Pawsey

Quantum Algorithm Design with QuOp_MPI

Edric Matwiejew

The Pawsey Supercomputing Research Centre is an unincorporated joint venture between

Core Members

CSIRO · Curtin University · Murdoch University · The University of Western Australia

Founding Associate Member

ECU Edith Cowan University

and proudly funded by

NCRIS National Research Infrastructure for Australia — An Australian Government Initiative

Government of Western Australia

Quantum Computing

Computation on information encoded into the basis states and amplitudes of a quantum state.

The fundamental unit of information: $|\text{qubit}\rangle = c_0|0\rangle + c_1|1\rangle$

A quantum processor with n qubits can store 2^n bits of information in superposition.

Quantum algorithm development aims to manipulate entanglement, interference and superposition to solve difficult problems.

Combinatorial Optimisation

Solves problems in chemistry, finance, logics, logistics, bioinformatics...

Solutions represented as vectors of n **combinatorial variables**: $\boldsymbol{s} = (s_0, \ldots, s_{n-1})$

Each variable can take one of m values: $\mathcal{X} = \{x_0, \ldots, x_{m-1}\}$

Each solution has a **cost**: $C : \boldsymbol{s} \to \mathbb{R}$

Goal: Identify solutions at, or close to, the global minimum:

$$\boldsymbol{s}_{\text{opt}} = \left\{ \boldsymbol{s} \mid C(\boldsymbol{s}) \in \min\{C(\boldsymbol{s}) \mid \boldsymbol{s} \in \mathcal{S}\} \right\}$$

Classically challenging:

- Lack of exploitable local structure
- Search space grows exponentially $|\mathcal{S}| = m^n$

Quantum Variational Algorithms for Optimisation

Ansatz State

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

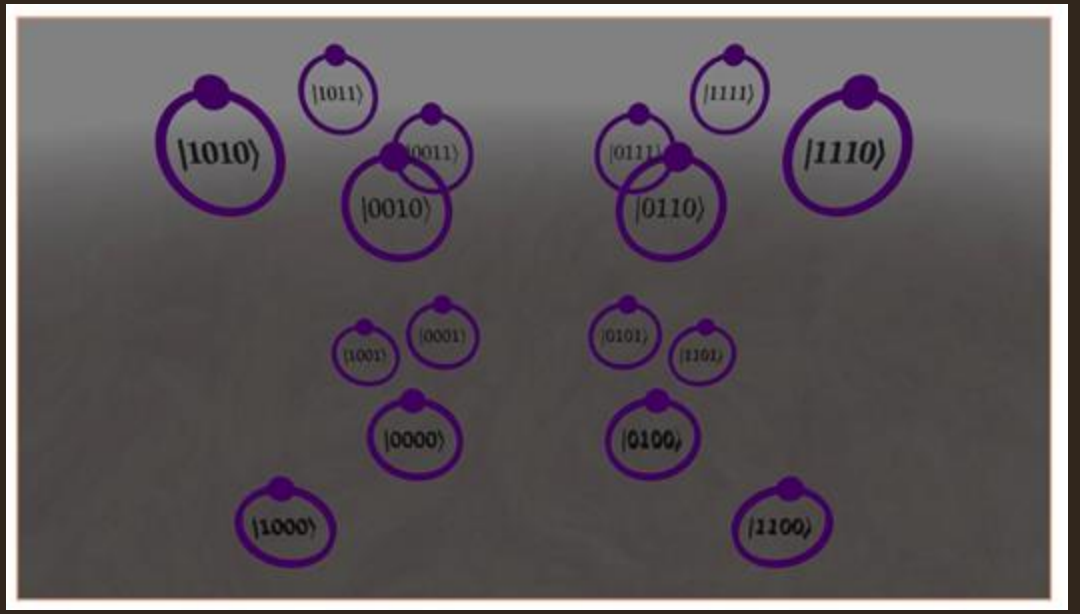
Quantum Variational Algorithms (QVAs) prepare an ansatz state in which low-cost (or high-quality) solutions are **amplified**.

Quantum Variational Algorithms for Optimisation

Initial State
- Solution encoding
- Starting distribution

Ansatz State

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



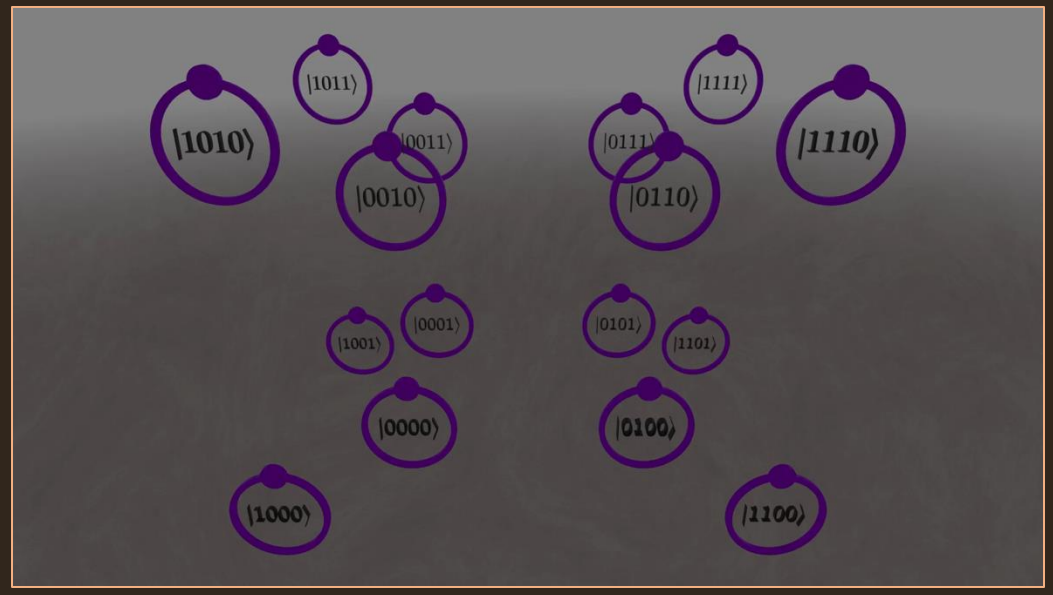
Quantum Variational Algorithms for Optimisation

Ansatz State

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

Initial State
- Solution encoding
- Starting distribution

Phase-Shift Unitary
- Cost function encoding
- Parameterisation (γ_i)



Quantum Variational Algorithms for Optimisation

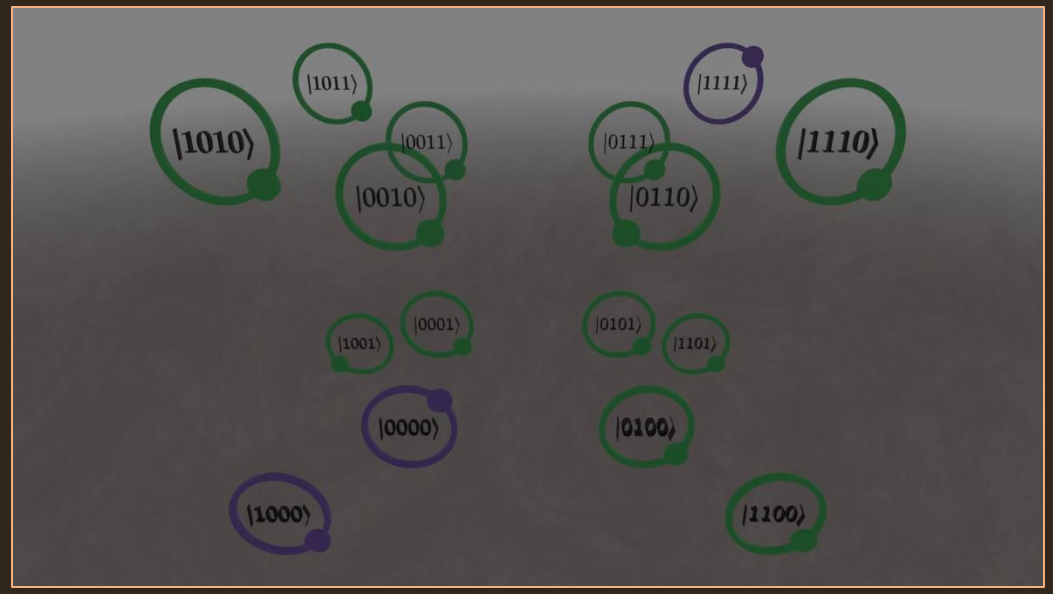
Mixing Unitary
- Graph Structure
- Walk time (t_i)

Initial State
- Solution encoding
- Starting distribution

Ansatz State

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

Phase-Shift Unitary
- Cost function encoding
- Parameterisation (γ_i)



Quantum Variational Algorithms for Optimisation

Mixing Unitary
- Graph Structure
- Walk time (t_i)

Initial State
- Solution encoding
- Starting distribution

Ansatz State

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

Ansatz Depth (p)
- Number of iterations

Phase-Shift Unitary
- Cost function encoding
- Parameterisation (γ_i)

Quantum Variational Algorithms for Optimisation

Mixing Unitary
- Graph Structure
- Walk time (t_i)

Initial State
- Solution encoding
- Starting distribution

Ansatz State

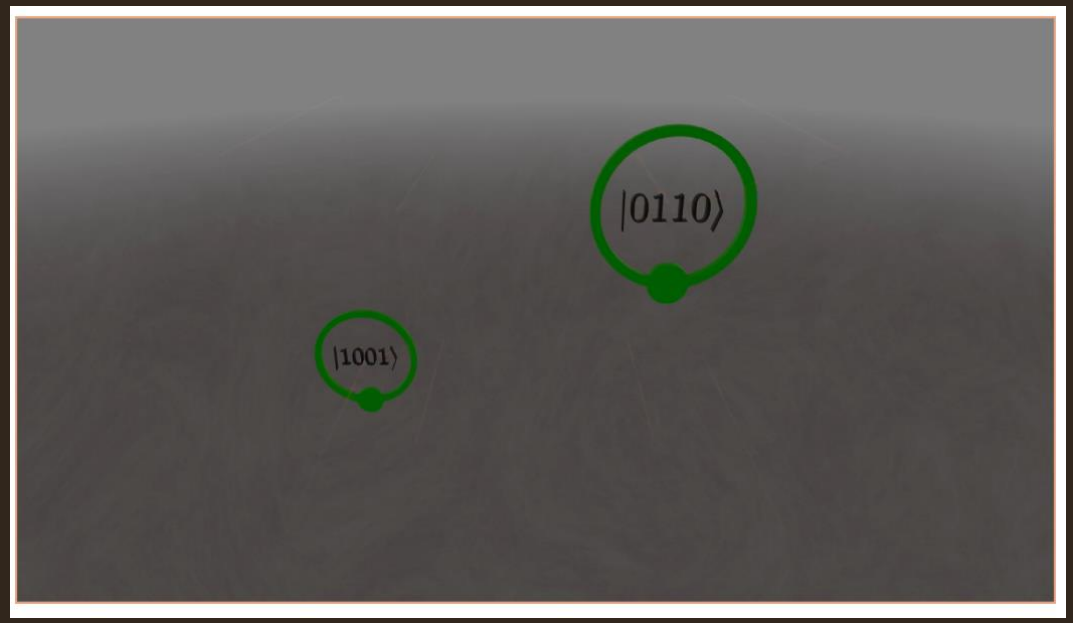
$$|\boldsymbol{\theta}\rangle = \prod_{i=1}^{p} \hat{U}_W(\boldsymbol{t}_i)\hat{U}_Q(\gamma_i)|\psi_0\rangle$$

Ansatz Depth (p)
- Number of iterations

Phase-Shift Unitary
- Cost function encoding
- Parameterisation (γ_i)

Variational Parameters

$$\boldsymbol{\theta} = ((\gamma_0, \boldsymbol{t}_0), (\gamma_1, \boldsymbol{t}_1), \ldots, (\gamma_{p-1}, \boldsymbol{t}_{p-1}))$$



Quantum Variational Algorithms for Optimisation

Mixing Unitary
- Graph Structure
- Walk time (t_i)

Initial State
- Solution encoding
- Starting distribution

Ansatz State

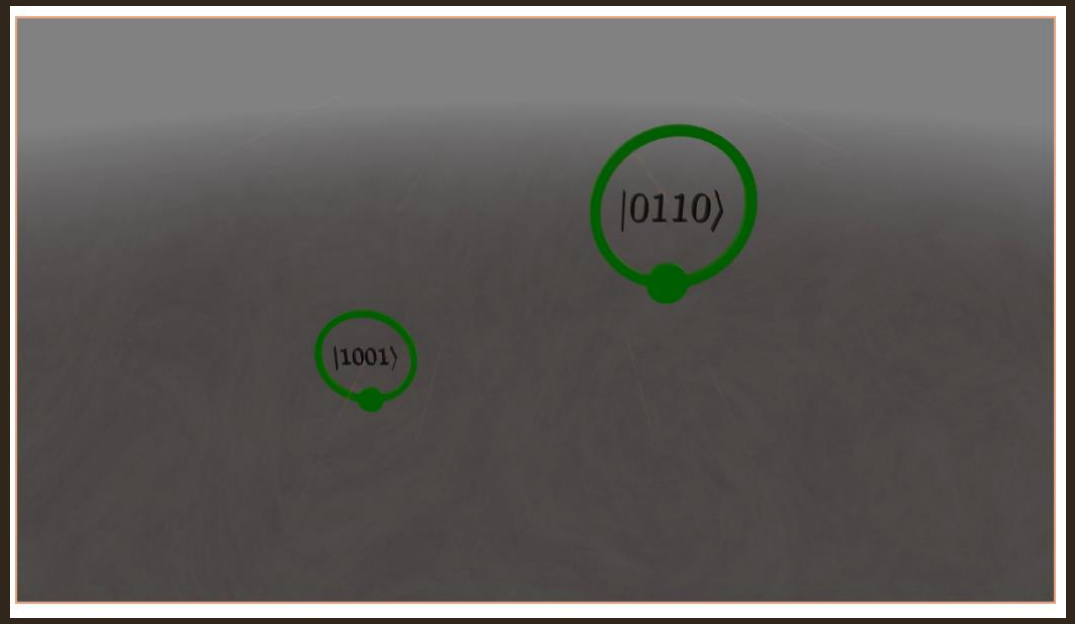
$$|\boldsymbol{\theta}\rangle = \prod_{i=1}^{p} \hat{U}_W(\boldsymbol{t}_i)\hat{U}_Q(\gamma_i)|\psi_0\rangle$$

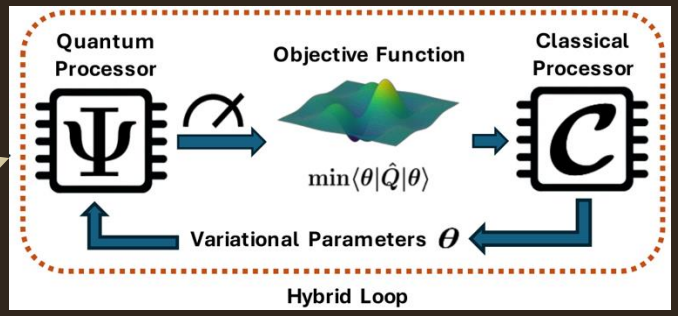
Ansatz Depth (p)
- Number of iterations

Phase-Shift Unitary
- Cost function encoding
- Parameterisation (γ_i)

Variational Parameters

$$\boldsymbol{\theta} = ((\gamma_0, \boldsymbol{t}_0), (\gamma_1, \boldsymbol{t}_1), \ldots, (\gamma_{p-1}, \boldsymbol{t}_{p-1}))$$





Challenges in QVA Design

Complex Quantum Dynamics

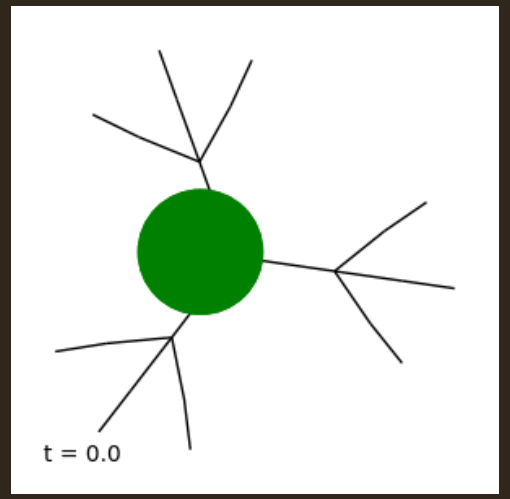
- Difficult to derive analytical proofs for convergence and scalability.

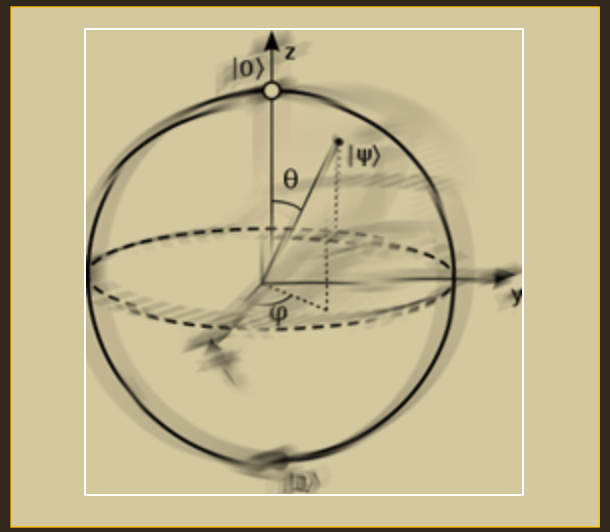
Current Quantum Processors

- Experimental devices
- Size and stability limitations
- Not suitable for validation and benchmarking

Reliance on Classical Simulation

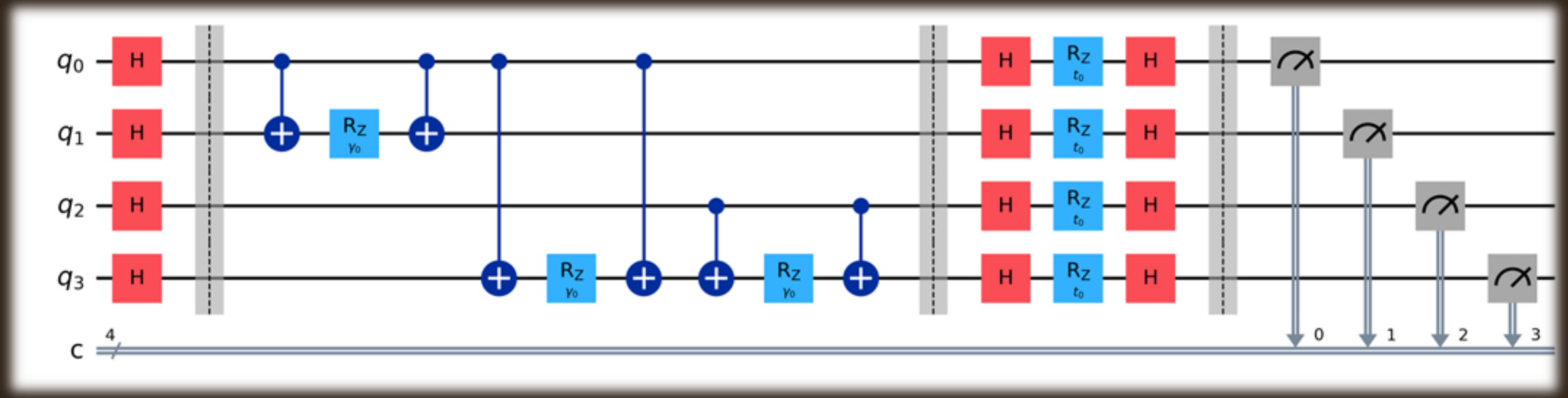
- Computationally intensive
- Requires specialist skillsets





Circuit-Based Simulation

$|\psi_0\rangle$ $\hat{U}_Q$ $\hat{U}_W$



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

$$R_z(\theta) = \begin{pmatrix} e^{-i\frac{\theta}{2}} & 0 \\ 0 & e^{i\frac{\theta}{2}} \end{pmatrix}$$

$$\text{CNOT} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

Hamiltonian-Based Simulation

$$\hat{U}_Q(\gamma)\,|\psi\rangle = \exp\left(-\mathrm{i}\gamma\begin{bmatrix} q_0 & 0 & \dots & 0 \\ 0 & q_1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & q_{N-1} \end{bmatrix}\right)\begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_{N-1} \end{bmatrix} = \begin{bmatrix} e^{-\mathrm{i}\gamma q_0} c_0 \\ e^{-\mathrm{i}\gamma q_1} c_1 \\ \vdots \\ e^{-\mathrm{i}\gamma q_{N-1}} c_{N-1} \end{bmatrix}$$

$$\hat{U}_W(t)\,|\psi\rangle = \exp\left(-it\hat{H}\right)|\psi\rangle = \left(\sum_{n=0}^{\infty}\frac{(-it\hat{H})^n}{n!}\right)|\psi\rangle$$

QuOp_MPI

- **Qu**antum **Op**timisation with **MPI**
- **M**essage **P**assing **I**nterface: parallel computing with networked compute nodes (i.e. **supercomputers**)
- A modular Python interface to MPI-parallel backends
- Study established QVAs and develop novel algorithms
- High-precision state-vector simulation of the fundamental unitary dynamics

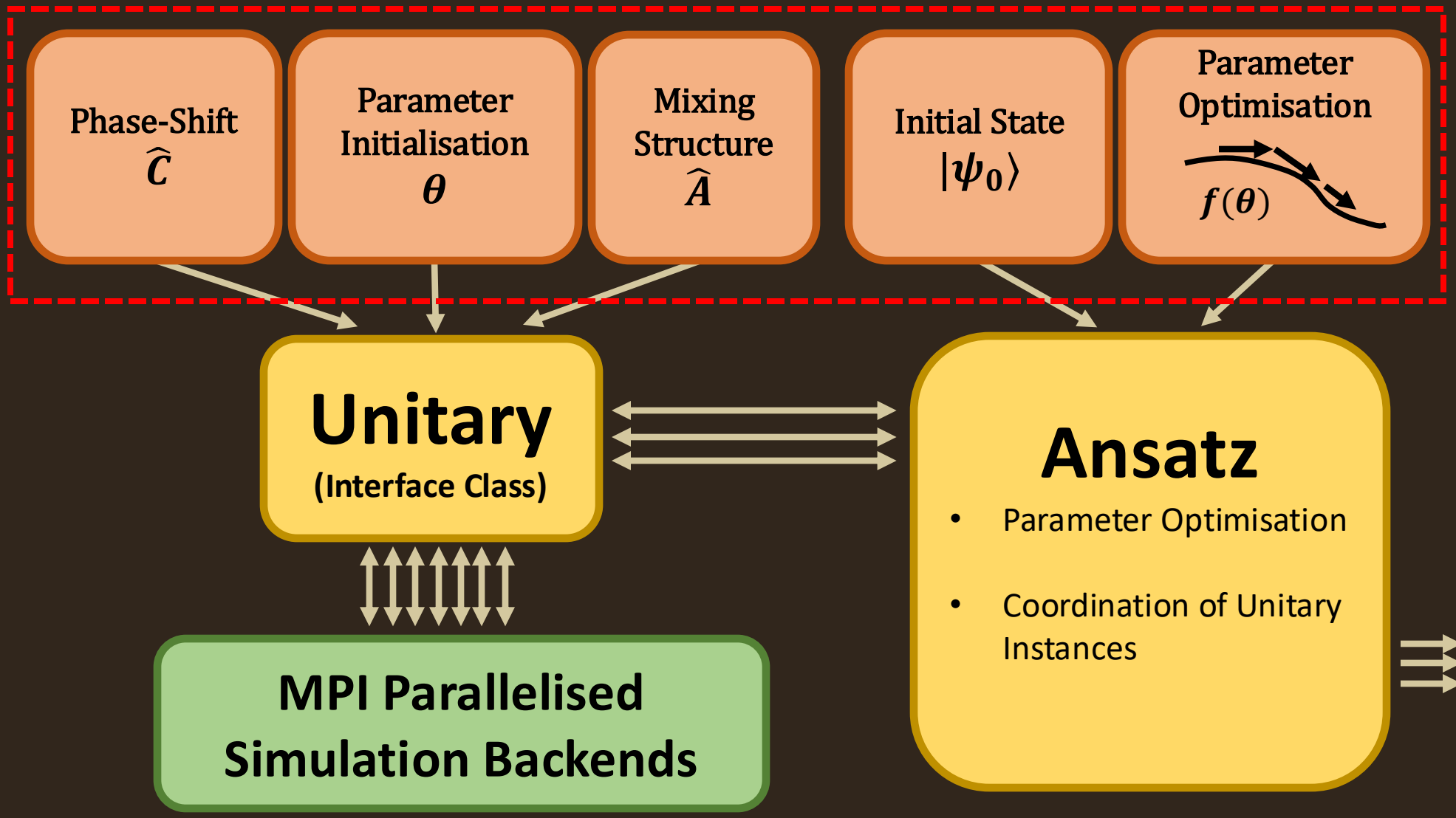
```python
from quop_mpi.algorithm.combinatorial import qaoa, serial
from quop_mpi.toolkit import I, Z
import networkx as nx

Graph = nx.circular_ladder_graph(4)
vertices = len(Graph.nodes)
system_size = 2 ** vertices
G = nx.to_scipy_sparse_array(Graph)

def maxcut_qualities(G):
  C = 0
  for i in range(G.shape[0]):
    for j in range(G.shape[0]):
      if G[i, j] != 0:
        C += 0.5 * (I(vertices) - (Z(i, vertices) @ Z(j, vertices)))
  return -C.diagonal()

alg = qaoa(system_size)
alg.set_qualities(serial, {'args':[maxcut_qualities, G]})
alg.set_depth(2)
alg.execute()
alg.print_result()
alg.save("maxcut", "depth 2", "w")
```

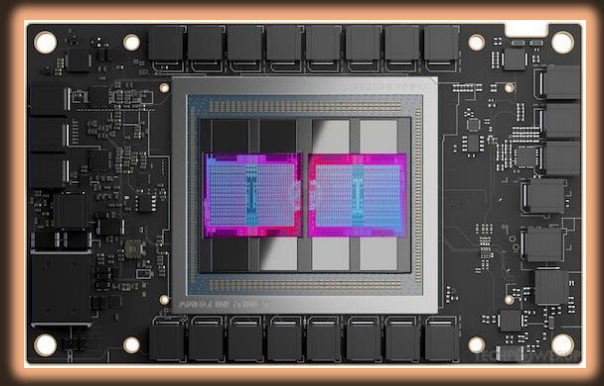
Software Architecture



QuOp Functions

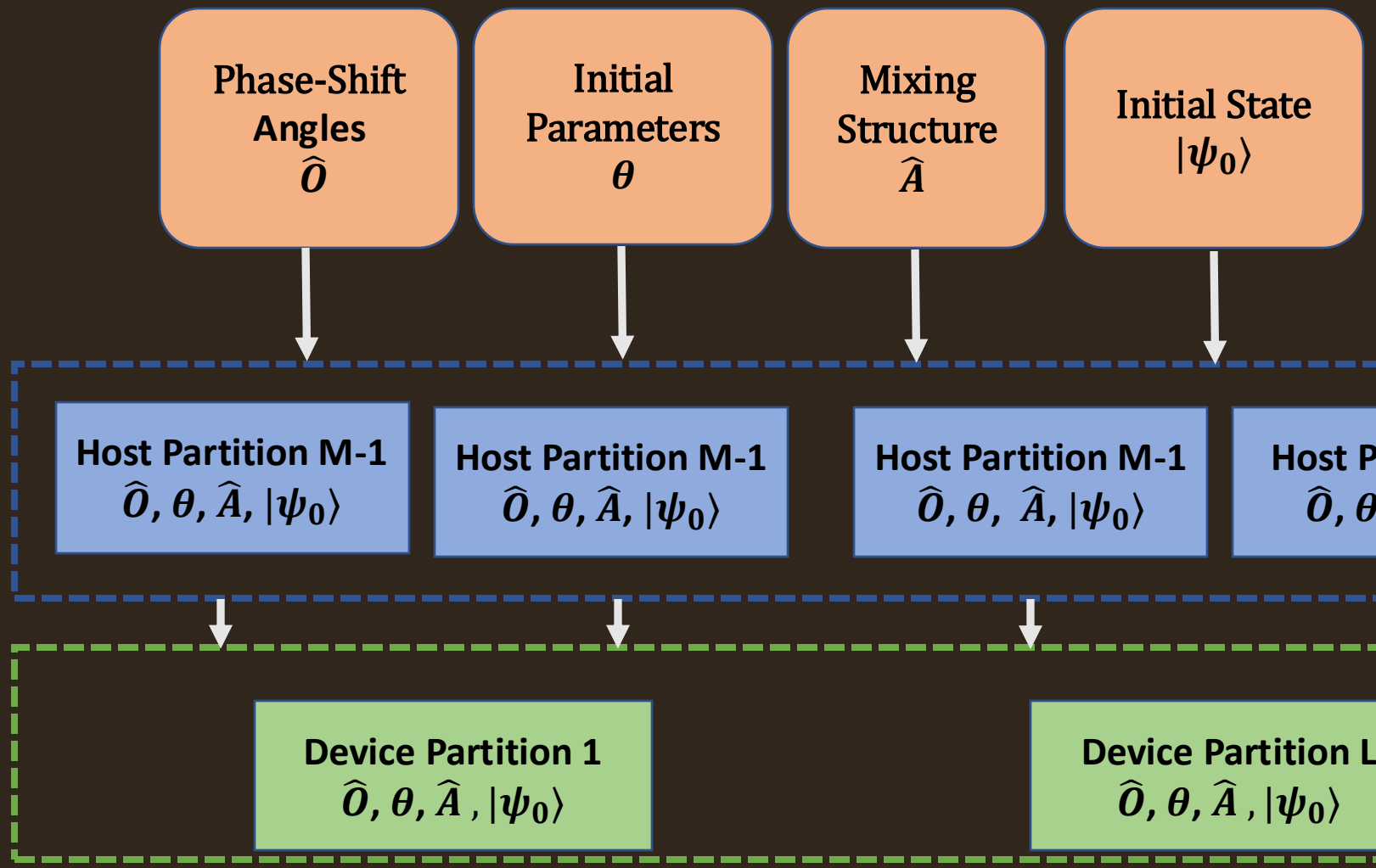
- Python Language
- User-definable
- Flexible and Dynamic
- Called in MPI-parallel

QuOp_Wavefront



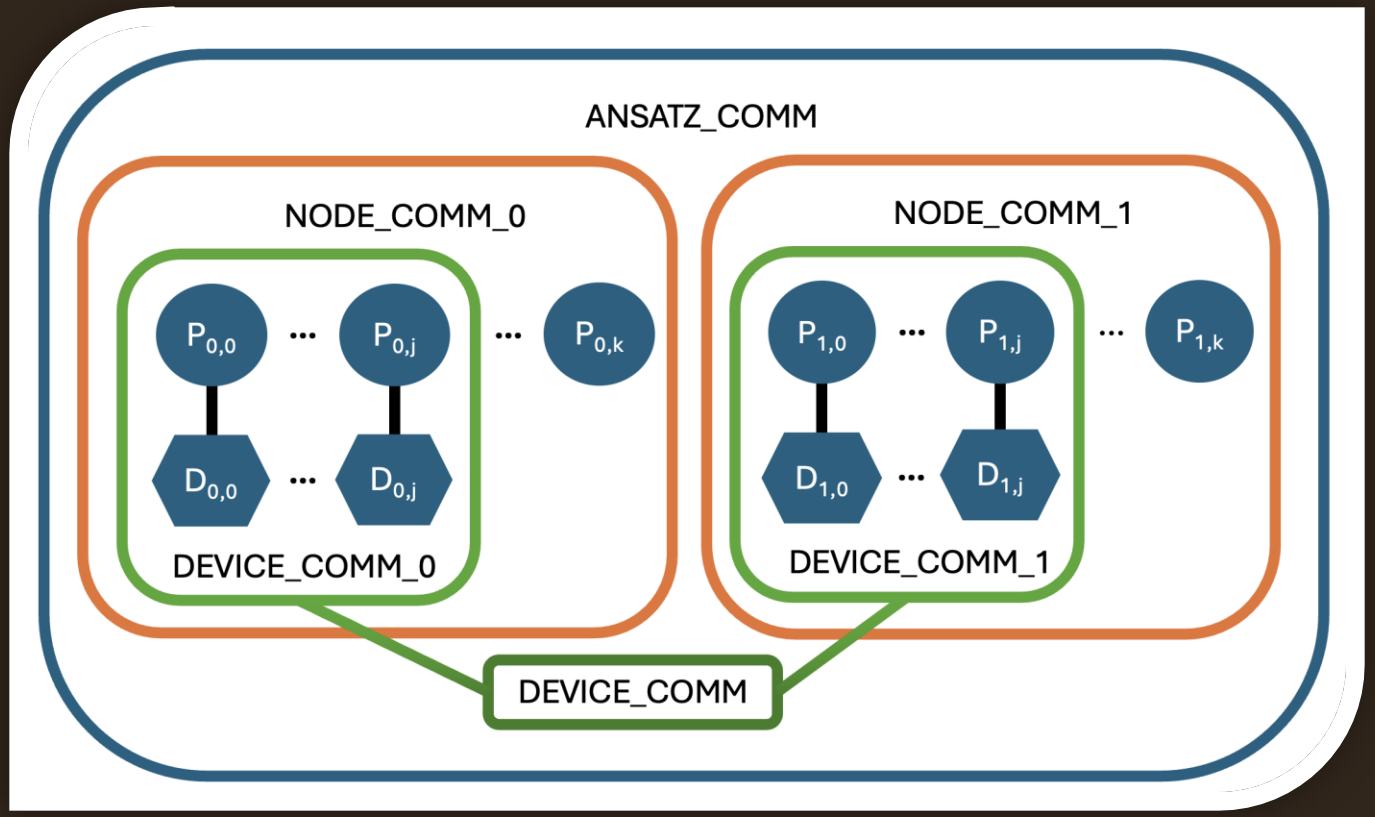
- QuOP_Wavefront brings GPU acceleration to QuOp_MPI.
- Uses HIP in Fortran and C++.
- Targets the latest generation of Cray systems equipped with AMD GPUs.
- Goal: Efficient and scalable GPU acceleration on distributed-memory systems.

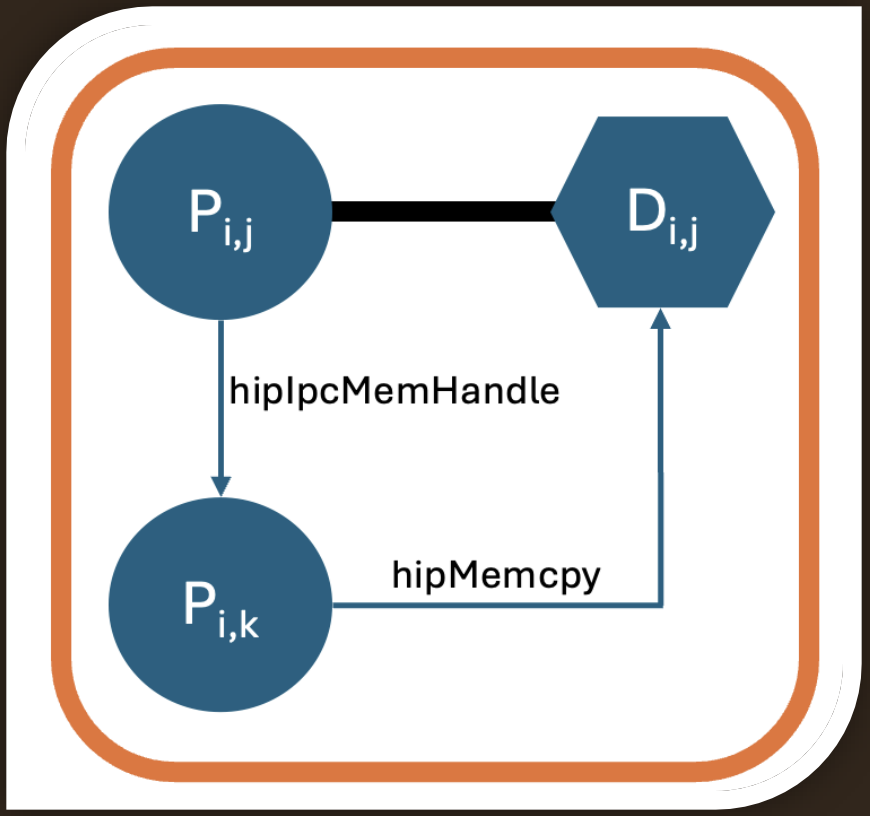
Communicator Structure



- Initial setup on device.
- Compute intensive steps on device.
- GPU to GPU MPI buffer transfers.

Communicator Structure





FFT-Based Simulation in High-Dimensions

In unconstrained combinatorial optimization, the solution space is described by:

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

A row-major ordering of the possible combinations.

A natural choice of mixing unitary is a tensor product of complete graphs:

$$H = K_0 \otimes \cdots \otimes K_{m-1}$$

H is the adjacency matrix of a Hamming graph on m-tuples, solutions are connected if they differ in exactly one of their possible values.

FFT-Based Simulation in High-Dimensions

The resulting mixing unitary is efficiently computed as a multidimensional FFT:

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

We are interested in studying problems where classical optimisation is hard, the potential for a quantum advantage increases with n.

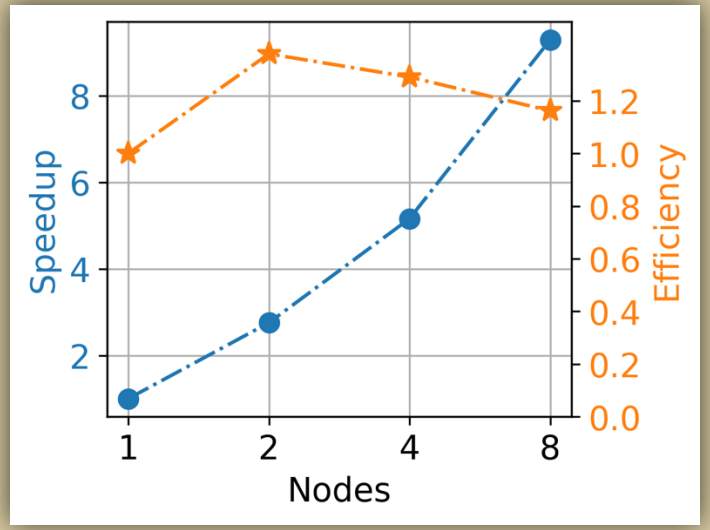
Available FFT libraries with MPI and HIP support natively support FFTs up n = 3.

FFT-Based Simulation in High-Dimensions

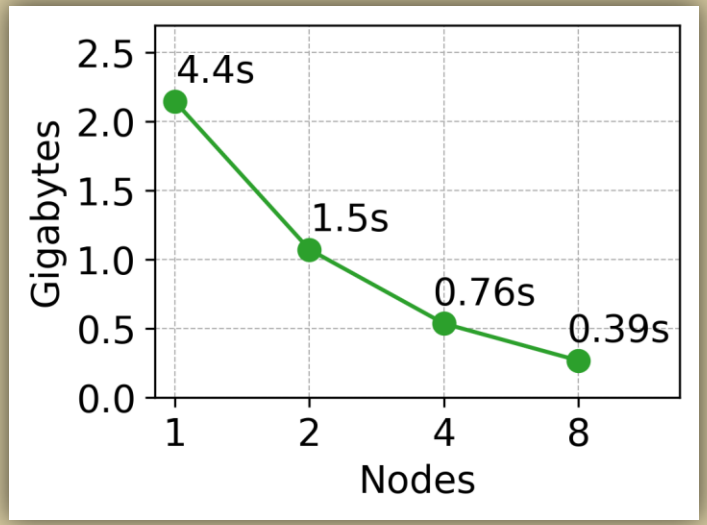
- To address this, we developed **SHAFFT** (Scalable High-dimensional Accelerated FFT), a C++ library with a Fortran interface.
- It implements a method developed by Dalcin et al. (2018)
- The state vector is distributed using a balanced, block-contiguous slab decomposition.
- Global redistributions are performed using **MPI_Alltoallw** with subarray datatypes, avoiding the need for local transpositions.
- Local transforms over contiguous dimensions are executed using **hipFFT**.

FFT-Based Simulation in High-Dimensions

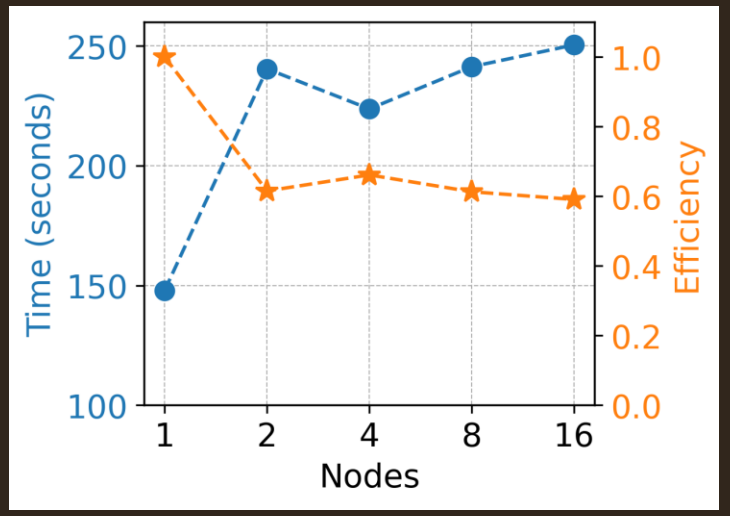
Strong Scaling



Subarray Size



Weak Scaling



- 16 MPI-processes and 8 GPUs per node.
- Two distributed coordinates, three batches of local FFTs and two redistributions.
- Strong scaling: n=3, size = $2^{10} \times 2^{10} \times 2^{10}$.
- Weak scaling: n=6, first $6 - \log_2(\text{Nodes})$ dimensions size 2^5, the remaining size 2^6.
- Node Configuration: AMD EPYC 7A53 64-Core GPU, eight AMD MI250X GPUs and 256 GB RAM.

Future Work

- Reordering of the global communicator to pair processes with GPUs in the same NUMA region
- Simulation of mixing unitaries with sparse unitaries
- Soon: Standalone release of SHAFFT