In bocca al lupo

Hiperwalk $|\langle\rangle$

Crepi il lupo!

Hiperwalk: A Python Package for Quantum Walk Simulations with HPC

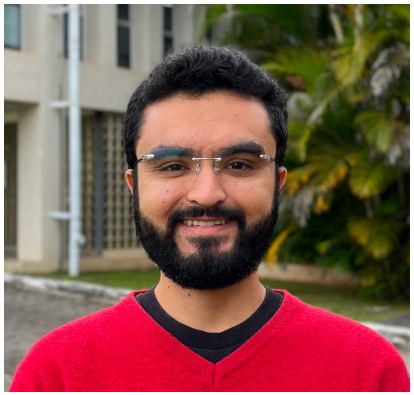
Developed at
National Laboratory of Scientific Computing – LNCC

QSQW 2025 – Napoli, Italia

Developers

Hiperwalk $|\langle\rangle$

Renato Portugal



Gustavo Bezerra

Paulo Motta

Anderson Santos

Newcomers:

Eduardo Garcia

Hiago Rocha

Introduction

Hiperwalk is a software package designed to simulate quantum walks on arbitrary graphs using High-Performance Computing (HPC).

Key Features:

- Package follows the best practices of software engineering
- Open-source and aimed at fostering community-driven development in the field of quantum walks
- Easy to install:
  - `pip install hiperwalk`
  - Download a Docker version for HPC applications.

Features of Hiperwalk

Python Frontend:

- User-friendly interface for defining graphs and quantum walks.
- Simulates continuous-time and coined quantum walks.
- Allows simulation on large graphs.
- Provides tools for visualizing and analyzing quantum walk dynamics.
- Open-source and extensible, encouraging community contributions.

Inner Core (HiperBLAS):

- C-based core for high-performance computing.
- Uses CPUs, GPUs, and accelerators for parallel processing.
- Leverages parallel devices for efficient linear algebra computations.
- Seamlessly integrates C components with Python.

Continuous-Time Quantum Walk:

- Evolution driven by a Hamiltonian:

$$H = -\gamma A - \sum_{v \in M} |v\rangle\langle v|$$

- γ is a positive parameter.
- Evolution operator: $U(t) = \mathrm{e}^{-\mathrm{i}Ht}$
- The state of the walk at time t:

$$|\psi(t)\rangle = U(t)|\psi(0)\rangle$$

- The probability of finding the walker on vertex v at time t:

$$\left|\langle v|\psi(t)\rangle\right|^2$$

Example: CTQW on a Cycle

Creating a quantum walk:

```
cycle = hpw.Cycle(101)
qw = hpw.ContinuousTime(cycle, gamma=1/(2*np.sqrt(2)), time=1)
```

Initial condition:

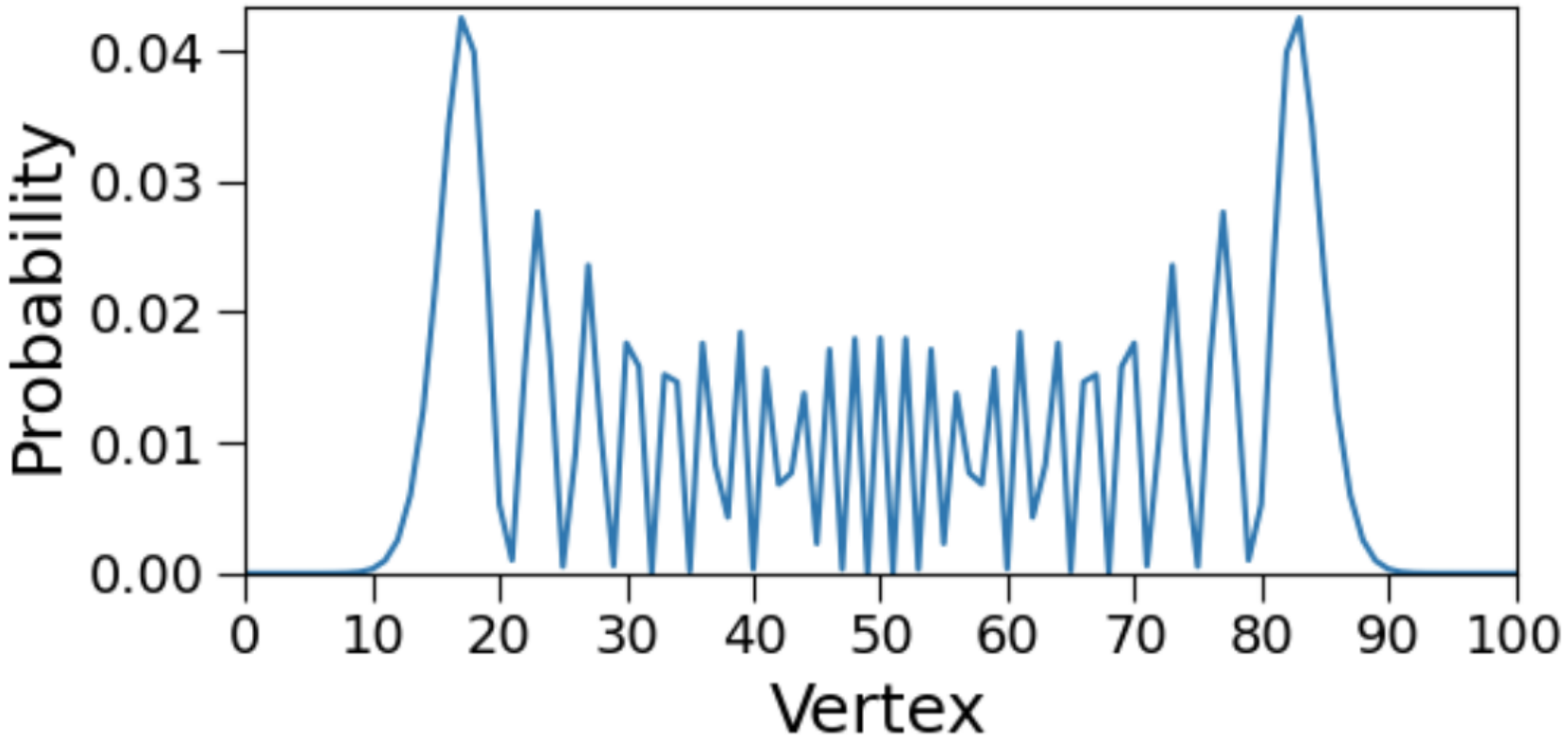
```
psi0 = qw.ket(50)
```

Simulation:

```
psi = qw.simulate(range=(50,51), state=psi0)
prob = qw.probability_distribution(psi)
hpw.plot_probability_distribution(prob, plot='line')
```

Example: CTQW on a Cycle

```
psi_final = qw.simulate(range=(50,51), state=psi0)
prob = qw.probability_distribution(psi_final)
hpw.plot_probability_distribution(prob, plot='line', figsize=(6,3))
```



Coined Quantum Walk Model

- Evolution operator: $U = S\,C$.
- Flip-flop shift operator $S|v, w\rangle = |w, v\rangle$.
- The Grover coin:

$$C|v, w\rangle = \sum_{v' \in N(v)} \left(\frac{2}{d(v)} - \delta_{w,v'} \right) |v, v'\rangle$$

- The probability of finding the walker on a vertex v after t steps:

$$p_v(t) = \sum_{w \in N(v)} \left| \langle v, w | \psi(t) \rangle \right|^2$$

Example: Grover Walk on 2D Grid

Creating a quantum walk:

```
grid = hpw.Grid(50, diagonal=True, periodic=True)
qw = hpw.Coined(grid, shift='persistent', coin='grover')
```

Initial condition:

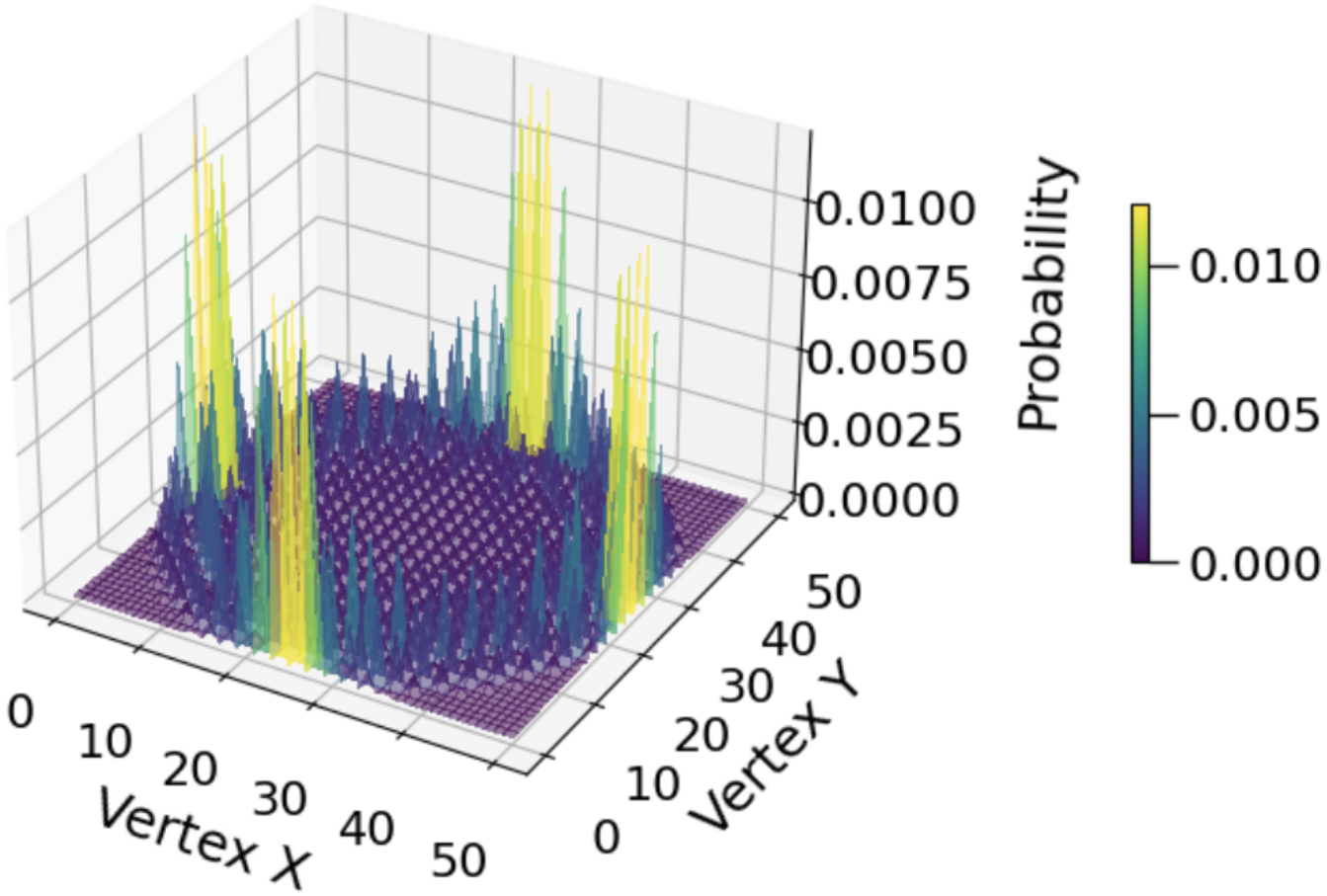
```
center = np.array((25,25))
psi0 = qw.state([(0.5, (center, center + (1, 1))),
                 (-0.5, (center, center + (1, -1))),
                 (-0.5, (center, center + (-1, 1))),
                 (0.5, (center, center + (-1, -1)))])
```

Simulation:

```
psi = qw.simulate(range=(25,26), state=psi0)
prob = qw.probability_distribution(psi)
hpw.plot_probability_distribution(prob, graph=grid)
```

Example: Grover Walk on 2D Grid

```
psi_final = qw.simulate(range=(n//2,n//2+1), state=psi0)
prob = qw.probability_distribution(psi_final)
hpw.plot_probability_distribution(prob, graph=grid, figsize=(9,4.5))
```

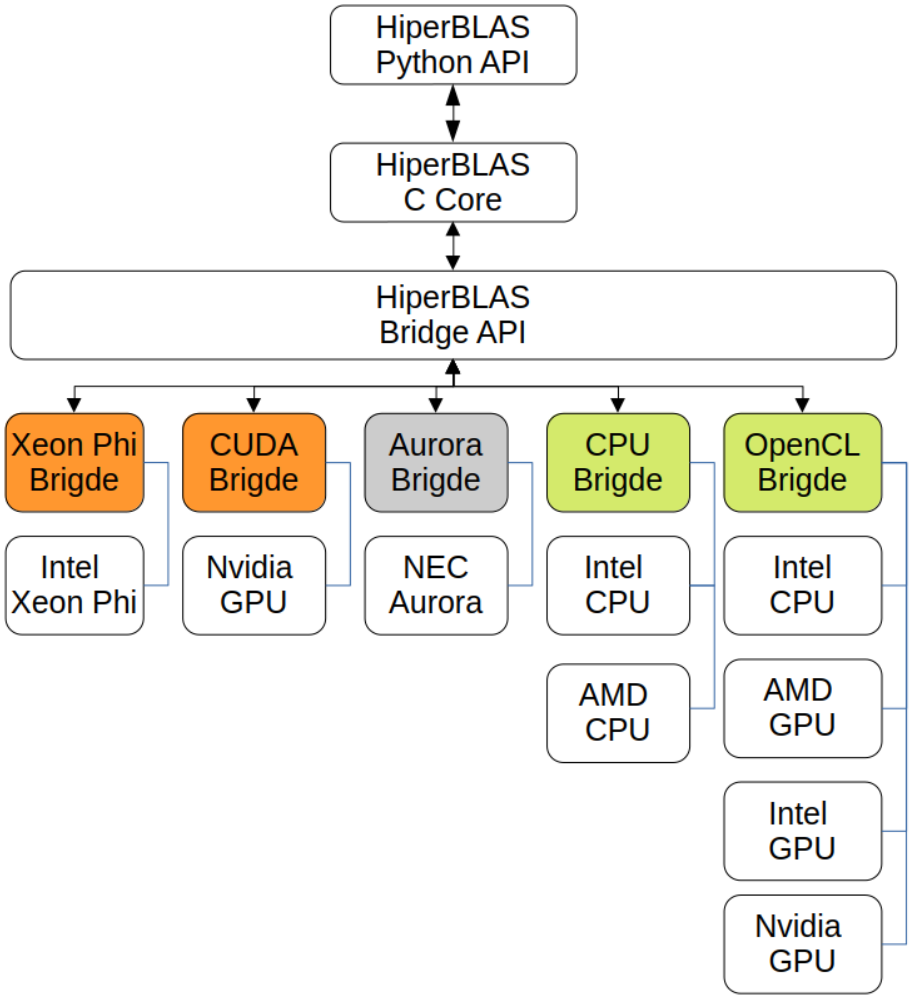


Features of Hiperwalk for Quantum Walk Applications

- Simulation of quantum walks on arbitrary graphs.
- Graphs can be defined using NetworkX.
- Plotting probability distribution.
- Animation of quantum walks.
- Improved display for known graphs.
- Allow marked vertices.
- Calculates optimal runtime.

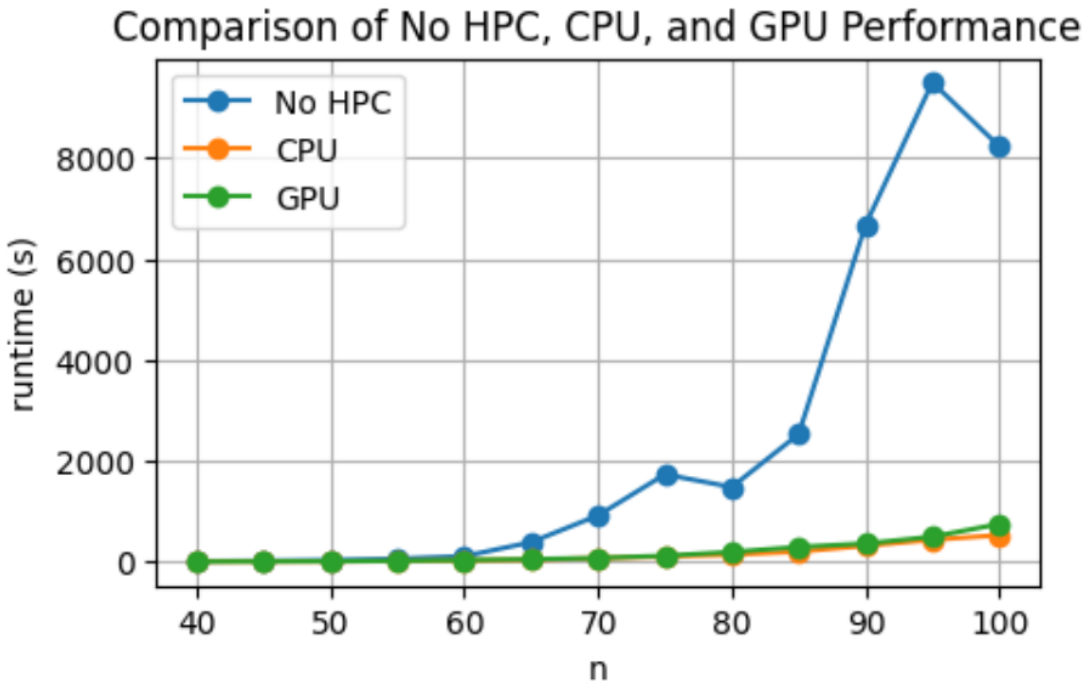
System Architecture

- **Python Frontend and C-Based HiperBLAS**



Benchmarks and Performance

- Showed scalability with increasing graph sizes.
- Effective utilization of GPUs and other parallel devices.
- Reduced computational time for large-scale simulations.



Running time versus n on the Johnson graph $J(n, 2)$

Future Work

Python Frontend:

- Implementation of more quantum walk models (e.g., Staggered, Szegedy, OpenQW, and others).
- Support for additional graph types (e.g., Bipartite, Johnson graphs, and others).
- Enhanced visualization tools.
- Implementation of mixing times.
- Implementation of Perfect State Transfer (PST).
- Decomposition into universal quantum gates.

Inner Core:

- Multi-GPU support with advanced memory management.
- Heterogeneous HPC support.

Conclusion

- Hiperwalk provides a robust platform for simulating quantum walks.
- Combines ease of use with computational efficiency.
- Open-source nature fosters community-driven development.
- Future updates aim to expand capabilities and applications.

Explore Hiperwalk: https://hiperwalk.org

Reference: Quantum Week 2023
IEEE International Conf. on Quantum Computing and Eng. (QCE)

Hiperwalk $|\langle\rangle$

THE END – THANK YOU

https://hiperwalk.org

GitHub:

https://github.com/hiperwalk

https://github.com/hiperblas

Contact:

hiperwalk@gmail.com