Quantum Algorithm Design with QuOp_MPI

E. Matwiejew[1]

Pawsey Supercomputing Research Centre, 1 Bryce Ave, Kensington Western Australia 6151
edric.matwiejew@CSIRO.au

Abstract. QuOp_MPI is a framework for designing and benchmarking quantum variational algorithms (QVA) by quantum simulation. The package provides a highly customisable Python interface to performant MPI-parallel backends, enabling researchers to write concise and scalable simulation code. We present recent results for the integration of GPU acceleration using the open-source HIP programming model into QuOp_MPI to support scalable simulation workflows on the current generation of clusters with heterogeneous computing environments.

Keywords: MPI · HIP · Heterogeneous Parallel Computing · Quantum Computing · Quantum Optimisation

Quantum Variational Algorithms (QVAs) represent a promising approach to solving complex optimisation problems by leveraging the unique strengths of both quantum and classical computing paradigms [3, 10]. At their core, QVAs for optimisation are defined by an ansatz unitary consisting of interleaved, classically parameterised phase-shift $\hat{U}_Q$ and mixing unitaries $\hat{U}_W$, which are applied to an initial superposition state $|\psi_0\rangle$ whose basis states represent the problem's solution space, and a specified number of iterations p:

$$|\gamma, \boldsymbol{t}\rangle = \prod_{i=1}^{p} \hat{U}_W(\boldsymbol{t}_i)\hat{U}_Q(\gamma_i)\,|\psi_0\rangle\,. \tag{1}$$

Both the phase-shift and mixing unitaries are defined by the time evolution unitary $\exp\left(-\mathrm{i}\theta\hat{H}\right)$ with a time-independent Hamiltonian $\hat{H}$ and $\theta \in \mathbb{R}$. In the phase-shift unitary, the evolution is parameterised by $\theta = \gamma$, and the Hamiltonian $\hat{H} = \hat{Q}$ is a diagonal operator that encodes the solution costs into the phase of $|\psi_0\rangle$. The mixing unitary is parameterised by one or more parameters $\theta = \boldsymbol{t}$, with its action defined by a Hamiltonian with non-zero off-diagonal elements $\hat{H} = \hat{W}$. By varying the ansatz parameters $(\gamma, \boldsymbol{t})$ to minimise the average measured solution cost $\langle Q\rangle$, interference is manipulated to amplify the probability of measuring low-cost solutions, with the amount of possible amplification increasing with the iteration count p. These algorithms are of considerable interest as they are robust to noise and have a flexible structure targeting near-term Noisy Intermediate-Scale Quantum (NISQ) processors [3, 8, 10].

However, the path towards the deployment of practical QVAs is not straightforward; the design of practical QVAs faces multiple challenges. The quantum dynamics involved in these algorithms are often complex, precluding analytical proofs of performance or scalability. Consequently, performance advantages over classical methods have only been demonstrated for a limited number of specific instances. Furthermore, current quantum processors are constrained in both the number of qubits and coherence time, making it difficult to validate and benchmark QVAs on actual quantum hardware [1, 9, 11]. As a result, researchers rely heavily on high-performance simulations, which are computationally intensive and require domain-specific expertise. To address these challenges, we have developed QuOp_MPI [7], a framework for the design and benchmarking of QVAs by parallel simulation.

QuOp_MPI enables the rapid prototyping of novel algorithms by providing a modular Python interface to MPI-parallel backends that provide high-precision state vector simulation of the fundamental unitary dynamics. This allows researchers to focus on their research goals while writing concise simulation programs that can scale from their laptop to hundreds of compute nodes [6, 7]. A workflow based on user-definable Python functions that are incorporated into the parallel simulation workflow at run-time achieves dynamic and flexible control of the algorithm unitaries and parameter optimisation scheme.

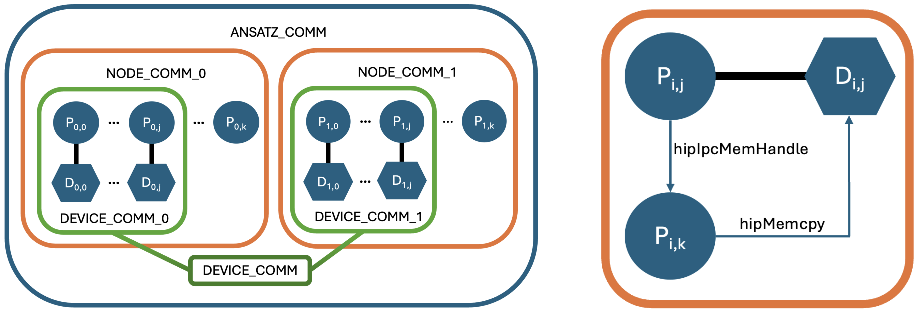
QuOp_MPI adopts a simulation approach that targets Hamiltonian structures efficiently implemented on quantum processors. Algorithms are defined in Python using 'unitary' building blocks that provide an interface to compile extension modules that implement numerical methods optimised according to the adjacency structure of the Hamiltonian [7]. QuOp_MPI dynamically determines the parallel distribution of the state-vector and MPI communication structures according to the backends associated with the simulation instance, allowing users to leverage a combination of MPI-parallel sparse matrix and FFT-based simulation methods through a unified interface.

Recent development has focussed on integrating GPU-accelerated methods into the existing QuOp_MPI codebase through the `QuOp_Wavefront` backend. This integration leverages distributed memory parallelism with MPI and shared-memory parallelism through GPU acceleration using the open-source HIP programming model. Here, we describe a high-dimensional Fast Fourier Transform (FFT) based method for the simulation QVAs for unconstrained combinatorial optimisation problems and benchmark its performance on Setonix, a 30 PetaFlop HPE Cray EX system hosted at the Pawsey Supercomputing Research Centre.

1 QuOp_Wavefront

The `QuOp_Wavefront` is a GPU-accelerated backend for `QuOp_MPI` that integrates with the Fortran `context` and `propagator` interface classes, available from version 1.2 of the package onwards. A single `context` instance manages buffers for the initial and evolved state vectors and handles common operations, including computation of the expectation value $\langle Q \rangle$. Multiple `propagator` instances point to the `context` instance, with each computing the action of a specific unitary type on the state vector via an MPI-parallelised numerical method, interfacing with the `QuOp_MPI` Python layer through a consistent interface. At run-time, the `context` determines a partitioning scheme for the state vector and other distributed arrays, which is modified by the `propagator` instances to arrive at a scheme compatible with their respective numerical methods.

The communicator structure of the GPU-enabled backend is shown in Fig. 1a. A partitioning of the state vector across the available GPUs is first determined, defined by the number of local state vector elements and their corresponding global index offsets, with one MPI process assigned to each device. This partitioning also defines sub-communicators that contain processes within the same shared memory space, denoted as `NODE_COMM_i`, which may include processes not associated with a device. Generation of the initial state, observables, and other user-defined data structures occur on the host over this communicator as part of the simulation setup phase in the Python layer. This approach maintains the flexibility of the Python-interpreted environment while maximising the use of the available CPU cores, which typically outnumber the GPUs by several factors. Buffer transfer from the host to devices uses HIP inter-process communication handles, as shown in Fig. 1b, with device memory access synchronised through blocking point-to-point communication.



(a) (b)

Fig. 1: (a) MPI communicator structure of a `QuOp_Wavefront` QVA simulation instance with two compute nodes. The `DEVICE_COMM` communicator contains processes that are assigned to a GPU, `NODE_COMM_i` and `DEVICE_COMM_i` contain processes in the same shared memory space on the host and `ANSATZ_COMM` is the global communicator for the simulation instance. (b) Data transfer from process $P_{i,k}$ to a buffer on device $D_{i,j}$ which is allocated on process $P_{i,j}$. The `hipIpcMemHandle` instance is sent to $P_{i,k}$ by point-to-point communication over `NODE_COMM_i`.

2 Mixing Unitary Simulation with High-Dimensional Fast Fourier Transforms

In unconstrained combinatorial optimisation problems, the solution space is of the form,

$$\mathcal{S} \cong \mathcal{X}^{\otimes n} = \underbrace{\mathcal{X} \times \mathcal{X} \times \cdots \times \mathcal{X}}_{n \text{ times}}, \tag{2}$$

where $\mathcal{X}$ is an ordered set containing the possible values for each combinatorial variable [5].

In terms of the number of iterations required for substantial amplification, a class of $\hat{W}$ that is often efficient for this problem type is given by the adjacency matrix of the Hamming graphs on $m-$tuples, which connect between solutions that differ in exactly one of their combinatorial variables [2, 5]. These are defined as,

$$H = K_{|\mathcal{X}|}^{\times n}, \tag{3}$$

where $K_{|\mathcal{X}|} = \mathcal{I}_{|\mathcal{X}|} - \delta_{ij}$ is the adjacency matrix of a complete graph and $\times n$ is the Cartesian product of the matrix n times. As $K_{|\mathcal{X}|}$ is a circulant matrix, $\hat{U}_W$ can be efficiently simulated using an $n-$dimensional FFT as,

$$|\psi(t)\rangle = \bigotimes_{j=0}^{n-1} \left(\mathcal{F}_{i_j}^{-1} e^{-\mathrm{i} t_j \hat{\Lambda}_{i_j}} \mathcal{F}_{i_j} \right) |\psi_0\rangle\,, \tag{4}$$

where $\mathcal{F}$ is the the discrete Fourier transform, $|\psi_0\rangle = \bigotimes_{j=0}^{n-1} \alpha_{i,j} |i_j\rangle$ with $i \in 0, \ldots, m-1$ and $\hat{\Lambda} = (|\mathcal{X}| - 1, -1, \ldots, -1|)$ are the eigenvalues of the j-th $K_{|\mathcal{X}|}$.

To achieve this in MPI parallel, `QuOp_Wavefront` utilises a method proposed by Dalcin et al. whereby the state vector is distributed as a balanced block-contiguous slab decomposition over sub-communicators formed from the dimensions of a cartesian communicator and global redistributions are performed using an `MPI_AlltoAllw` with subarray derived datatypes, avoiding the need for local transpositions [4].

To simulate an iteration of a QVA according to Eq. (1), the action of the phase-shift operator is first computed as,

$$\mathcal{A}^{(\gamma)}_{i_0,\ldots,i_{n-1}} = \exp\bigl(-\mathrm{i}\gamma q_{i_0,\ldots,i_{n-1}}\bigr)\mathcal{A}^{(0)}_{i_0,\ldots,i_{n-1}}.$$

where tensor $\mathcal{A}_{i_0,\ldots,i_{n-1}}$ contains the coefficeints of the state vector according to the structure of Eq. (2) and $q_{i_0,\ldots,i_{n-1}} \in \hat{Q}$. Next, an $n-$dimensional FFT is performed over the contiguous coordinates of $\mathcal{A}^{(\gamma)}$,

$$\mathcal{A}^{(\gamma)}_{i_0/P_0,\ldots,i_{m-1}/P_{m-1},i_{k+m},\ldots,i_{n-1}} \xrightarrow{\mathrm{DFT}_{\{i_{k+1},\ldots,i_{n-1}\}}} \tilde{\mathcal{A}}^{(\gamma)}_{i_0/P_0,\ldots,i_{m-1}/P_{m-1},j_{k+m-1},\ldots,j_{n-1}},$$

where i_j/P_l denotes a coordinate that is partitioned over P_l processes. The non-contiguous dimensions are then redistributed as,

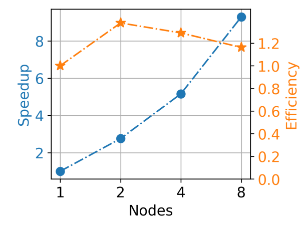
$$\tilde{\mathcal{A}}^{(\gamma)}_{i_0/P_0,\dots,i_{m-1}/P_{m-1},j_{k+m-1},\dots,j_{n-1}} \to \tilde{\mathcal{A}}^{(\gamma)}_{i_0,\dots,i_{m-1},j_{k+m-1},\dots,j_{n-m-1}/P_0,\dots,j_{n-1}/P_{m-1}},$$

which, for a tensor with L distributed dimensions, requires $L-1$ calls to `MPI_AlltoAllw`. The local FFTs at each stage are performed using the hipFFT FFT marshalling library, with local transforms higher than three dimensions performed as a minimal sequence of lower-dimensional batched transforms. The eigenvalue phase shift in Fourier space (see Eq. (4)) is then computed as,

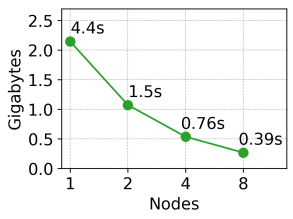
$$\tilde{\mathcal{A}}^{(\gamma,\boldsymbol{t})}_{i_0,\dots,i_{n-1}} = \exp\left(-\mathrm{i}t_{i_k}\lambda_{i_0,\dots,i_{n-1}}\right)\tilde{\mathcal{A}}^{(\gamma)}_{i_0,\dots,i_{n-1}}.$$

After which, the inverse set of transforms and exchanges are performed to obtain the final evolved state.

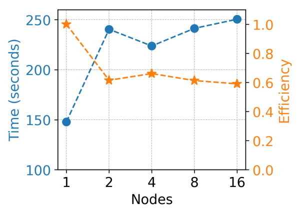
3 Results



(a)



(b)



(c)

Fig. 2: (a) Weak scaling for computation of $\mathcal{A}^{(\gamma,\boldsymbol{t})}$ of dimensions $2^{10}\times 2^{10}\times 2^{10}$ and (b) the subarray size in each call to MPI_AlltoAllw with the mean transfer time shown as annotations. (b) Shows the weak scaling behaviour for simulation of $\mathcal{A}^{\gamma,\boldsymbol{t}}$ where for N nodes, the first $6-\log_2(N)$ dimensions have size 2^5, and the remaining dimensions have size 2^6. Each node has 16 MPI processes and 8 GPUs.

Strong and weak scaling benchmarks for the FFT-based state evaluation method described in Section 2 were performed on the ‘Setonix’ HPE Cray EX system hosted at the Pawsey Supercomputing Research Centre. Each node was configured with an AMD EPYC 7A53 64-Core GPU, eight AMD MI250X GPUs and 256 GB RAM. To benchmark the base performance of the package, the diagonal of Q was generated as a constant `NumPy` array. State propagation was simulated with $p=5$ iterations for five repeats.

Strong scaling results are shown in Fig. 2a, where for 1, 2, 4 and 8 nodes, the dimensions of the initial cartesian communicator were $4\times 2\times 1$, $4\times 4\times 1$, $8\times 1\times 1$

and $8 \times 8 \times 1$, with three batches of one-dimensional FFTs and two redistribution operations required to compute the three-dimensional transform in one direction. As shown in Fig. 2b, the `MPI_AlltoAllw` operation is the dominant factor in program run-time as a doubling in devices halves the data communicated with each redistribution by a factor of two.

The weak scaling behaviour is shown in Fig. 2c, where for 1, 2, 4, 8 and 16 nodes the initial cartesian communicator had dimensions $4 \times 2 \times 1 \times \cdots \times 1$, $4 \times 2 \times 1 \times \cdots \times 1$, $4 \times 4 \times 1 \times \cdots \times 1$, $8 \times 4 \times 1 \times \cdots \times 1$ and $8 \times 8 \times 1 \times \cdots \times 1$. The computation is again memory-bound, however, the efficiency at two nodes and higher is close to 0.6 as the balanced block decomposition ensures that the total data sent to and from each device in an exchange operation remains constant.

4 Conclusion

These results demonstrate that the high-dimensional FFT-based simulation method employed by the `QuOp_Wavefront` backend for Hamming-graph-based QVAs is efficient and scalable in the strong and weak scaling regimes. Future work will address the memory-efficient simulation of QVAs with mixing unitaries defined by sparse Hamiltonians.

References

1. Anschuetz, E.R., Kiani, B.T.: Quantum variational algorithms are swamped with traps. Nature Communications **13**(1), 7760 (2022)
2. Bennett, T., Noakes, L., Wang, J.: Non-variational Quantum Combinatorial Optimisation. arXiv preprint arXiv:2404.03167 (2024)
3. Cerezo, M., Arrasmith, A., Babbush, R., Benjamin, S.C., Endo, S., Fujii, K., McClean, J.R., Mitarai, K., Yuan, X., Cincio, L., *et al.*: Variational quantum algorithms. Nature Reviews Physics **3**(9), 625–644 (2021)
4. Dalcin, L., Mortensen, M., Keyes, D.E.: Fast Parallel Multidimensional FFT Using Advanced MPI. Journal of Parallel and Distributed Computing **128**, 137–150 (2019)
5. Matwiejew, E., Wang, J.: Quantum walk informed variational algorithm design. arXiv preprint arXiv:2406.11620 (2024)
6. Matwiejew, E., Pye, J., Wang, J.B.: Quantum optimisation for continuous multivariable functions by a structured search. Quantum Science and Technology **8**(4), 045013 (2023)
7. Matwiejew, E., Wang, J.B.: QuOp_MPI: A framework for parallel simulation of quantum variational algorithms. Journal of Computational Science **62**, 101711 (2022)
8. Preskill, J.: Quantum computing in the NISQ era and beyond. Quantum **2**, 79 (2018)
9. Streif, M., Leib, M.: Comparison of QAOA with quantum and simulated annealing. (2019). arXiv: 1901.01903 `[quant-ph]`
10. Symons, B.C., Galvin, D., Sahin, E., Alexandrov, V., Mensa, S.: A practitioner's guide to quantum algorithms for optimisation problems. Journal of Physics A: Mathematical and Theoretical **56**(45), 453001 (2023)

11. Zhou, L., Wang, S.-T., Choi, S., Pichler, H., Lukin, M.D.: Quantum approximate optimization algorithm: Performance, mechanism, and implementation on near-term devices. Physical Review X **10**(2), 021067 (2020)