qaoa_36:PWR=1E-1 (b) qaoa_36:PWR=1E-2 (c) qaoa_36:PWR=1E-3 (d) qaoa_36:PWR=1E-4 (e) qaoa_36:PWR=1E-5

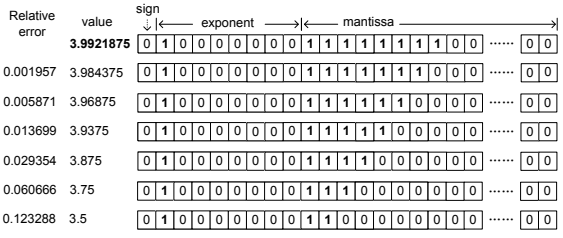


(f) sup_36:PWR=1E-1 (g) sup_36:PWR=1E-2 (h) sup_36:PWR=1E-3 (i) sup_36:PWR=1E-4 (j) sup_36:PWR=1E-5

Figure 12: Distribution of maximum relative errors on compression of qaoa_36 and sup_36 (Solution C and D overlap)



(a) Prediction and quantization in solution A/B

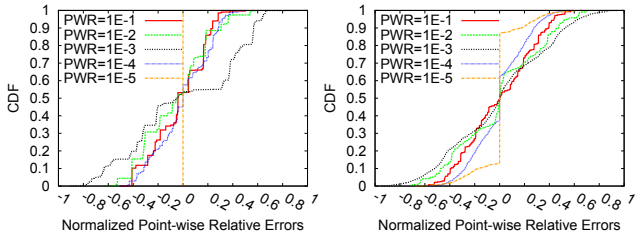


(b) Discrete relative errors when truncating bits in solution C/D

Figure 13: Illustrating why Solution C/D leads to lower compression errors than Solution A/B

different numbers of nodes. The wall-time limits are 3, 6, and 24 hours for jobs of 128 nodes, 256 nodes, and 1,024 nodes, respectively. For jobs running exceeding wall-time limits, we use our checkpoint design, described in Section 3.5, to complete the simulation.

For single-node experiments, we run our simulation on a single 64-core Intel®Xeon Phi™ processor 7210 (KNL) in the Joint Laboratory for System Evaluation (JLSE) at Argonne. There is no wall-time limit on the system.



(a) qaoa_36 (b) sup_36

Figure 14: Distribution of normalized compression errors (Solution C)

In this evaluation, we set 128 ranks per node. As described in Section 3, the state vector on each rank is divided into multiple blocks, each containing 1,048,576 amplitudes, which means each block is 16 MB. Since each rank decompresses at most 2 blocks on MCDRAM at the same time, we need to allocate $128 \times 2 \times 16MB = 4GB$ of memory on MCDRAM. Thus, we configure the memory mode by using *--attrs mcdram=equal* to operate 50% MCDRAM as flat memory and the other 50% as cache [38].

5.2 Scalability

We evaluate the scaling behavior by using a basic program that applies a Hadamard gate on each qubit. Figure 15 shows the normalized execution time for running different sizes of simulations on a single node. The scaling of our simulation technique for 51-qubit quantum circuit running on different numbers of nodes is depicted in Figure 16. For simulations of real applications that requires more time in compression, decompression, and computation, the scaling is expected to be better.

5.3 Benchmarks

To show the results against real applications, we choose multiple important quantum applications as our benchmarks. The benchmarks are selected to have different program characteristics in order to show that our simulation could work with arbitrary quantum

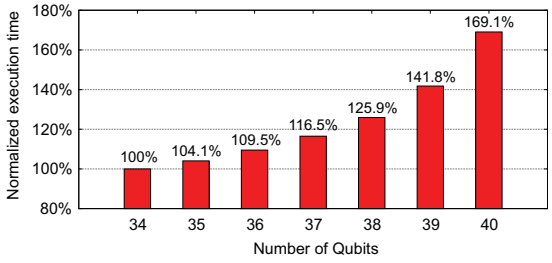


Figure 15: Normalized execution time for running various sizes of simulations on a single node.

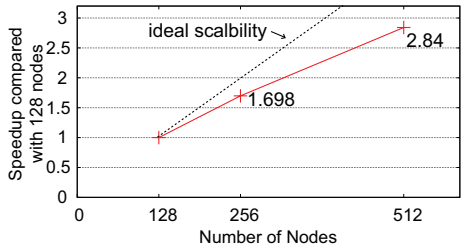


Figure 16: Scaling behavior of applying Hadamard gates on a 51-qubit system on Theta.

circuits. Our approach performs well for both shallow-circuit and deep-circuit applications. The initial state is $|0\rangle^{\otimes n}$. The programs we use for our benchmarking include the following.

- Grover: Grover's search algorithm is for database search, and it leads to significant speedups compared with classical search algorithms [32, 37]. Our benchmark uses Grover's search algorithm to find the square root number [37], and thus the oracle consists of X and Toffoli gates.
- Random circuit sampling: The circuit is proposed by Google to show the quantum supremacy, and we follow the rules to construct the random circuits [9]. Since our approach is not optimized for the circuits, we do not plan to run the random circuits with many layers. Thus, the circuit depth is 11 in our experiments.
- QAOA: The quantum approximate optimization algorithm is a hybrid quantum-classical variational algorithm. Our benchmark uses QAOA to solve MAXCUT on a random 4 regular graph problem [27]. QAOA is an important circuit because it is one of the most promising quantum algorithms in the NISQ era [54].
- QFT: This is the quantum circuit for quantum Fourier transform [37], which is an important function in many quantum algorithms (Shor's algorithm [58], phase estimation algorithm [17], and the algorithm for hidden subgroup problem [39]), and this is a deep circuit. We randomly apply X gate to the initial state as the input for the QFT in our experiments.

5.4 Experimental Results

We present our main results in Table 2. We run benchmarks with the total system memory capacities much less than the theoretical memory requirements to manifest the strength of our approach.

Our first benchmark application is Grover's search algorithm with 47, 59 and 61 qubits. Previously, 61-qubit simulations of general quantum applications were thought to be impossible because of infeasible memory requirements. Using our methods, the state vectors can be compressed with high compression ratios because the amplitude values of the state vectors of this application are similar and regular, and hence our approach can use only 768 TB to successfully perform the simulation of 61-qubit Grover's search algorithm, which theoretically requires 32 EB to execute the full-state simulation. We also use our technique to complete the 47-qubit Grover's search algorithm simulation using 24 TB instead of 2 PB, the memory requirement without our technique. Next, we present the simulation of Google's quantum supremacy random circuits. Since our method exploits non-uniformity and structure in quantum computations, it does not work as well on random circuits. If we run many layers of the circuits, we have to drop the simulation fidelity to meet the available memory capacity. Thus, we present the simulation results of depth of 11 random circuits. Although our simulation technique is not designed specifically for the supremacy circuits, our approach can simulate the circuit with 42 qubits by using 48 TB, and simulate the circuit with 45 qubits by using 192 TB. The next benchmark is QAOA. Since QAOA is an important quantum application, it is critical to the simulation of QAOA circuits with limited memory capacity. In fact, we can get higher compression ratios and still get successful results because QAOA is robust to low-fidelity. Finally, the results with QFT circuits show that our techniques are effective for simulating high-depth circuits and compress the state vectors with more than 21X compression ratios, with a fidelity of 0.962.

For the quantum applications we choose, we show that our technique can simulate the circuits with high fidelity. Among the selected benchmarks, random circuits use more entanglement than others. Since our techniques compress the state vector, more entanglement leads to less compressible vectors. Thus, our technique does better when there is less entanglement.

5.5 Discussion

We successfully increase the maximum simulation size from 45 qubits to 61 qubits for the Grover's search circuit simulation on the Argonne Theta supercomputer using less than 0.8 PB. As for simulations of the other quantum applications, we have compression ratios from 4.85X to 21.34X. The results provide evidence that we are able to simulate those applications with 47 to 49 qubits on Theta. In other words, for simulations of general circuits, our approach can increase the simulation size by 2 to 16 qubits. In this work, we seek to achieve high-fidelity simulation results for general circuits. For different purposes that do not require high-simulation fidelity, the compression ratios and simulation size would be further increased.

We use the compression ratios to estimate our approach on the Summit supercomputer [64] at Oak Ridge National Laboratory (ORNL). The expected maximum simulation size for general circuits is 63 qubits. In 2021, we will have the exascale supercomputing system Aurora [26] at Argonne. The estimated maximum simulation size will be increased from 48 qubits to 64 qubits. Our compression techniques can combine with other simulation techniques [16, 33, 43, 69, 71] to scale quantum circuit simulation.

We trade time and slight fidelity for memory space. This trade-off makes the classical simulation of several quantum circuits possible, while the simulation time grows linearly with the number of gates. The time complexity is polynomial with circuit depth.

Table 2: Experimental results. The first row shows the memory requirements of the simulations without our techniques. The ratios of our system memory sizes to the required memory sizes are presented in the fourth row. The fifth row provides the simulation time and the breakdown. The minimum compression ratio during the simulation is shown in the last row.

Benchmark	Grover			Random Circuit Sampling				QAOA			QFT
Number of Qubits	61	59	47	5×9	6×7	6×6	7×5	45	43	42	36
(Memory Requirement)	(32 EB)	(8 EB)	(2 PB)	(512 TB)	(64 TB)	(1 TB)	(512 GB)	(512TB)	(128 TB)	(64 TB)	(1 TB)
Number of Gates	314	310	305	227	261	165	208	394	344	336	3258
Number of Nodes	4096	4096	128	1024	128	1	1	1024	256	128	1
Total System Memory	768 TB	768 TB	24 TB	192 TB	24 TB	192 GB	192 GB	192TB	48 TB	24 TB	192 GB
(Sys Mem / Req.)	(0.002%)	(0.009%)	(1.17%)	(37.5%)	(37.5%)	(18.75%)	(37.5%)	(37.5%)	(37.5%)	(37.5%)	(18.75%)
Total Time (Hour)	8.14	3.48	0.49	4.87	8.64	7.96	6.23	13.34	5.83	8.65	78.98
Compression Time	1.87%	4.59%	2.04%	55.79%	40.26%	59.10%	58.57%	50.66%	44.97%	41.02%	57.86%
Decompression Time	1.87%	3.73%	4.08%	31.47%	22.19%	33.78%	30.59%	26.46%	27.64%	25.52%	37.68%
Communication Time	32.7%	20.98%	36.73%	0.12%	0.57%	0.02%	0.03%	3.03%	0.22%	0.23%	2.56%
Computation Time	63.47%	70.70%	57.15%	12.60%	36.97%	7.08%	10.8%	19.84%	27.16%	33.22%	1.9%
Time per Gate (Sec)	93.34	40.49	5.78	64.69	119.22	173.65	107.86	121.91	61.02	92.64	87.27
Simulation Fidelity	0.996	0.996	1	0.987	0.993	0.933	0.985	0.895	0.999	0.999	0.962
Compression Ratio	7.39×10^4	8.26×10^4	1.06×10^4	6.03	9.40	8.16	10.05	5.38	4.85	9.25	21.34

6 CONCLUSION AND FUTURE WORK

Our approach performs full-state simulation of general quantum circuits using data compression techniques. This method allows the Schrödinger-style simulation to trade time and simulation fidelity for memory space to increase the simulation size. Our compression techniques can combine with other simulation techniques [16, 33, 43, 69, 71]. By using our lossy compression, we can compress state vectors and reduce memory footprints significantly compared with the existing techniques [33, 59]. The compression errors are not correlated to the data, and hence the errors might be used to further simulate noise on real devices. The modern noise simulations add errors to perfect simulations. However, we could further adapt our lossy compression errors to noise models and then build a simulation which models noise naturally. In addition, we plan to implement our simulator on GPU-based supercomputing systems to reduce the compression and decompression time.

In summary, we have described our simulation techniques to reduce the memory requirement of full-state quantum circuit simulations by using data compression techniques. Our method provides a new option in the set of simulation tools to scale quantum circuit simulations. We propose a novel lossy compression technique to optimize compression ratios and compression speed for quantum circuit simulations. Using our approach, the memory requirement of simulating the 61-qubit Grover's search algorithm is reduced from 32 exabytes to 768 terabytes of memory on the Argonne Theta supercomputer using 4,096 nodes. We also present experimental results of random circuits, QAOA, and QFT simulations to show that our technique can achieve general circuit simulations. The compression ratio results further suggest that our technique can increase the simulation size by 2 to 16 qubits for general quantum applications. Our simulator provides a platform for quantum software debugging and quantum hardware validation.

ACKNOWLEDGMENTS

This research was supported by the Exascale Computing Project (ECP), Project Number: 17-SC-20-SC, a collaborative effort of two DOE organizations - the Office of Science and the National Nuclear Security Administration, responsible for the planning and preparation of a capable exascale ecosystem, including software, applications, hardware, advanced system engineering and early testbed platforms, to support the nation's exascale computing imperative. This research used the resources of the Argonne Leadership Computing Facility, which is a U.S. Department of Energy (DOE) Office of Science User Facility supported under Contract DE-AC02-06CH11357. Yuri Alexeev, Hal Finkel, and Xin-Chuan Ryan Wu were supported by the DOE Office of Science. This work is also supported by the National Science Foundation under Grant No. 1619253. This work is funded in part by EPiQC, an NSF Expedition in Computing, under grants CCF-1730449/1832377, and in part by STAQ, under grant NSF Phy-1818914.

REFERENCES

[1] Scott Aaronson and Lijie Chen. 2016. Complexity-theoretic foundations of quantum supremacy experiments. *arXiv preprint arXiv:1612.05903* (2016).
[2] Scott Aaronson and Daniel Gottesman. 2004. Improved simulation of stabilizer circuits. *Physical Review A* 70, 5 (2004), 052328.
[3] Ryo Asai. 2016. MCDRAM as High-Bandwidth Memory (HBM) in Knights Landing processors: developers guide. *Colfax International* (2016).
[4] Rami Barends, Julian Kelly, Anthony Megrant, Andrzej Veitia, Daniel Sank, Evan Jeffrey, Ted C White, Josh Mutus, Austin G Fowler, Brooks Campbell, et al. 2014. Superconducting quantum circuits at the surface code threshold for fault tolerance. *Nature* 508, 7497 (2014), 500.
[5] Charles H. Bennett, David P. DiVincenzo, John A. Smolin, and William K. Wootters. 1996. Mixed-state entanglement and quantum error correction. *Phys. Rev. A* 54 (Nov 1996), 3824–3851. Issue 5. https://doi.org/10.1103/PhysRevA.54.3824
[6] Ethan Bernstein and Umesh Vazirani. 1997. Quantum complexity theory. *SIAM Journal on computing* 26, 5 (1997), 1411–1473.
[7] Jacob Biamonte and Ville Bergholm. 2017. Tensor networks in a nutshell. *arXiv preprint arXiv:1708.00006* (2017).
[8] BlosC compressor. [n. d.]. http://blosc.org/. Online.
[9] Sergio Boixo, Sergei V Isakov, Vadim N Smelyanskiy, Ryan Babbush, Nan Ding, Zhang Jiang, Michael J Bremner, John M Martinis, and Hartmut Neven. 2018. Characterizing quantum supremacy in near-term devices. *Nature Physics* 14, 6 (2018), 595.
[10] Sergio Boixo, Sergei V Isakov, Vadim N Smelyanskiy, and Hartmut Neven. 2017. Simulation of low-depth quantum circuits as complex undirected graphical models. *arXiv preprint arXiv:1712.05384* (2017).
[11] X Bonet-Monroig, R Sagastizabal, M Singh, and TE O'Brien. 2018. Low-cost error mitigation by symmetry verification. *Physical Review A* 98, 6 (2018), 062339.
[12] Gilles Brassard. 1996. Teleportation as a quantum computation. *arXiv preprint quant-ph/9605035* (1996).
[13] Sergey Bravyi and David Gosset. 2016. Improved classical simulation of quantum circuits dominated by Clifford gates. *Physical review letters* 116, 25 (2016), 250501.
[14] M. Burtscher and P. Ratanaworabhan. 2009. FPC: A High-Speed Compressor for Double-Precision Floating-Point Data. *IEEE Trans. Comput.* 58, 1 (Jan 2009), 18–31. https://doi.org/10.1109/TC.2008.131
[15] Jianxin Chen, Fang Zhang, Mingcheng Chen, Cupjin Huang, Michael Newman, and Yaoyun Shi. 2018. Classical simulation of intermediate-size quantum circuits. *arXiv preprint arXiv:1805.01450* (2018).
[16] Zhao-Yun Chen, Qi Zhou, Cheng Xue, Xia Yang, Guang-Can Guo, and Guo-Ping Guo. 2018. 64-qubit quantum circuit simulation. *Science Bulletin* (2018).
[17] Richard Cleve, Artur Ekert, Chiara Macchiavello, and Michele Mosca. 1998. Quantum algorithms revisited. *Proceedings of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences* 454, 1969 (1998), 339–354.

[18] John Clyne, Pablo Mininni, Alan Norton, and Mark Rast. 2007. Interactive desktop analysis of high resolution simulations: application to turbulent plume dynamics and current sheet formation. *New Journal of Physics* 9, 301 (2007), 1–29.
[19] Yann Collet. 2015. Zstandard - Real-time data compression algorithm. *http://facebook.github.io/zstd/* (2015).
[20] Koen De Raedt, Kristel Michielsen, Hans De Raedt, Binh Trieu, Guido Arnold, Marcus Richter, Th Lippert, H Watanabe, and N Ito. 2007. Massively parallel quantum computer simulator. *Computer Physics Communications* 176, 2 (2007), 121–136.
[21] L Peter Deutsch. 1996. GZIP file format specification version 4.3.
[22] Sheng Di and Franck Cappello. 2016. Fast error-bounded lossy HPC data compression with SZ. In *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 730–739.
[23] Sheng Di and Franck Cappello. 2016. Fast Error-Bounded Lossy HPC Data Compression with SZ. In *IPDPS 2016*. 730–739.
[24] David P DiVincenzo. 1995. Two-bit gates are universal for quantum computation. *Physical Review A* 51, 2 (1995), 1015.
[25] Suguru Endo, Simon C Benjamin, and Ying Li. 2018. Practical quantum error mitigation for near-future applications. *Physical Review X* 8, 3 (2018), 031027.
[26] Argonne Leadership Computing Facility. 2019. Aurora Supercomputer. https://www.alcf.anl.gov/programs/aurora-esp. Online. Accessed: 2019-04-06.
[27] Edward Farhi, Jeffrey Goldstone, and Sam Gutmann. 2014. A quantum approximate optimization algorithm. *arXiv preprint arXiv:1411.4028* (2014).
[28] Leslie Ann Goldberg and Heng Guo. 2017. The complexity of approximating complex-valued Ising and Tutte partition functions. *computational complexity* 26, 4 (2017), 765–833.
[29] Daniel Gottesman. 1997. Stabilizer codes and quantum error correction. *arXiv preprint quant-ph/9705052* (1997).
[30] Daniel Gottesman. 1998. The Heisenberg representation of quantum computers. *arXiv preprint quant-ph/9807006* (1998).
[31] William D Gropp, William Gropp, Ewing Lusk, and Anthony Skjellum. 1999. *Using MPI: portable parallel programming with the message-passing interface*. Vol. 1. MIT press.
[32] Lov K Grover. 1996. A fast quantum mechanical algorithm for database search. *arXiv preprint quant-ph/9605043* (1996).
[33] Thomas Häner and Damian S Steiger. 2017. 0.5 petabyte simulation of a 45-qubit quantum circuit. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. ACM, 33.
[34] Yipeng Huang and Margaret Martonosi. 2019. Statistical assertions for validating patterns and finding bugs in quantum programs". In *2019 ACM/IEEE 46th Annual International Symposium on Computer Architecture (ISCA)*. IEEE.
[35] D. A. Huffman. 1952. A Method for the Construction of Minimum-Redundancy Codes. *Proceedings of the IRE* 40, 9 (Sep. 1952), 1098–1101. https://doi.org/10.1109/JRPROC.1952.273898
[36] Lawrence Ibarria, Peter Lindstrom, Jarek Rossignac, and Andrzej Szymczak. 2003. Out-of-core compression and decompression of large n-dimensional scalar fields. *Computer Graphics Forum* 22, 3 (2003), 343–348.
[37] Ali JavadiAbhari, Shruti Patil, Daniel Kudrow, Jeff Heckey, Alexey Lvov, Frederic T Chong, and Margaret Martonosi. 2015. ScaffCC: Scalable compilation and analysis of quantum programs. *Parallel Comput.* 45 (2015), 2–17.
[38] James Jeffers and James Reinders. 2013. *Intel Xeon Phi coprocessor high performance programming*. Newnes.
[39] Richard Jozsa. 2001. Quantum factoring, discrete logarithms, and the hidden subgroup problem. *Computing in science & engineering* 3, 2 (2001), 34.
[40] Abhinav Kandala, Kristan Temme, Antonio D Córcoles, Antonio Mezzacapo, Jerry M Chow, and Jay M Gambetta. 2019. Error mitigation extends the computational reach of a noisy quantum processor. *Nature* 567, 7749 (2019), 491.
[41] Julian Kelly. 2018. A preview of Bristlecone, GoogleâĂŹs new quantum processor. *Google Research Blog* 5 (2018).
[42] Sriram Lakshminarasimhan, Neil Shah, Stephane Ethier, Seung-Hoe Ku, Choong-Seock Chang, Scott Klasky, Rob Latham, Rob Ross, and Nagiza F Samatova. 2013. Isabela for effective in situ compression of scientific data. *Concurrency and Computation: Practice and Experience* 25, 4 (2013), 524–540.
[43] Riling Li, Bujiao Wu, Mingsheng Ying, Xiaoming Sun, and Guangwen Yang. 2018. Quantum supremacy circuit simulation on Sunway taihuLight. *arXiv preprint arXiv:1804.04797* (2018).
[44] X. Liang, S. Di, D. Tao, S. Li, S. Li, H. Guo, Z. Chen, and F. Cappello. 2018. Error-Controlled Lossy Compression Optimized for High Compression Ratios of Scientific Datasets. In *2018 IEEE International Conference on Big Data (Big Data)*. 438–447. https://doi.org/10.1109/BigData.2018.8622520
[45] Peter Lindstrom. 2014. Fixed-rate compressed floating-point arrays. *IEEE Transactions on Visualization and Computer Graphics* 20, 12 (2014), 2674–2683.
[46] Peter Lindstrom and Martin Isenburg. 2006. Fast and efficient compression of floating-point data. *IEEE Transactions on Visualization and Computer Graphics* 12, 5 (2006), 1245–1250.
[47] T. Lu, Q. Liu, X. He, H. Luo, E. Suchyta, J. Choi, N. Podhorszki, S. Klasky, M. Wolf, T. Liu, and Z. Qiao. 2018. Understanding and Modeling Lossy Compression Schemes on HPC Scientific Data. In *2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. 348–357. https://doi.org/10.1109/IPDPS.2018.00044
[48] Igor L Markov, Aneeqa Fatima, Sergei V Isakov, and Sergio Boixo. 2018. Quantum supremacy is both closer and farther than it appears. *arXiv preprint arXiv:1807.10749* (2018).
[49] Igor L Markov and Yaoyun Shi. 2008. Simulating quantum computation by contracting tensor networks. *SIAM J. Comput.* 38, 3 (2008), 963–981.
[50] Alexander McCaskey, Eugene Dumitrescu, Mengsu Chen, Dmitry Lyakh, and Travis Humble. 2018. Validating quantum-classical programming models with tensor network simulations. *PloS one* 13, 12 (2018), e0206704.
[51] Michael A Nielsen and Isaac Chuang. 2002. Quantum computation and quantum information.
[52] Scott Parker, Vitali Morozov, Sudheer Chunduri, Kevin Harms, Chris Knight, and Kalyan Kumaran. 2017. *Early Evaluation of the Cray XC40 Xeon Phi system 'ThetaâĂŹat Argonne*. Technical Report. Argonne National Lab.(ANL), Argonne, IL (United States).
[53] Edwin Pednault, John A Gunnels, Giacomo Nannicini, Lior Horesh, Thomas Magerlein, Edgar Solomonik, and Robert Wisnieff. 2017. Breaking the 49-qubit barrier in the simulation of quantum circuits. *arXiv preprint arXiv:1710.05867* (2017).
[54] John Preskill. 2018. Quantum Computing in the NISQ era and beyond. *Quantum* 2 (2018), 79.
[55] Quantiki. 2019. A list of QC simulators. https://quantiki.org/wiki/list-qc-simulators. Online.
[56] Ramiro Sagastizabal, Xavier Bonet-Monroig, Malay Singh, MA Rol, CC Bultink, X Fu, CH Price, VP Ostroukh, N Muthusubramanian, A Bruno, et al. 2019. Error mitigation by symmetry verification on a variational quantum eigensolver. *arXiv preprint arXiv:1902.11258* (2019).
[57] Naoto Sasaki, Kento Sato, Toshio Endo, and Satoshi Matsuoka. 2015. Exploration of Lossy Compression for Application-Level Checkpoint/Restart. In *IPDPS 2015*. 914–922.
[58] Peter W Shor. 1999. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM review* 41, 2 (1999), 303–332.
[59] Mikhail Smelyanskiy, Nicolas PD Sawaya, and Alán Aspuru-Guzik. 2016. qHiPSTER: the quantum high performance software testing environment. *arXiv preprint arXiv:1601.07195* (2016).
[60] Avinash Sodani. 2015. Knights Landing (KNL): 2nd generation Intel® Xeon Phi processor. In *2015 IEEE Hot Chips 27 Symposium (HCS)*. IEEE, 1–24.
[61] Erich Strohmaier, Jack Dongarra, Horst Simon, Martin Meuer, and Hans Meuer. 2018. TOP500 list. https://www.top500.org/lists/2018/11/. (2018).
[62] Dingwen Tao, Sheng Di, Zizhong Chen, and Franck Cappello. 2017. Significantly improving lossy compression for scientific data sets based on multidimensional prediction and error-controlled quantization. In *Parallel and Distributed Processing Symposium (IPDPS), 2017 IEEE International*. IEEE, 1129–1139.
[63] Benjamin Villalonga, Dmitry Lyakh, Sergio Boixo, Hartmut Neven, Travis S Humble, Rupak Biswas, Eleanor G Rieffel, Alan Ho, and Salvatore Mandrà. 2019. Establishing the Quantum Supremacy Frontier with a 281 Pflop/s Simulation. *arXiv preprint arXiv:1905.00444* (2019).
[64] Jack Wells, Buddy Bland, Jeff Nichols, Jim Hack, Fernanda Foertter, Gaute Hagen, Thomas Maier, Moetasim Ashfaq, Bronson Messer, and Suzanne Parete-Koon. 2016. Announcing Supercomputer Summit. (6 2016).
[65] Ian H. Witten, Radford M. Neal, and John G. Cleary. 1987. Arithmetic Coding for Data Compression. *Commun. ACM* 30, 6 (June 1987), 520–540. https://doi.org/10.1145/214762.214771
[66] Dingwen Tao Zizhong Chen Franck Cappello Xin Liang, Sheng Di. 2018. Efficient Transformation Scheme for Lossy Data Compression with Point-wise Relative Error Bound. In *CLUSTER*. IEEE.
[67] J. Ziv and A. Lempel. 1977. A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory* 23, 3 (May 1977), 337–343. https://doi.org/10.1109/TIT.1977.1055714
[68] XiangYu Zou, Tao Lu, Wen Xia, Xuan Wang, Weizhe Zhang, Sheng Di, Dingwen Tao, and Franck Cappello. 2019. Accelerating Relative-error Bounded Lossy Compression for HPC datasets with Precomputation-Based Mechanisms. In *Proceedings of the 35th International Conference on Massive Storage Systems and Technology (MSST19)*. IEEE.
[69] Alwin Zulehner, Philipp Niemann, Rolf Drechsler, and Robert Wille. 2019. Accuracy and Compactness in Decision Diagrams for Quantum Computation. In *Design, Automation and Test in Europe*.
[70] Alwin Zulehner and Robert Wille. 2018. Advanced simulation of quantum computations. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (2018).
[71] Alwin Zulehner and Robert Wille. 2019. Matrix-Vector vs. Matrix-Matrix Multiplication: Potential in DD-based Simulation of Quantum Computations. *Quantum* 1, 1 (2019), 1–1.

Appendix: Artifact Description/Artifact Evaluation

SUMMARY OF THE EXPERIMENTS REPORTED

We ran our simulation on the Argonne's Theta supercomputer. Here's the list of the loaded modules.

Currently Loaded Modulefiles: 1) modules/3.2.11.1 2) intel/18.0.0.128 3) craype-network-aries 4) craype/2.5.15 5) cray-libsci/18.07.1 6) udreg/2.3.2-6.0.6.0_15.18__g5196236.ari 7) ugni/6.0.14-6.0.6.0_18.12__g777707d.ari 8) pmi/5.0.14 9) dmapp/7.1.1-6.0.6.0_51.37__g5a674e0.ari 10) gni-headers/5.0.12-6.0.6.0_3.26__g527b6e1.ari 11) xpmem/2.2.14-6.0.6.1_5.14__g34333c9.ari 12) job/2.2.3-6.0.6.0_9.47__g6c4e934.ari 13) dvs/2.7_2.2.100-6.0.6.1_12.6__g542256c 14) alps/6.6.1-6.0.6.1_4.1__ga6396bb.ari 15) rca/2.2.18-6.0.6.0_19.14__g2aa4f39.ari 16) atp/2.1.3 17) perftools-base/7.0.4 18) PrgEnv-intel/6.0.4 19) craype-mic-knl 20) cray-mpich/7.7.3 21) nompirun/nompirun 22) darshan/3.1.5 23) trackdeps 24) xalt

ARTIFACT AVAILABILITY

Software Artifact Availability: All author-created software artifacts are maintained in a public repository under an OSI-approved license.

Hardware Artifact Availability: There are no author-created hardware artifacts.

Data Artifact Availability: All author-created data artifacts are maintained in a public repository under an OSI-approved license.

Proprietary Artifacts: None of the associated artifacts, author-created or otherwise, are proprietary.

List of URLs and/or DOIs where artifacts are available:

`https://github.com/ryanxw/compressed\_qcsim`
`https://github.com/disheng222/SZ`
`https://github.com/intel/Intel-QS`

BASELINE EXPERIMENTAL SETUP, AND MODIFICATIONS MADE FOR THE PAPER

Relevant hardware details: Cray XC40 with Intel Xeon Phi 7230 processors

Operating systems and versions: Linux version 4.4.162-94.72-default (geeko@buildhost) (gcc version 4.8.5 (SUSE Linux))

Compilers and versions: icpc (ICC) 18.0.2 20180210

Libraries and versions: cray-mpich/7.7.3

Key algorithms: SZ lossy compression and Intel-QS simulator

Paper Modifications: We design a new lossy compression algorithm for compressing quantum circuit simulation data, and integrate our algorithm into Intel-QS simulator.

Output from scripts that gathers execution environment information.

```
LESSKEY=/etc/lesskey.bin
MODULE_VERSION_STACK=3.2.11.1
KSH_AUTOLOAD=1
MKLROOT=/opt/intel/compilers_and_libraries_2018.2.19
↪ 9/linux/mkl
PE_LIBSCI_VOLATILE_PRGENV=CRAY GNU INTEL
PE_SMA_DEFAULT_PKGCONFIG_VARIABLES=PE_SMA_COMPFLAG_@
↪ prgenv@
PE_TPSL_64_DEFAULT_GENCOMPS_INTEL_mic_knl=160
MANPATH=/opt/intel/man/common:/opt/intel/compilers_a
↪ nd_libraries_2018.2.199/linux/mpi/man:/opt/intel
↪ /documentation_2018/en/debugger//gdb-ia/man/:/op
↪ t/intel/documentation_2018/en/debugger//gdb-igfx
↪ /man/:/opt/cray/pe/mpt/7.7.3/gni/man/mpich:/opt/
↪ cray/pe/perftools/7.0.4/man:/opt/cray/pe/papi/5.
↪ 6.0.4/share/pdoc/man:/opt/cray/pe/atp/2.1.3/man:
↪ /opt/cray/alps/6.6.1-6.0.6.1_4.1__ga6396bb.ari/m
↪ an:/opt/cray/job/2.2.3-6.0.6.0_9.47__g6c4e934.ar
↪ i/man:/opt/cray/pe/pmi/5.0.14/man:/opt/cray/pe/l
↪ ibsci/18.07.1/man:/opt/cray/pe/man/csmlversion:/
↪ opt/cray/pe/craype/2.5.15/man:/opt/intel/man/com
↪ mon:/opt/intel/compilers_and_libraries_2018.0.12
↪ 8/linux/mpi/man:/opt/intel/documentation_2017/en
↪ /debugger//gdb-ia/man/:/opt/intel/documentation_
↪ 2017/en/debugger//gdb-mic/man/:/opt/intel/docume
↪ ntation_2017/en/debugger//gdb-igfx/man/:/opt/cra
↪ y/pe/modules/3.2.11.1/share/man:/usr/local/man:/
↪ usr/share/man:/usr/man:/opt/cray/share/man:/opt/
↪ cray/pe/man:
DB2INSTANCE=db2cat
NNTPSERVER=news
PE_PAPI_DEFAULT_ACCEL_FAMILY_LIBS_nvidia=,-lcupti,-l
↪ cudart,-lcuda
PE_PETSC_DEFAULT_GENCOMPILERS_CRAY_sandybridge=8.6
PE_PETSC_DEFAULT_GENCOMPS_CRAY_skylake=86
PE_TPSL_DEFAULT_GENCOMPS_INTEL_x86_skylake=160
PE_CXX_PKGCONFIG_LIBS=mpichcxx
XDG_SESSION_ID=1449
HOSTNAME=thetalogin6
IBM_DB_LIB=/dbhome/db2cat/sqllib/lib
CRAY_UDREG_INCLUDE_OPTS=-I/opt/cray/udreg/2.3.2-6.0.
↪ 6.0_15.18__g5196236.ari/include
PE_FFTW_DEFAULT_TARGET_mic_knl=mic_knl
PE_PETSC_DEFAULT_GENCOMPILERS_INTEL_mic_knl=16.0
PE_TPSL_64_DEFAULT_GENCOMPS_INTEL_interlagos=160
PE_TRILINOS_DEFAULT_GENCOMPS_CRAY_x86_64=87
XKEYSYMDB=/usr/X11R6/lib/X11/XKeysymDB
CRAY_SITE_LIST_DIR=/etc/opt/cray/pe/modules
```

```
LIBRARYMODULES=acml:alps:cray-dwarf:cray-fftw:cray-g
↪ a:cray-hdf5:cray-hdf5-parallel:cray-libsci:cray-
↪ libsci_acc:cray-mpich:cray-mpich2:cray-mpich-abi
↪ :cray-netcdf:cray-netcdf-hdf5parallel:cray-paral
↪ lel-netcdf:cray-petsc:cray-petsc-complex:cray-sh
↪ mem:cray-tpsl:cray-trilinos:cudatoolkit:fftw:ga:
↪ hdf5:hdf5-parallel:iobuf:libfast:netcdf:netcdf-h
↪ df5parallel:ntk:onesided:papi:petsc:petsc-comple
↪ x:pmi:tpsl:trilinos:xt-libsci:xt-mpich2:xt-mpt:x
↪ t-papi
PE_NETCDF_DEFAULT_VOLATILE_PKGCONFIG_PATH=/opt/cray/
↪ pe/netcdf/4.6.1.3/@PRGENV@/@PE_NETCDF_DEFAULT_GE
↪ NCOMPS@/lib/pkgconfig
PE_PARALLEL_NETCDF_DEFAULT_VOLATILE_PKGCONFIG_PATH=/
↪ opt/cray/pe/parallel-netcdf/1.8.1.3/@PRGENV@/@PE
↪ _PARALLEL_NETCDF_DEFAULT_GENCOMPS@/lib/pkgconfig
PE_SMA_DEFAULT_COMPFLAG_GNU=-fcray-pointer
PE_TRILINOS_DEFAULT_VOLATILE_PKGCONFIG_PATH=/opt/cra
↪ y/pe/trilinos/12.12.1.1/@PRGENV@/@PE_TRILINOS_DE
↪ FAULT_GENCOMPS@/@PE_TRILINOS_DEFAULT_TARGET@/lib
↪ /pkgconfig
INTEL_LICENSE_FILE=/opt/intel/compilers_and_librarie
↪ s_2018.0.128/linux/licenses:/opt/intel/licenses:
↪ /intel/licenses:/opt/intel/compilers_and_librari
↪ es_2018.2.199/linux/licenses:/opt/intel/licenses
↪ :/home/USER/intel/licenses
IPPROOT=/opt/intel/compilers_and_libraries_2018.2.19
↪ 9/linux/ipp
PE_ENV=INTEL
PE_HDF5_DEFAULT_GENCOMPILERS_GNU=8.2 7.1 6.1 5.3 4.9
PE_MPICH_ALTERNATE_LIBS_dpm=_dpm
PE_SMA_DEFAULT_COMPFLAG=
PE_TPSL_64_DEFAULT_GENCOMPILERS_CRAY_x86_64=8.6
SHELL=/bin/bash
TERM=xterm-256color
HOST=thetalogin6
PE_TPSL_DEFAULT_GENCOMPS_CRAY_x86_skylake=86
PKGCONFIG_ENABLED=1
HISTSIZE=1000
PROFILEREAD=true
INTEL_MINOR_VERSION=0.128
PE_PETSC_DEFAULT_GENCOMPS_CRAY_sandybridge=86
PE_TPSL_DEFAULT_GENCOMPILERS_GNU_x86_skylake=8.2 7.1
↪ 6.1
SSH_CLIENT=128.135.98.18 64766 22
CRAYPE_DIR=/opt/cray/pe/craype/2.5.15
CRAY_UGNI_POST_LINK_OPTS=-L/opt/cray/ugni/6.0.14-6.0
↪ .6.0_18.12__g777707d.ari/lib64
CRAY_XPMEM_POST_LINK_OPTS=-L/opt/cray/xpmem/2.2.14-6
↪ .0.6.1_5.14__g34333c9.ari/lib64
GDBSERVER_MIC=/opt/intel/debugger_2018/gdb/targets/i
↪ ntel64/x200/bin/gdbserver
PE_NETCDF_DEFAULT_VOLATILE_PRGENV=GNU
PE_PARALLEL_NETCDF_DEFAULT_VOLATILE_PRGENV=GNU
PE_PETSC_DEFAULT_GENCOMPS_GNU_haswell=71 53 49
PE_PETSC_DEFAULT_GENCOMPS_INTEL_haswell=160
PE_TPSL_64_DEFAULT_GENCOMPS_INTEL_x86_skylake=160
PE_TPSL_DEFAULT_GENCOMPS_GNU_sandybridge=82 71 53 49
PE_TPSL_DEFAULT_REQUIRED_PRODUCTS=PE_MPICH:PE_LIBSCI
PE_TRILINOS_DEFAULT_VOLATILE_PRGENV=CRAY GNU INTEL
LIBRARY_PATH=/opt/intel/compilers_and_libraries_2018
↪ .2.199/linux/ipp/lib/intel64:/opt/intel/compiler
↪ s_and_libraries_2018.2.199/linux/compiler/lib/in
↪ tel64_lin:/opt/intel/compilers_and_libraries_201
↪ 8.2.199/linux/mkl/lib/intel64_lin:/opt/intel/com
↪ pilers_and_libraries_2018.2.199/linux/tbb/lib/in
↪ tel64/gcc4.7:/opt/intel/compilers_and_libraries_
↪ 2018.2.199/linux/tbb/lib/intel64/gcc4.7:/opt/int
↪ el/compilers_and_libraries_2018.2.199/linux/daal
↪ /lib/intel64_lin:/opt/intel/compilers_and_librar
↪ ies_2018.0.128/linux/ipp/lib/intel64:/opt/intel/
↪ compilers_and_libraries_2018.0.128/linux/compile
↪ r/lib/intel64:/opt/intel/compilers_and_libraries
↪ _2018.0.128/linux/mkl/lib/intel64:/opt/intel/com
↪ pilers_and_libraries_2018.0.128/linux/tbb/lib/in
↪ tel64/gcc4.4:/opt/intel/compilers_and_libraries_
↪ 2018.0.128/linux/daal/lib/intel64_lin:/opt/intel
↪ /compilers_and_libraries_2018.0.128/linux/daal/.
↪ ./compiler/lib/intel64_lin
PE_FFTW_DEFAULT_VOLATILE_PKGCONFIG_PATH=/opt/cray/pe
↪ /fftw/3.3.8.1/@PE_FFTW_DEFAULT_TARGET@/lib/pkgco
↪ nfig
PE_HDF5_DEFAULT_VOLATILE_PRGENV=GNU
PE_HDF5_PARALLEL_DEFAULT_VOLATILE_PKGCONFIG_PATH=/op
↪ t/cray/pe/hdf5-parallel/1.10.2.0/@PRGENV@/@PE_HD
↪ F5_PARALLEL_DEFAULT_GENCOMPS@/lib/pkgconfig
PE_NETCDF_HDF5PARALLEL_DEFAULT_VOLATILE_PKGCONFIG_PA
↪ TH=/opt/cray/pe/netcdf-hdf5parallel/4.6.1.3/@PRG
↪ ENV@/@PE_NETCDF_HDF5PARALLEL_DEFAULT_GENCOMPS@/l
↪ ib/pkgconfig
PE_PETSC_DEFAULT_GENCOMPS_CRAY_interlagos=86
CRAY_MPICH2_DIR=/opt/cray/pe/mpt/7.7.3/gni/mpich-int
↪ el/16.0
INTEL_PATH=/opt/intel/compilers_and_libraries_2018.0
↪ .128
PE_GA_DEFAULT_VOLATILE_PRGENV=GNU
PE_LIBSCI_DEFAULT_GENCOMPS_GNU_x86_64=71 61 51 49
PE_TPSL_64_DEFAULT_GENCOMPILERS_CRAY_interlagos=8.6
PE_TPSL_DEFAULT_GENCOMPS_CRAY_mic_knl=86
LD_PRELOAD=/soft/buildtools/trackdeps/$LIB/trackdeps
↪ .so
MORE=-sl
IBM_DB_DIR=/dbhome/db2cat/sqllib
FPATH=:/opt/cray/pe/modules/3.2.11.1/init/sh_funcs/n
↪ o_redirect
PERFTOOLS_VERSION=7.0.4
PE_MPICH_DEFAULT_GENCOMPILERS_GNU=7.1 5.1 4.9
PE_PKGCONFIG_PRODUCTS=PE_MPICH:PE_LIBSCI
PE_TPSL_DEFAULT_GENCOMPS_INTEL_x86_64=160
PE_MPICH_GENCOMPS_GNU=71 51 49
DARSHAN_PRELOAD=/soft/perftools/darshan/darshan-3.1.
↪ 5/lib/libdarshan.so
```

```
PE_PAPI_DEFAULT_ACCEL_LIBS_nvidia35=,-lcupti,-lcudart,-lcuda
PE_PETSC_DEFAULT_REQUIRED_PRODUCTS=PE_MPICH:PE_LIBSCI:PE_HDF5_PARALLEL:PE_TPSL
PE_TPSL_64_DEFAULT_GENCOMPS_CRAY_haswell=86
PE_TPSL_64_DEFAULT_VOLATILE_PKGCONFIG_PATH=/opt/cray/pe/tpsl/18.06.1/@PRGENV@64/@PE_TPSL_64_DEFAULT_GENCOMPS@/@PE_TPSL_64_DEFAULT_TARGET@/lib/pkgconfig
PE_CRAY_DEFAULT_FIXED_PKGCONFIG_PATH=/opt/cray/pe/parallel-netcdf/1.8.1.3/CRAY/8.6/lib/pkgconfig:/opt/cray/pe/netcdf-hdf5parallel/4.6.1.3/CRAY/8.6/lib/pkgconfig:/opt/cray/pe/netcdf/4.6.1.3/CRAY/8.6/lib/pkgconfig:/opt/cray/pe/hdf5-parallel/1.10.2.0/CRAY/8.6/lib/pkgconfig:/opt/cray/pe/hdf5/1.10.2.0/CRAY/8.6/lib/pkgconfig:/opt/cray/pe/ga/5.3.0.8/CRAY/8.6/lib/pkgconfig
PE_TRILINOS_DEFAULT_GENCOMPILERS_CRAY_x86_64=8.7
SSH_TTY=/dev/pts/30
MIC_LD_LIBRARY_PATH=/opt/intel/compilers_and_libraries_2018.0.128/linux/mpi/mic/lib:/opt/intel/compilers_and_libraries_2018.0.128/linux/compiler/lib/mic:/opt/intel/compilers_and_libraries_2018.0.128/linux/ipp/lib/mic:/opt/intel/compilers_and_libraries_2018.0.128/linux/compiler/lib/mic:/opt/intel/compilers_and_libraries_2018.0.128/linux/mkl/lib/mic:/opt/intel/compilers_and_libraries_2018.0.128/linux/tbb/lib/mic
PE_LIBSCI_DEFAULT_OMP_REQUIRES_openmp=_mp
PE_PETSC_DEFAULT_GENCOMPS_CRAY_x86_64=86
PE_TPSL_64_DEFAULT_GENCOMPILERS_CRAY_sandybridge=8.6
PE_FORTRAN_PKGCONFIG_LIBS=mpichf90
CRAYPAT_ALPS_COMPONENT=/opt/cray/pe/perftools/7.0.4/sbin/pat_alps
CRAYPAT_LD_LIBRARY_PATH=/opt/cray/pe/gcc-libs:/opt/cray/gcc-libs:/opt/cray/pe/perftools/7.0.4/lib64
PE_SMA_DEFAULT_VOLATILE_PKGCONFIG_PATH=/opt/cray/pe/mpt/7.7.3/gni/sma@PE_SMA_DEFAULT_DIR_DEFAULT64@/lib64/pkgconfig
ALLINEA_QUEUE_DLL=/opt/cray/pe/mpt/7.7.3/gni/mpich-intel/16.0/lib/libtvmpich.so.3.0.1
PE_TRILINOS_DEFAULT_GENCOMPS_INTEL_x86_64=160
CRAY_MPICH_BASEDIR=/opt/cray/pe/mpt/7.7.3/gni
USER=USER
JRE_HOME=/usr/lib64/jvm/java/jre
PE_HDF5_PARALLEL_DEFAULT_GENCOMPILERS_GNU=8.2 7.1 6.1 5.3 4.9
PE_NETCDF_HDF5PARALLEL_DEFAULT_GENCOMPILERS_GNU=8.2 7.1 6.1 5.3 4.9
PE_TPSL_64_DEFAULT_GENCOMPS_CRAY_x86_skylake=86
PE_TPSL_64_DEFAULT_GENCOMPS_INTEL_haswell=160
LD_LIBRARY_PATH=/opt/intel/compilers_and_libraries_2018.2.199/linux/compiler/lib/intel64:/opt/intel/compilers_and_libraries_2018.2.199/linux/compiler/lib/intel64_lin:/opt/intel/compilers_and_libraries_2018.2.199/linux/mpi/intel64/lib:/opt/intel/compilers_and_libraries_2018.2.199/linux/mpi/mic/lib:/opt/intel/compilers_and_libraries_2018.2.199/linux/ipp/lib/intel64:/opt/intel/compilers_and_libraries_2018.2.199/linux/compiler/lib/intel64_lin:/opt/intel/compilers_and_libraries_2018.2.199/linux/mkl/lib/intel64_lin:/opt/intel/compilers_and_libraries_2018.2.199/linux/tbb/lib/intel64/gcc4.7:/opt/intel/compilers_and_libraries_2018.2.199/linux/tbb/lib/intel64/gcc4.7:/opt/intel/debugger_2018/iga/lib:/opt/intel/debugger_2018/libipt/intel64/lib:/opt/intel/compilers_and_libraries_2018.2.199/linux/daal/lib/intel64_lin:/soft/perftools/darshan/darshan-3.1.5/lib:/opt/cray/pe/papi/5.6.0.4/lib64:/opt/cray/job/2.2.3-6.0.6.0_9.47__g6c4e934.ari/lib64:/opt/intel/compilers_and_libraries_2018.0.128/linux/compiler/lib/intel64:/opt/intel/compilers_and_libraries_2018.0.128/linux/compiler/lib/intel64_lin:/opt/intel/compilers_and_libraries_2018.0.128/linux/mpi/intel64/lib:/opt/intel/compilers_and_libraries_2018.0.128/linux/mpi/mic/lib:/opt/intel/compilers_and_libraries_2018.0.128/linux/ipp/lib/intel64:/opt/intel/compilers_and_libraries_2018.0.128/linux/compiler/lib/intel64:/opt/intel/compilers_and_libraries_2018.0.128/linux/mkl/lib/intel64:/opt/intel/compilers_and_libraries_2018.0.128/linux/tbb/lib/intel64/gcc4.4:/opt/intel/debugger_2017/libipt/intel64/lib:/opt/intel/compilers_and_libraries_2018.0.128/linux/daal/lib/intel64_lin:/opt/intel/compilers_and_libraries_2018.0.128/linux/daal/../compiler/lib/intel64_lin:/dbhome/db2cat/sqllib/lib64:/dbhome/db2cat/sqllib/lib64/gskit:/dbhome/db2cat/sqllib/lib32
```

```
LS_COLORS=rs=0:di=01;34:ln=01;36:mh=00:pi=40;33:so=0
↪ 1;35:do=01;35:bd=40;33;01:cd=40;33;01:or=40;31;0
↪ 1:mi=00:su=37;41:sg=30;43:ca=30;41:tw=30;42:ow=3
↪ 4;42:st=37;44:ex=01;32:*.tar=01;31:*.tgz=01;31:*
↪ .arc=01;31:*.arj=01;31:*.taz=01;31:*.lha=01;31:*
↪ .lz4=01;31:*.lzh=01;31:*.lzma=01;31:*.tlz=01;31:
↪ *.txz=01;31:*.tzo=01;31:*.t7z=01;31:*.zip=01;31:
↪ *.z=01;31:*.Z=01;31:*.dz=01;31:*.gz=01;31:*.lrz=
↪ 01;31:*.lz=01;31:*.lzo=01;31:*.xz=01;31:*.bz2=01
↪ ;31:*.bz=01;31:*.tbz=01;31:*.tbz2=01;31:*.tz=01;
↪ 31:*.deb=01;31:*.rpm=01;31:*.jar=01;31:*.war=01;
↪ 31:*.ear=01;31:*.sar=01;31:*.rar=01;31:*.alz=01;
↪ 31:*.ace=01;31:*.zoo=01;31:*.cpio=01;31:*.7z=01;
↪ 31:*.rz=01;31:*.cab=01;31:*.jpg=01;35:*.jpeg=01;
↪ 35:*.gif=01;35:*.bmp=01;35:*.pbm=01;35:*.pgm=01;
↪ 35:*.ppm=01;35:*.tga=01;35:*.xbm=01;35:*.xpm=01;
↪ 35:*.tif=01;35:*.tiff=01;35:*.png=01;35:*.svg=01
↪ ;35:*.svgz=01;35:*.mng=01;35:*.pcx=01;35:*.mov=0
↪ 1;35:*.mpg=01;35:*.mpeg=01;35:*.m2v=01;35:*.mkv=
↪ 01;35:*.webm=01;35:*.ogm=01;35:*.mp4=01;35:*.m4v
↪ =01;35:*.mp4v=01;35:*.vob=01;35:*.qt=01;35:*.nuv
↪ =01;35:*.wmv=01;35:*.asf=01;35:*.rm=01;35:*.rmvb
↪ =01;35:*.flc=01;35:*.avi=01;35:*.fli=01;35:*.flv
↪ =01;35:*.gl=01;35:*.dl=01;35:*.xcf=01;35:*.xwd=0
↪ 1;35:*.yuv=01;35:*.cgm=01;35:*.emf=01;35:*.ogv=0
↪ 1;35:*.ogx=01;35:*.aac=00;36:*.au=00;36:*.flac=0
↪ 0;36:*.m4a=00;36:*.mid=00;36:*.midi=00;36:*.mka=
↪ 00;36:*.mp3=00;36:*.mpc=00;36:*.ogg=00;36:*.ra=0
↪ 0;36:*.wav=00;36:*.oga=00;36:*.opus=00;36:*.spx=
↪ 00;36:*.xspf=00;36:
PE_FFTW_DEFAULT_TARGET_interlagos=interlagos
PE_LIBSCI_DEFAULT_VOLATILE_PRGENV=CRAY GNU INTEL
PE_PETSC_DEFAULT_GENCOMPILERS_INTEL_interlagos=16.0
PE_TPSL_64_DEFAULT_GENCOMPILERS_INTEL_mic_knl=16.0
PE_TPSL_DEFAULT_GENCOMPS_CRAY_x86_64=86
PE_TRILINOS_DEFAULT_GENCOMPILERS_GNU_x86_64=8.2 7.3
↪ 5.1 4.9
PE_TRILINOS_DEFAULT_GENCOMPILERS_INTEL_x86_64=16.0
CRAY_RCA_POST_LINK_OPTS=-L/opt/cray/rca/2.2.18-6.0.6
↪ .0_19.14__g2aa4f39.ari/lib64
↪ -lrca
PE_LIBSCI_PKGCONFIG_VARIABLES=PE_LIBSCI_OMP_REQUIRES
↪ _@openmp@:PE_SCI_EXT_LIBPATH:PE_SCI_EXT_LIBNAME
PE_PETSC_DEFAULT_VOLATILE_PRGENV=CRAY CRAY64 GNU
↪ GNU64 INTEL INTEL64
PE_PKGCONFIG_LIBS=darshan-runtime:mpich:AtpSigHandle
↪ r:cray-rca:libsci_mpi:libsci
PE_TPSL_64_DEFAULT_GENCOMPILERS_GNU_sandybridge=8.2
↪ 7.1 5.3 4.9
PE_TPSL_64_DEFAULT_GENCOMPILERS_INTEL_haswell=16.0
PE_MPICH_FIXED_PRGENV=INTEL
PSTLROOT=/opt/intel/compilers_and_libraries_2018.2.1
↪ 99/linux/pstl
XNLSPATH=/usr/share/X11/nls
PE_PETSC_DEFAULT_GENCOMPILERS_CRAY_mic_knl=8.6
PE_PETSC_DEFAULT_GENCOMPILERS_CRAY_x86_64=8.6
PE_PETSC_DEFAULT_GENCOMPILERS_INTEL_skylake=16.0
PE_PETSC_DEFAULT_GENCOMPS_GNU_interlagos=71 53 49
PE_PETSC_DEFAULT_GENCOMPS_GNU_sandybridge=71 53 49
PE_PETSC_DEFAULT_GENCOMPS_INTEL_interlagos=160
PE_PETSC_DEFAULT_GENCOMPS_INTEL_sandybridge=160
PE_TPSL_DEFAULT_GENCOMPS_GNU_haswell=82 71 53 49
INTEL_VERSION=18.0.0.128
MIC_LIBRARY_PATH=/opt/intel/compilers_and_libraries_
↪ 2018.0.128/linux/mpi/mic/lib:/opt/intel/compiler
↪ s_and_libraries_2018.0.128/linux/compiler/lib/mi
↪ c:/opt/intel/compilers_and_libraries_2018.0.128/
↪ linux/tbb/lib/mic
MPICH_ABORT_ON_ERROR=1
PE_LIBSCI_DEFAULT_GENCOMPS_CRAY_x86_64=86
PE_PAPI_DEFAULT_PKGCONFIG_VARIABLES=PE_PAPI_ACCEL_LI
↪ BS_@accelerator@
↪ PE_PAPI_ACCEL_FAMILY_LIBS_@accelerator_family@
PE_PETSC_DEFAULT_GENCOMPILERS_CRAY_haswell=8.6
PE_PETSC_DEFAULT_GENCOMPS_GNU_mic_knl=53
PE_PETSC_DEFAULT_GENCOMPS_INTEL_mic_knl=160
PE_TPSL_64_DEFAULT_GENCOMPILERS_GNU_interlagos=8.2
↪ 7.1 5.3 4.9
PE_TPSL_64_DEFAULT_GENCOMPS_INTEL_sandybridge=160
MPICH_DIR=/opt/cray/pe/mpt/7.7.3/gni/mpich-intel/16.0
HOSTTYPE=x86_64
ATP_POST_LINK_OPTS=-Wl,-L/opt/cray/pe/atp/2.1.3/libA
↪ pp/
CPATH=/opt/intel/compilers_and_libraries_2018.2.199/
↪ linux/ipp/include:/opt/intel/compilers_and_libra
↪ ries_2018.2.199/linux/mkl/include:/opt/intel/com
↪ pilers_and_libraries_2018.2.199/linux/pstl/inclu
↪ de:/opt/intel/compilers_and_libraries_2018.2.199
↪ /linux/tbb/include:/opt/intel/compilers_and_libr
↪ aries_2018.2.199/linux/tbb/include:/opt/intel/co
↪ mpilers_and_libraries_2018.2.199/linux/daal/incl
↪ ude:/opt/intel/compilers_and_libraries_2018.0.12
↪ 8/linux/ipp/include:/opt/intel/compilers_and_lib
↪ raries_2018.0.128/linux/mkl/include:/opt/intel/c
↪ ompilers_and_libraries_2018.0.128/linux/tbb/incl
↪ ude:/opt/intel/compilers_and_libraries_2018.0.12
↪ 8/linux/daal/include
PE_FFTW_DEFAULT_REQUIRED_PRODUCTS=PE_MPICH
PE_FFTW_DEFAULT_TARGET_sandybridge=sandybridge
PE_HDF5_PARALLEL_DEFAULT_REQUIRED_PRODUCTS=PE_MPICH
PE_NETCDF_HDF5PARALLEL_DEFAULT_REQUIRED_PRODUCTS=PE_
↪ HDF5_PARALLEL
PE_PETSC_DEFAULT_GENCOMPILERS_INTEL_sandybridge=16.0
PE_TPSL_64_DEFAULT_GENCOMPILERS_CRAY_haswell=8.6
PE_MPICH_FORTRAN_PKGCONFIG_LIBS=mpichf90
IBM_DB_HOME=/dbhome/db2cat/sqllib
RCLOCAL_PRGENV=true
PE_PETSC_DEFAULT_GENCOMPILERS_GNU_mic_knl=5.3
FROM_HEADER=
CHPL_CG_CPP_LINES=1
OFFLOAD_INIT=on_start
PE_LIBSCI_DEFAULT_GENCOMPILERS_INTEL_x86_64=16.0
```

```
PE_LIBSCI_GENCOMPS_INTEL_x86_64=160
PE_PRODUCT_LIST=CRAYPE_MIC-KNL:CRAY_RCA:CRAY_ALPS:DV
↪  S:CRAY_XPMEM:CRAY_DMAPP:CRAY_PMI:CRAY_UGNI:CRAY_
↪  UDREG:CRAY_LIBSCI:CRAYPE:INTEL:PERFTOOLS:CRAYPAT
PE_TPSL_DEFAULT_GENCOMPILERS_CRAY_x86_64=8.6
PE_TPSL_DEFAULT_GENCOMPS_GNU_interlagos=82 71 53 49
PAGER=less
PE_PETSC_DEFAULT_GENCOMPILERS_GNU_x86_64=7.1 5.3 4.9
PE_TPSL_DEFAULT_GENCOMPS_GNU_x86_skylake=82 71 61
CRAY_MPICH_ROOTDIR=/opt/cray/pe/mpt/7.7.3
CSHEDIT=emacs
PE_LIBSCI_GENCOMPILERS_GNU_x86_64=7.1 6.1 5.1 4.9
PE_PETSC_DEFAULT_GENCOMPS_GNU_skylake=61
PE_PETSC_DEFAULT_GENCOMPS_INTEL_skylake=160
PE_TPSL_64_DEFAULT_GENCOMPILERS_INTEL_x86_64=16.0
PE_MPICH_GENCOMPILERS_CRAY=8.6
PE_MPICH_MODULE_NAME=cray-mpich
XDG_CONFIG_DIRS=/etc/xdg
CRAYPAT_ROOT=/opt/cray/pe/perftools/7.0.4
INTEL_MAJOR_VERSION=18.0
PE_LIBSCI_DEFAULT_GENCOMPILERS_CRAY_x86_64=8.6
PE_LIBSCI_GENCOMPS_CRAY_x86_64=86
PE_MPICH_DEFAULT_VOLATILE_PRGENV=CRAY GNU
PE_MPICH_TARGET_VAR_nvidia20=-lcudart
PE_TPSL_64_DEFAULT_REQUIRED_PRODUCTS=PE_MPICH:PE_LIB
↪  SCI
PE_TPSL_DEFAULT_GENCOMPS_CRAY_haswell=86
PE_TPSL_DEFAULT_GENCOMPS_CRAY_sandybridge=86
MINICOM=-c on
LIBGL_DEBUG=quiet
USERMODULES=acml:alps:apprentice:apprentice2:atp:blc
↪  r:cce:chapel:cray-ccdb:cray-fftw:cray-ga:cray-hd
↪  f5:cray-hdf5-parallel:cray-lgdb:cray-libsci:cray
↪  -libsci_acc:cray-mpich:cray-mpich2:cray-mpich-com
↪  pat:cray-netcdf:cray-netcdf-hdf5parallel:cray-pa
↪  rallel-netcdf:craypat:craype:cray-petsc:cray-pet
↪  sc-complex:craypkg-gen:cray-shmem:cray-snplaunch
↪  er:cray-tpsl:cray-trilinos:cudatoolkit:ddt:fftw:
↪  ga:gcc:hdf5:hdf5-parallel:intel:iobuf:java:lgdb:
↪  libfast:libsci_acc:mpich1:netcdf:netcdf-hdf5para
↪  llel:netcdf-nofsync:netcdf-nofsync-hdf5parallel:
↪  ntk:onesided:papi:parallel-netcdf:pathscale:perf
↪  tools:perftools-lite:petsc:petsc-complex:pgi:pmi
↪  :PrgEnv-cray:PrgEnv-gnu:PrgEnv-intel:PrgEnv-path
↪  scale:PrgEnv-pgi:stat:totalview:tpsl:trilinos:xt
↪  -asyncpe:xt-craypat:xt-lgdb:xt-libsci:xt-mpich2:x
↪  t-mpt:xt-papi:xt-shmem:xt-totalview
CRAY_DMAPP_INCLUDE_OPTS=-I/opt/cray/dmapp/7.1.1-6.0.
↪  6.0_51.37__g5a674e0.ari/include
↪  -I/opt/cray/gni-headers/5.0.12-6.0.6.0_3.26__g52
↪  7b6e1.ari/include
CRAY_LIBSCI_BASE_DIR=/opt/cray/pe/libsci/18.07.1
CRAY_LIBSCI_DIR=/opt/cray/pe/libsci/18.07.1
DVS_VERSION=0.9.0
NLSPATH=/opt/intel/compilers_and_libraries_2018.2.19
↪  9/linux/compiler/lib/intel64/locale/%l_%t/%N:/op
↪  t/intel/compilers_and_libraries_2018.2.199/linux
↪  /mkl/lib/intel64_lin/locale/%l_%t/%N:/opt/intel/
↪  debugger_2018/gdb/intel64/share/locale/%l_%t/%N:
↪  /opt/intel/compilers_and_libraries_2018.0.128/li
↪  nux/compiler/lib/intel64/locale/%l_%t/%N:/opt/in
↪  tel/compilers_and_libraries_2018.0.128/linux/mkl
↪  /lib/intel64/locale/%l_%t/%N:/opt/intel/debugger
↪  _2017/gdb/intel64_mic/share/locale/%l_%t/%N:/opt
↪  /intel/debugger_2017/gdb/intel64/share/locale/%l
↪  _%t/%N
PE_LIBSCI_PKGCONFIG_LIBS=libsci_mpi:libsci
PE_NETCDF_DEFAULT_GENCOMPS_GNU=
PE_PARALLEL_NETCDF_DEFAULT_GENCOMPS_GNU=51 49
PE_TPSL_64_DEFAULT_GENCOMPS_GNU_mic_knl=71 53
PE_TPSL_64_DEFAULT_GENCOMPS_GNU_x86_64=82 71 53 49
PATH=/home/USER/anaconda3/bin:/opt/intel/compilers_a
↪  nd_libraries_2018.2.199/linux/bin/intel64:/opt/i
↪  ntel/compilers_and_libraries_2018.2.199/linux/mp
↪  i/intel64/bin:/opt/xalt/bin:/soft/buildtools/tra
↪  ckdeps/bin:/soft/perftools/darshan/darshan-3.1.5
↪  /bin:/opt/modulefiles/nompirun/bin:/opt/cray/pe/
↪  mpt/7.7.3/gni/bin:/opt/cray/pe/perftools/7.0.4/b
↪  in:/opt/cray/pe/papi/5.6.0.4/bin:/opt/cray/rca/2
↪  .2.18-6.0.6.0_19.14__g2aa4f39.ari/bin:/opt/cray/
↪  alps/6.6.1-6.0.6.1_4.1__ga6396bb.ari/sbin:/opt/c
↪  ray/job/2.2.3-6.0.6.0_9.47__g6c4e934.ari/bin:/op
↪  t/cray/pe/craype/2.5.15/bin:/opt/intel/compilers
↪  _and_libraries_2018.0.128/linux/bin/intel64:/opt
↪  /intel/compilers_and_libraries_2018.0.128/linux/
↪  mpi/intel64/bin:/opt/intel/debugger_2017/gdb/int
↪  el64_mic/bin:/opt/cray/pe/modules/3.2.11.1/bin:/
↪  usr/local/bin:/usr/bin:/bin:/usr/bin/X11:/usr/ga
↪  mes:/dbhome/db2cat/sqllib/bin:/dbhome/db2cat/sql
↪  lib/adm:/dbhome/db2cat/sqllib/misc:/usr/lib/mit/
↪  bin:/usr/lib/mit/sbin:/opt/cray/pe/bin
MAIL=/var/mail/USER
MODULE_VERSION=3.2.11.1
PAT_REPORT_PRUNE_NAME=_cray$mt_execute_,_cray$mt_sta
↪  rt_,__cray_hwpc_,f_cray_hwpc_,cstart,__pat_,pat_
↪  region_,PAT_,OMP.slave_loop,slave_entry,_new_sla
↪  ve_entry,_thread_pool_slave_entry,THREAD_POOL_jo
↪  in,__libc_start_main,_start,__start,start_thread
↪  ,__wrap_,UPC_ADIO_,_upc_,upc_,__caf_,__pgas_,sys
↪  call,__device_stub
PE_HDF5_DEFAULT_VOLATILE_PKGCONFIG_PATH=/opt/cray/pe
↪  /hdf5/1.10.2.0/@PRGENV@/@PE_HDF5_DEFAULT_GENCOMP
↪  S@/lib/pkgconfig
PE_PKGCONFIG_DEFAULT_PRODUCTS=PE_TRILINOS:PE_TPSL_64
↪  :PE_TPSL:PE_PETSC:PE_PARALLEL_NETCDF:PE_NETCDF_H
↪  DF5PARALLEL:PE_NETCDF:PE_MPICH:PE_LIBSCI:PE_HDF5
↪  _PARALLEL:PE_HDF5:PE_GA:PE_FFTW
PE_TPSL_DEFAULT_GENCOMPILERS_GNU_x86_64=8.2 7.1 5.3
↪  4.9
PE_TPSL_DEFAULT_GENCOMPS_CRAY_interlagos=86
```

```
PE_MPICH_GENCOMPILERS_GNU=7.1 5.1 4.9
CPU=x86_64
XTPE_NETWORK_TARGET=aries
ATP_IGNORE_SIGTERM=1
PE_FFTW_DEFAULT_TARGET_abudhabi=abudhabi
PE_NETCDF_DEFAULT_GENCOMPILERS_GNU=8.2 7.1 6.1 5.3 4.9
PE_PARALLEL_NETCDF_DEFAULT_GENCOMPILERS_GNU=5.1 4.9
PE_PETSC_DEFAULT_GENCOMPS_CRAY_mic_knl=86
PE_TPSL_64_DEFAULT_GENCOMPILERS_GNU_x86_skylake=8.2
↪ 7.1 6.1
PE_TPSL_DEFAULT_GENCOMPILERS_GNU_haswell=8.2 7.1 5.3
↪ 4.9
_=/usr/bin/env
SSH_SENDS_LOCALE=yes
JAVA_BINDIR=/usr/lib64/jvm/java/bin
PE_HDF5_PARALLEL_DEFAULT_FIXED_PRGENV=CRAY INTEL
PE_HDF5_PARALLEL_DEFAULT_GENCOMPS_GNU=
PE_NETCDF_HDF5PARALLEL_DEFAULT_FIXED_PRGENV=CRAY
↪ INTEL
PE_NETCDF_HDF5PARALLEL_DEFAULT_GENCOMPS_GNU=
PE_SMA_DEFAULT_DIR_CRAY_DEFAULT64=64
PE_TPSL_64_DEFAULT_GENCOMPILERS_CRAY_x86_skylake=8.6
CRAY_UDREG_POST_LINK_OPTS=-L/opt/cray/udreg/2.3.2-6.
↪ 0.6.0_15.18__g5196236.ari/lib64
PE_TPSL_64_DEFAULT_GENCOMPS_CRAY_sandybridge=86
PE_TPSL_64_DEFAULT_VOLATILE_PRGENV=CRAY CRAY64 GNU
↪ GNU64 INTEL INTEL64
PE_TPSL_DEFAULT_GENCOMPILERS_CRAY_mic_knl=8.6
PE_TPSL_DEFAULT_GENCOMPS_INTEL_interlagos=160
TBBROOT=/opt/intel/compilers_and_libraries_2018.2.19
↪ 9/linux/tbb
INPUTRC=/etc/inputrc
CRAYPE_VERSION=2.5.15
CRAY_ALPS_POST_LINK_OPTS=-L/opt/cray/alps/6.6.1-6.0.
↪ 6.1_4.1__ga6396bb.ari/lib64
PE_TPSL_DEFAULT_GENCOMPS_GNU_mic_knl=71 53
PE_MPICH_VOLATILE_PRGENV=CRAY GNU
JAVA_HOME=/usr/lib64/jvm/java
TARGETMODULES=craype-abudhabi:craype-abudhabi-cu:cra
↪ ype-accel-host:craype-accel-nvidia20:craype-acce
↪ l-nvidia30:craype-accel-nvidia35:craype-barcelon
↪ a:craype-broadwell:craype-haswell:craype-hugepag
↪ es128K:craype-hugepages128M:craype-hugepages16M:
↪ craype-hugepages256M:craype-hugepages2M:craype-h
↪ ugepages32M:craype-hugepages4M:craype-hugepages5
↪ 12K:craype-hugepages512M:craype-hugepages64M:cra
↪ ype-hugepages8M:craype-intel-knc:craype-interlag
↪ os:craype-interlagos-cu:craype-istanbul:craype-i
↪ vybridge:craype-mc12:craype-mc8:craype-mic-knl:c
↪ raype-network-aries:craype-network-gemini:craype
↪ -network-infiniband:craype-network-none:craype-ne
↪ twork-seastar:craype-sandybridge:craype-shanghai
↪ :craype-target-compute_node:craype-target-local_
↪ host:craype-target-native:craype-xeon:xtpe-barce
↪ lona:xtpe-interlagos:xtpe-interlagos-cu:xtpe-ist
↪ anbul:xtpe-mc12:xtpe-mc8:xtpe-network-gemini:xtp
↪ e-network-seastar:xtpe-shanghai:xtpe-target-nati
↪ ve:xtpe-xeon
_LMFILES_=/opt/cray/pe/modulefiles/modules/3.2.11.1:
↪ /opt/modulefiles/intel/18.0.0.128:/opt/cray/pe/c
↪ raype/2.5.15/modulefiles/craype-network-aries:/o
↪ pt/cray/pe/modulefiles/craype/2.5.15:/opt/cray/p
↪ e/modulefiles/cray-libsci/18.07.1:/opt/cray/ari/
↪ modulefiles/udreg/2.3.2-6.0.6.0_15.18__g5196236.
↪ ari:/opt/cray/ari/modulefiles/ugni/6.0.14-6.0.6.
↪ 0_18.12__g777707d.ari:/opt/cray/pe/modulefiles/p
↪ mi/5.0.14:/opt/cray/ari/modulefiles/dmapp/7.1.1-
↪ 6.0.6.0_51.37__g5a674e0.ari:/opt/cray/ari/module
↪ files/gni-headers/5.0.12-6.0.6.0_3.26__g527b6e1.
↪ ari:/opt/cray/ari/modulefiles/xpmem/2.2.14-6.0.6
↪ .1_5.14__g34333c9.ari:/opt/cray/ari/modulefiles/
↪ job/2.2.3-6.0.6.0_9.47__g6c4e934.ari:/opt/cray/a
↪ ri/modulefiles/dvs/2.7_2.2.100-6.0.6.1_12.6__g54
↪ 2256c:/opt/cray/ari/modulefiles/alps/6.6.1-6.0.6
↪ .1_4.1__ga6396bb.ari:/opt/cray/ari/modulefiles/r
↪ ca/2.2.18-6.0.6.0_19.14__g2aa4f39.ari:/opt/cray/
↪ pe/modulefiles/atp/2.1.3:/opt/cray/pe/modulefile
↪ s/perftools-base/7.0.4:/opt/cray/pe/modulefiles/
↪ PrgEnv-intel/6.0.4:/opt/cray/pe/craype/2.5.15/mo
↪ dulefiles/craype-mic-knl:/opt/cray/pe/modulefile
↪ s/cray-mpich/7.7.3:/opt/modulefiles/nompirun/nom
↪ pirun:/soft/environment/modules/modulefiles/dars
↪ han/3.1.5:/soft/environment/modules/modulefiles/
↪ trackdeps:/soft/environment/modules/modulefiles/
↪ xalt
PE_LIBSCI_DEFAULT_OMP_REQUIRES=
PE_MPICH_DEFAULT_GENCOMPS_CRAY=86
PE_PETSC_DEFAULT_GENCOMPILERS_GNU_sandybridge=7.1
↪ 5.3 4.9
PE_TPSL_DEFAULT_GENCOMPILERS_INTEL_haswell=16.0
PE_LIBSCI_MODULE_NAME=cray-libsci/18.07.1
PE_PETSC_DEFAULT_GENCOMPILERS_CRAY_skylake=8.6
PE_TPSL_DEFAULT_GENCOMPILERS_CRAY_interlagos=8.6
PE_TPSL_DEFAULT_GENCOMPILERS_GNU_mic_knl=7.1 5.3
```

```
LANG=en_US.UTF-8
GDB_CROSS=/opt/intel/debugger_2018/gdb/intel64/bin/g
↪ db-ia
PE_TPSL_64_DEFAULT_GENCOMPS_GNU_x86_skylake=82 71 61
PE_INTEL_FIXED_PKGCONFIG_PATH=/opt/cray/pe/mpt/7.7.3
↪ /gni/mpich-intel/16.0/lib/pkgconfig
MODULEPATH=/opt/cray/pe/perftools/7.0.4/modulefiles:
↪ /opt/cray/pe/craype/2.5.15/modulefiles:/opt/cray
↪ /pe/modulefiles:/opt/cray/modulefiles:/opt/modul
↪ efiles:/soft/environment/modules/modulefiles:/op
↪ t/cray/ari/modulefiles
PYTHONSTARTUP=/etc/pythonstart
PE_LIBSCI_GENCOMPILERS_CRAY_x86_64=8.6
PE_MPICH_NV_LIBS_nvidia20=-lcudart
PE_MPICH_VOLATILE_PKGCONFIG_PATH=/opt/cray/pe/mpt/7.
↪ 7.3/gni/mpich-@PRGENV@@PE_MPICH_DIR_DEFAULT64@/@
↪ PE_MPICH_GENCOMPS@/lib/pkgconfig
SDK_HOME=/usr/lib64/jvm/java
TZ=UTC
LOADEDMODULES=modules/3.2.11.1:intel/18.0.0.128:cray
↪ pe-network-aries:craype/2.5.15:cray-libsci/18.07
↪ .1:udreg/2.3.2-6.0.6.0_15.18__g5196236.ari:ugni/
↪ 6.0.14-6.0.6.0_18.12__g777707d.ari:pmi/5.0.14:dm
↪ app/7.1.1-6.0.6.0_51.37__g5a674e0.ari:gni-header
↪ s/5.0.12-6.0.6.0_3.26__g527b6e1.ari:xpmem/2.2.14
↪ -6.0.6.1_5.14__g34333c9.ari:job/2.2.3-6.0.6.0_9.4
↪ 7__g6c4e934.ari:dvs/2.7_2.2.100-6.0.6.1_12.6__g5
↪ 42256c:alps/6.6.1-6.0.6.1_4.1__ga6396bb.ari:rca/
↪ 2.2.18-6.0.6.0_19.14__g2aa4f39.ari:atp/2.1.3:per
↪ ftools-base/7.0.4:PrgEnv-intel/6.0.4:craype-mic-
↪ knl:cray-mpich/7.7.3:nompirun/nompirun:darshan/3
↪ .1.5:trackdeps:xalt
SHMEM_ABORT_ON_ERROR=1
CRAY_DMAPP_POST_LINK_OPTS=-L/opt/cray/dmapp/7.1.1-6.
↪ 0.6.0_51.37__g5a674e0.ari/lib64
PE_FFTW_DEFAULT_TARGET_ivybridge=ivybridge
PE_FFTW_DEFAULT_TARGET_share=share
PE_FFTW_DEFAULT_TARGET_x86_skylake=x86_skylake
PE_PKG_CONFIG_PATH=/opt/cray/pe/cti/1.0.7/lib/pkgcon
↪ fig:/opt/cray/pe/cti/1.0.6/lib/pkgconfig:/opt/cr
↪ ay/pe/cti/1.0.4/lib/pkgconfig:/opt/cray/pe/cti/1
↪ .0.5/lib/pkgconfig
PE_TPSL_64_DEFAULT_GENCOMPS_GNU_interlagos=82 71 53
↪ 49
PE_TPSL_DEFAULT_GENCOMPILERS_INTEL_mic_knl=16.0
CRAY_RCA_INCLUDE_OPTS=-I/opt/cray/rca/2.2.18-6.0.6.0
↪ _19.14__g2aa4f39.ari/include
↪ -I/opt/cray/krca/2.2.4-6.0.6.1_5.30__g8505b97.ar
↪ i/include
↪ -I/opt/cray-hss-devel/8.0.0/include
PAT_BUILD_PAPI_BASEDIR=/opt/cray/pe/papi/5.6.0.4
PE_LIBSCI_OMP_REQUIRES_openmp=_mp
PE_PETSC_DEFAULT_GENCOMPILERS_GNU_skylake=6.1
PE_TPSL_DEFAULT_GENCOMPILERS_CRAY_x86_skylake=8.6
PE_TPSL_64_DEFAULT_GENCOMPS_CRAY_mic_knl=86
PE_TPSL_DEFAULT_GENCOMPILERS_INTEL_x86_64=16.0
CRAY_MPICH_DIR=/opt/cray/pe/mpt/7.7.3/gni/mpich-inte
↪ l/16.0
PE_MPICH_CXX_PKGCONFIG_LIBS=mpichcxx
DB2_HOME=/dbhome/db2cat/sqllib
DAALROOT=/opt/intel/compilers_and_libraries_2018.2.1
↪ 99/linux/daal
PE_LIBSCI_DEFAULT_GENCOMPS_INTEL_x86_64=160
PE_MPICH_PKGCONFIG_VARIABLES=PE_MPICH_NV_LIBS_@accel
↪ erator@:PE_MPICH_ALTERNATE_LIBS_@multithreaded@:
↪ PE_MPICH_ALTERNATE_LIBS_@dpm@
APP2_STATE=7.0.4
CRAY_PMI_POST_LINK_OPTS=-L/opt/cray/pe/pmi/5.0.14/li
↪ b64
MPM_LAUNCHER=/opt/intel/debugger_2018/mpm/mic/bin/st
↪ art_mpm.sh
PE_HDF5_DEFAULT_FIXED_PRGENV=CRAY INTEL
PE_TPSL_64_DEFAULT_GENCOMPILERS_CRAY_mic_knl=8.6
PE_TPSL_DEFAULT_GENCOMPILERS_INTEL_x86_skylake=16.0
PE_TPSL_DEFAULT_VOLATILE_PKGCONFIG_PATH=/opt/cray/pe
↪ /tpsl/18.06.1/@PRGENV@/@PE_TPSL_DEFAULT_GENCOMPS
↪ @/@PE_TPSL_DEFAULT_TARGET@/lib/pkgconfig
CRAY_MPICH2_VER=7.7.3
PE_MPICH_PKGCONFIG_LIBS=mpich
GPG_TTY=/dev/pts/30
INTEL_PYTHONHOME=/opt/intel/debugger_2018/python/int
↪ el64/
PE_GA_DEFAULT_GENCOMPILERS_GNU=5.3 4.9
PE_LIBSCI_VOLATILE_PKGCONFIG_PATH=/opt/cray/pe/libsc
↪ i/18.07.1/@PRGENV@/@PE_LIBSCI_GENCOMPS@/@PE_LIBS
↪ CI_TARGET@/lib/pkgconfig
PE_MPICH_ALTERNATE_LIBS_multithreaded=_mt
PE_NETCDF_DEFAULT_FIXED_PRGENV=CRAY INTEL
PE_PARALLEL_NETCDF_DEFAULT_FIXED_PRGENV=CRAY INTEL
HOME=/home/USER
SHLVL=2
JDK_HOME=/usr/lib64/jvm/java
QT_SYSTEM_DIR=/usr/share/desktop-data
CRAY_LIBSCI_VERSION=18.07.1
PE_HDF5_PARALLEL_DEFAULT_VOLATILE_PRGENV=GNU
PE_MPICH_TARGET_VAR_nvidia35=-lcudart
PE_NETCDF_HDF5PARALLEL_DEFAULT_VOLATILE_PRGENV=GNU
PE_PKGCONFIG_PRODUCTS_DEFAULT=PE_PAPI
PE_TPSL_64_DEFAULT_GENCOMPS_GNU_haswell=82 71 53 49
OSTYPE=linux
LESS_ADVANCED_PREPROCESSOR=no
IBM_DB_INCLUDE=/dbhome/db2cat/sqllib/include
PE_TPSL_DEFAULT_GENCOMPILERS_INTEL_interlagos=16.0
PE_MPICH_DEFAULT_VOLATILE_PKGCONFIG_PATH=/opt/cray/p
↪ e/mpt/7.7.3/gni/mpich-@PRGENV@@PE_MPICH_DEFAULT_
↪ DIR_DEFAULT64@/@PE_MPICH_DEFAULT_GENCOMPS@/lib/p
↪ kgconfig
PE_PETSC_DEFAULT_GENCOMPILERS_CRAY_interlagos=8.6
PE_TPSL_DEFAULT_VOLATILE_PRGENV=CRAY CRAY64 GNU GNU64
↪ INTEL INTEL64
XCURSOR_THEME=DMZ
LS_OPTIONS=-N --color=none -T 0
```

```
CRAY_PMI_INCLUDE_OPTS=-I/opt/cray/pe/pmi/5.0.14/incl
↪ ude
PE_TPSL_64_DEFAULT_GENCOMPS_CRAY_interlagos=86
PE_TPSL_DEFAULT_GENCOMPS_INTEL_sandybridge=160
WINDOWMANAGER=
PRGENVMODULES=PrgEnv-cray:PrgEnv-gnu:PrgEnv-intel:Pr
↪ gEnv-pgi
CRAYPE_NETWORK_TARGET=aries
ATP_MRNET_COMM_PATH=/opt/cray/pe/atp/2.1.3/libexec/a
↪ tp_mrnet_commnode_wrapper
PE_TPSL_DEFAULT_GENCOMPILERS_CRAY_haswell=8.6
PKG_CONFIG_PATH_DEFAULT=/opt/cray/pe/papi/5.6.0.4/li
↪ b64/pkgconfig
PE_MPICH_DIR_CRAY_DEFAULT64=64
PE_PETSC_DEFAULT_GENCOMPILERS_GNU_haswell=7.1 5.3 4.9
PE_TPSL_64_DEFAULT_GENCOMPILERS_GNU_mic_knl=7.1 5.3
PE_TPSL_DEFAULT_GENCOMPILERS_GNU_interlagos=8.2 7.1
↪ 5.3 4.9
PE_TPSL_DEFAULT_GENCOMPILERS_INTEL_sandybridge=16.0
LOGNAME=USER
MACHTYPE=x86_64-suse-linux
LESS=-M -I -R
G_FILENAME_ENCODING=@locale,UTF-8,ISO-8859-15,CP1252
CRAY_GNI_HEADERS_INCLUDE_OPTS=-I/opt/cray/gni-header
↪ s/5.0.12-6.0.6.0_3.26__g527b6e1.ari/include
CRAY_LIBSCI_PREFIX_DIR=/opt/cray/pe/libsci/18.07.1/I
↪ NTEL/16.0/x86_64
PE_HDF5_DEFAULT_GENCOMPS_GNU=
PE_MPICH_NV_LIBS=
PE_NETCDF_DEFAULT_REQUIRED_PRODUCTS=PE_HDF5
PE_TPSL_64_DEFAULT_GENCOMPILERS_GNU_haswell=8.2 7.1
↪ 5.3 4.9
PE_TPSL_64_DEFAULT_GENCOMPILERS_INTEL_sandybridge=16
↪ .0
PE_TPSL_DEFAULT_GENCOMPS_GNU_x86_64=82 71 53 49
PE_TRILINOS_DEFAULT_REQUIRED_PRODUCTS=PE_MPICH:PE_HD
↪ F5_PARALLEL:PE_NETCDF_HDF5PARALLEL:PE_LIBSCI:PE_
↪ TPSL
CVS_RSH=ssh
DMAPP_ABORT_ON_ERROR=1
PE_LIBSCI_OMP_REQUIRES=
PE_MPICH_DEFAULT_GENCOMPILERS_CRAY=8.6
PE_TRILINOS_DEFAULT_GENCOMPS_GNU_x86_64=82 73 51 49
PE_MPICH_GENCOMPS_CRAY=86
SSH_CONNECTION=128.135.98.18 64766 140.221.69.22 22
CLASSPATH=/opt/intel/compilers_and_libraries_2018.2.
↪ 199/linux/mpi/intel64/lib/mpi.jar:/opt/intel/com
↪ pilers_and_libraries_2018.2.199/linux/daal/lib/d
↪ aal.jar:/opt/intel/compilers_and_libraries_2018.
↪ 0.128/linux/daal/lib/daal.jar:/dbhome/db2cat/sql
↪ lib/java/db2java.zip:/dbhome/db2cat/sqllib/java/
↪ sqlj.zip:/dbhome/db2cat/sqllib/function:/dbhome/
↪ db2cat/sqllib/java/db2jcc_license_cu.jar:/dbhome
↪ /db2cat/sqllib/tools/clpplus.jar:/dbhome/db2cat/
↪ sqllib/tools/jline-0.9.93.jar:/dbhome/db2cat/sql
↪ lib/java/db2jcc.jar:.
XDG_DATA_DIRS=/usr/share
TOOLMODULES=apprentice:apprentice2:atp:chapel:cray-l
↪ gdb:craypat:craypkg-gen:cray-snplauncher:ddt:gdb
↪ :iobuf:papi:perftools:perftools-lite:stat:totalv
↪ iew:xt-craypat:xt-lgdb:xt-papi:xt-totalview
DVS_INCLUDE_OPTS=-I/opt/cray/dvs/2.7_2.2.100-6.0.6.1
↪ _12.6__g542256c/include
PE_LIBSCI_DEFAULT_REQUIRED_PRODUCTS=PE_MPICH
PE_MPICH_DEFAULT_FIXED_PRGENV=INTEL
PE_MPICH_DEFAULT_GENCOMPS_GNU=71 51 49
PE_TPSL_64_DEFAULT_GENCOMPILERS_INTEL_interlagos=16.0
PE_TPSL_DEFAULT_GENCOMPILERS_CRAY_sandybridge=8.6
TRACKDEPS_CMDS=cc1 cc1plus f951 clang clang-3.9
↪ clang-4.0 clang-5.0 clang-6.0 clang-7.0 cpp as
↪ mcpcom fortcom ccfe ftnfe driver ar ld.bfd
↪ ld.gold ld
MODULESHOME=/opt/cray/pe/modules/3.2.11.1
CRAY_PRGENVINTEL=loaded
PE_GA_DEFAULT_FIXED_PRGENV=CRAY INTEL
PE_LIBSCI_DEFAULT_VOLATILE_PKGCONFIG_PATH=/opt/cray/
↪ pe/libsci/18.07.1/@PRGENV@/@PE_LIBSCI_DEFAULT_GE
↪ NCOMPS@/@PE_LIBSCI_DEFAULT_TARGET@/lib/pkgconfig
PE_TPSL_DEFAULT_GENCOMPILERS_GNU_sandybridge=8.2 7.1
↪ 5.3 4.9
TRACKDEPS_LOG_PATH=/lus/theta-fs0/logs/trackdeps
LESSOPEN=lessopen.sh %s
PE_MPICH_NV_LIBS_nvidia35=-lcudart
PE_PETSC_DEFAULT_VOLATILE_PKGCONFIG_PATH=/opt/cray/p
↪ e/petsc/3.8.4.0/complex/@PRGENV@/@PE_PETSC_DEFAU
↪ LT_GENCOMPS@/@PE_PETSC_DEFAULT_TARGET@/lib/pkgco
↪ nfig
PKG_CONFIG_PATH=/opt/intel/compilers_and_libraries_2
↪ 018.2.199/linux/mkl/bin/pkgconfig:/soft/perftool
↪ s/darshan/darshan-3.1.5/lib/pkgconfig:/opt/cray/
↪ rca/2.2.18-6.0.6.0_19.14__g2aa4f39.ari/lib64/pkg
↪ config:/opt/cray/alps/6.6.1-6.0.6.1_4.1__ga6396b
↪ b.ari/lib64/pkgconfig:/opt/cray/xpmem/2.2.14-6.0
↪ .6.1_5.14__g34333c9.ari/lib64/pkgconfig:/opt/cra
↪ y/gni-headers/5.0.12-6.0.6.0_3.26__g527b6e1.ari/
↪ lib64/pkgconfig:/opt/cray/dmapp/7.1.1-6.0.6.0_51
↪ .37__g5a674e0.ari/lib64/pkgconfig:/opt/cray/pe/p
↪ mi/5.0.14/lib64/pkgconfig:/opt/cray/ugni/6.0.14-
↪ 6.0.6.0_18.12__g777707d.ari/lib64/pkgconfig:/opt
↪ /cray/udreg/2.3.2-6.0.6.0_15.18__g5196236.ari/li
↪ b64/pkgconfig:/opt/cray/pe/craype/2.5.15/pkg-con
↪ fig:/opt/cray/pe/iobuf/2.0.8/lib/pkgconfig:/opt/
↪ cray/pe/atp/2.1.3/lib/pkgconfig
COMPILER_PATH=/opt/xalt/bin
PELOCAL_PRGENV=true
CRAYPAT_OPTS_EXECUTABLE=sbin/pat-opts
LIBSCI_BASE_DIR=/opt/cray/pe/libsci/18.07.1
PE_TPSL_64_DEFAULT_GENCOMPS_INTEL_x86_64=160
```

```
INFOPATH=/opt/intel/documentation_2018/en/debugger//
↪  gdb-ia/info/:/opt/intel/documentation_2018/en/de
↪  bugger//gdb-igfx/info/:/opt/intel/documentation_
↪  2017/en/debugger//gdb-ia/info/:/opt/intel/docume
↪  ntation_2017/en/debugger//gdb-mic/info/:/opt/int
↪  el/documentation_2017/en/debugger//gdb-igfx/info/
LIBSCI_VERSION=18.07.1
PE_LIBSCI_DEFAULT_PKGCONFIG_VARIABLES=PE_LIBSCI_DEFA
↪  ULT_OMP_REQUIRES_@openmp@:PE_SCI_EXT_LIBPATH:PE_
↪  SCI_EXT_LIBNAME
PE_MPICH_NV_LIBS_nvidia60=-lcudart
PE_TPSL_64_DEFAULT_GENCOMPS_GNU_sandybridge=82 71 53
↪  49
PE_TPSL_DEFAULT_GENCOMPS_INTEL_mic_knl=160
XDG_RUNTIME_DIR=/run/user/33420
CRAY_PRE_COMPILE_OPTS=-hnetwork=aries
CRAY_ALPS_INCLUDE_OPTS=-I/opt/cray/alps/6.6.1-6.0.6.
↪  1_4.1__ga6396bb.ari/include
PE_FFTW_DEFAULT_TARGET_broadwell=broadwell
PE_LIBSCI_GENCOMPILERS_INTEL_x86_64=16.0
PE_TPSL_64_DEFAULT_GENCOMPILERS_GNU_x86_64=8.2 7.1
↪  5.3 4.9
CRAY_CPU_TARGET=mic-knl
CRAY_UGNI_INCLUDE_OPTS=-I/opt/cray/ugni/6.0.14-6.0.6
↪  .0_18.12__g777707d.ari/include
CRAY_XPMEM_INCLUDE_OPTS=-I/opt/cray/xpmem/2.2.14-6.0
↪  .6.1_5.14__g34333c9.ari/include
PE_LIBSCI_REQUIRED_PRODUCTS=PE_MPICH
PE_PAPI_DEFAULT_ACCELL_FAMILY_LIBS=
PE_TPSL_64_DEFAULT_GENCOMPS_CRAY_x86_64=86
craype_already_loaded=0
PE_LIBSCI_DEFAULT_GENCOMPILERS_GNU_x86_64=7.1 6.1 5.1
↪  4.9
PE_LIBSCI_GENCOMPS_GNU_x86_64=71 61 51 49
PE_TPSL_DEFAULT_GENCOMPS_INTEL_haswell=160
LESSCLOSE=lessclose.sh %s %s
DB2LIB=/dbhome/db2cat/sqllib/lib
ATP_HOME=/opt/cray/pe/atp/2.1.3
PE_FFTW_DEFAULT_TARGET_x86_64=x86_64
PE_PETSC_DEFAULT_GENCOMPILERS_INTEL_x86_64=16.0
G_BROKEN_FILENAMES=1
CRAY_LD_LIBRARY_PATH=/opt/cray/pe/mpt/7.7.3/gni/mpic
↪  h-intel/16.0/lib:/opt/cray/pe/perftools/7.0.4/li
↪  b64:/opt/cray/rca/2.2.18-6.0.6.0_19.14__g2aa4f39
↪  .ari/lib64:/opt/cray/alps/6.6.1-6.0.6.1_4.1__ga6
↪  396bb.ari/lib64:/opt/cray/xpmem/2.2.14-6.0.6.1_5
↪  .14__g34333c9.ari/lib64:/opt/cray/dmapp/7.1.1-6.
↪  0.6.0_51.37__g5a674e0.ari/lib64:/opt/cray/pe/pmi
↪  /5.0.14/lib64:/opt/cray/ugni/6.0.14-6.0.6.0_18.1
↪  2__g777707d.ari/lib64:/opt/cray/udreg/2.3.2-6.0.
↪  6.0_15.18__g5196236.ari/lib64:/opt/cray/pe/libsc
↪  i/18.07.1/INTEL/16.0/x86_64/lib
PE_FFTW_DEFAULT_TARGET_haswell=haswell
PE_GA_DEFAULT_GENCOMPS_GNU=53 49
PE_GA_DEFAULT_VOLATILE_PKGCONFIG_PATH=/opt/cray/pe/g
↪  a/5.3.0.8/@PRGENV@/@PE_GA_DEFAULT_GENCOMPS@/lib/
↪  pkgconfig
PE_INTEL_DEFAULT_FIXED_PKGCONFIG_PATH=/opt/cray/pe/p
↪  arallel-netcdf/1.8.1.3/INTEL/16.0/lib/pkgconfig:
↪  /opt/cray/pe/netcdf-hdf5parallel/4.6.1.3/INTEL/1
↪  6.0/lib/pkgconfig:/opt/cray/pe/netcdf/4.6.1.3/IN
↪  TEL/16.0/lib/pkgconfig:/opt/cray/pe/mpt/7.7.3/gn
↪  i/mpich-intel/16.0/lib/pkgconfig:/opt/cray/pe/hd
↪  f5-parallel/1.10.2.0/INTEL/16.0/lib/pkgconfig:/o
↪  pt/cray/pe/hdf5/1.10.2.0/INTEL/16.0/lib/pkgconfi
↪  g:/opt/cray/pe/ga/5.3.0.8/INTEL/18.0/lib/pkgconf
↪  ig
PE_PAPI_DEFAULT_ACCEL_LIBS=
PE_PETSC_DEFAULT_GENCOMPILERS_GNU_interlagos=7.1 5.3
↪  4.9
PE_PETSC_DEFAULT_GENCOMPILERS_INTEL_haswell=16.0
PE_SMA_DEFAULT_DIR_PGI_DEFAULT64=64
PE_TPSL_64_DEFAULT_GENCOMPILERS_INTEL_x86_skylake=16
↪  .0
COLORTERM=1
JAVA_ROOT=/usr/lib64/jvm/java
I_MPI_ROOT=/opt/intel/compilers_and_libraries_2018.2
↪  .199/linux/mpi
PE_MPICH_DEFAULT_DIR_CRAY_DEFAULT64=64
PE_PETSC_DEFAULT_GENCOMPS_CRAY_haswell=86
PE_PETSC_DEFAULT_GENCOMPS_GNU_x86_64=71 53 49
PE_PETSC_DEFAULT_GENCOMPS_INTEL_x86_64=160
intel_already_loaded=0
BASH_FUNC_module%%=() {  eval
↪  `/opt/cray/pe/modules/3.2.11.1/bin/modulecmd
↪  bash $*`
}
+ lsb_release -a
LSB Version:        n/a
Distributor ID:        SUSE
Description:        SUSE Linux Enterprise Server 12
↪  SP3
Release:        12.3
Codename:        n/a
+ uname -a
Linux thetalogin6 4.4.162-94.72-default #1 SMP Mon
↪  Nov 12 18:57:45 UTC 2018 (9de753f) x86_64 x86_64
↪  x86_64 GNU/Linux
+ lscpu
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
CPU(s):                32
On-line CPU(s) list:   0-31
Thread(s) per core:    1
Core(s) per socket:    16
Socket(s):             2
NUMA node(s):          2
Vendor ID:             GenuineIntel
CPU family:            6
```

```
Model:                63
Model name:           Intel(R) Xeon(R) CPU E5-2698 v3
↪  @ 2.30GHz
Stepping:             2
CPU MHz:              2216.136
CPU max MHz:          3600.0000
CPU min MHz:          1200.0000
BogoMIPS:             4600.18
Virtualization:       VT-x
L1d cache:            32K
L1i cache:            32K
L2 cache:             256K
L3 cache:             40960K
NUMA node0 CPU(s):
↪  0,2,4,6,8,10,12,14,16,18,20,22,24,26,28,30
NUMA node1 CPU(s):
↪  1,3,5,7,9,11,13,15,17,19,21,23,25,27,29,31
Flags:                fpu vme de pse tsc msr pae mce
↪  cx8 apic sep mtrr pge mca cmov pat pse36 clflush
↪  dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx
↪  pdpe1gb rdtscp lm constant_tsc arch_perfmon pebs
↪  bts rep_good nopl xtopology nonstop_tsc
↪  aperfmperf eagerfpu pni pclmulqdq dtes64 monitor
↪  ds_cpl vmx smx est tm2 ssse3 sdbg fma cx16 xtpr
↪  pdcm pcid dca sse4_1 sse4_2 x2apic movbe popcnt
↪  tsc_deadline_timer aes xsave avx f16c rdrand
↪  lahf_lm abm ida arat epb invpcid_single pln pts
↪  dtherm kaiser tpr_shadow vnmi flexpriority ept
↪  vpid fsgsbase tsc_adjust bmi1 avx2 smep bmi2 erms
↪  invpcid cqm xsaveopt cqm_llc cqm_occup_llc
+ cat /proc/meminfo
MemTotal:       263704472 kB
MemFree:        52834700 kB
MemAvailable:   189864592 kB
Buffers:         1803904 kB
Cached:         139644280 kB
SwapCached:            0 kB
Active:         24494700 kB
Inactive:       128579728 kB
Active(anon):   14617988 kB
Inactive(anon):  3176136 kB
Active(file):    9876712 kB
Inactive(file): 125403592 kB
Unevictable:     8388688 kB
Mlocked:         8388688 kB
SwapTotal:      134217724 kB
SwapFree:       134217724 kB
Dirty:              180 kB
Writeback:            0 kB
AnonPages:      20014360 kB
Mapped:           910668 kB
Shmem:           6168480 kB
Slab:           46859608 kB
SReclaimable:    3598280 kB
SUnreclaim:     43261328 kB
KernelStack:       28112 kB
PageTables:        76212 kB
NFS_Unstable:          0 kB
Bounce:                0 kB
WritebackTmp:          0 kB
CommitLimit:    266069960 kB
Committed_AS:   21277576 kB
VmallocTotal:   34359738367 kB
VmallocUsed:           0 kB
VmallocChunk:          0 kB
HardwareCorrupted:     0 kB
AnonHugePages:   8853504 kB
HugePages_Total:       0
HugePages_Free:        0
HugePages_Rsvd:        0
HugePages_Surp:        0
Hugepagesize:       2048 kB
DirectMap4k:     2230996 kB
DirectMap2M:    240812032 kB
DirectMap1G:    27262976 kB
+ inxi -F -c0
./collect_environment.sh: line 14: inxi: command not
↪  found
+ lsblk -a
NAME      MAJ:MIN RM   SIZE RO TYPE MOUNTPOINT
loop0       7:0    0     3G  0 loop
loop1       7:1    0  50.8G  0 loop
↪  /var/opt/cray/imps-image-binding/PE/squash_mount
↪  s/squashfs_NSX14L_mount_point
loop2       7:2    0   1.6G  0 loop
↪  /var/opt/cray/imps-image-binding/diags/squash_mo
↪  unts/squashfs_lueY5C_mount_point
loop3       7:3    0         1 loop
loop4       7:4    0         0 loop
loop5       7:5    0         0 loop
loop6       7:6    0         0 loop
loop7       7:7    0         0 loop
sda         8:0    0   700G  0 disk
├─sda1      8:1    0  1007K  0 part
├─sda2      8:2    0     2G  0 part /boot
├─sda3      8:3    0    20G  0 part
├─sda4      8:4    0   256G  0 part /tmp
└─sda5      8:5    0   128G  0 part [SWAP]
sdb         8:16   0 417.3G  0 disk
├─sdb1      8:17   0    10G  0 part /var/crash
└─sdb2      8:18   0 407.3G  0 part
↪  /var/opt/cray/persistent
sr0        11:0    1  1024M  0 rom
mira-home 253:150  0         0 disk /gpfs/mira-home
theta-fs1 253:151  0         0 disk /gpfs/theta-fs1
+ lsscsi -s
[0:2:0:0]    disk    DELL     PERC H330 Mini   4.25
↪  /dev/sda    751GB
[0:2:1:0]    disk    DELL     PERC H330 Mini   4.25
↪  /dev/sdb    448GB
```

```
[10:0:0:0]   cd/dvd  PLDS     DVD-ROM DU-8D5LH FD51
↪  /dev/sr0        -
+ module list
++ /opt/cray/pe/modules/3.2.11.1/bin/modulecmd bash
↪  list
Currently Loaded Modulefiles:
  1) modules/3.2.11.1
  2) intel/18.0.0.128
  3) craype-network-aries
  4) craype/2.5.15
  5) cray-libsci/18.07.1
  6) udreg/2.3.2-6.0.6.0_15.18__g5196236.ari
  7) ugni/6.0.14-6.0.6.0_18.12__g777707d.ari
  8) pmi/5.0.14
  9) dmapp/7.1.1-6.0.6.0_51.37__g5a674e0.ari
 10) gni-headers/5.0.12-6.0.6.0_3.26__g527b6e1.ari
 11) xpmem/2.2.14-6.0.6.1_5.14__g34333c9.ari
 12) job/2.2.3-6.0.6.0_9.47__g6c4e934.ari
 13) dvs/2.7_2.2.100-6.0.6.1_12.6__g542256c
 14) alps/6.6.1-6.0.6.1_4.1__ga6396bb.ari
 15) rca/2.2.18-6.0.6.0_19.14__g2aa4f39.ari
 16) atp/2.1.3
 17) perftools-base/7.0.4
 18) PrgEnv-intel/6.0.4
 19) craype-mic-knl
 20) cray-mpich/7.7.3
 21) nompirun/nompirun
 22) darshan/3.1.5
 23) trackdeps
 24) xalt
+ eval
+ nvidia-smi
./collect_environment.sh: line 18: nvidia-smi:
↪  command not found
+ lshw -short -quiet -sanitize
+ cat
./collect_environment.sh: line 19: lshw: command not
↪  found
+ lspci
./collect_environment.sh: line 19: lspci: command not
↪  found
```

ARTIFACT EVALUATION

Verification and validation studies: In the URL, we've provided all the material needed for running the experiments. We can directly compile it on the target system and run the experiments to get the results.

Accuracy and precision of timings: The analytical method of validating the accuracy is described in the paper. As for timings, we use time function to measure the timing.

Quantified the sensitivity of results to initial conditions and/or parameters of the computational environment: We've presented the conditions in detail in the paper. The initial condition of each state would be all 0 state.

Controls, statistics, or other steps taken to make the measurements and analyses robust to variability and unknowns in the system. We run our experiments by using the following scripts with different nodes and test files.

aprun -n allranks -N 128 -d $threads -cc depth -j 4 −env OMP_NUM_THREADS=$threads $EXE < $TEST