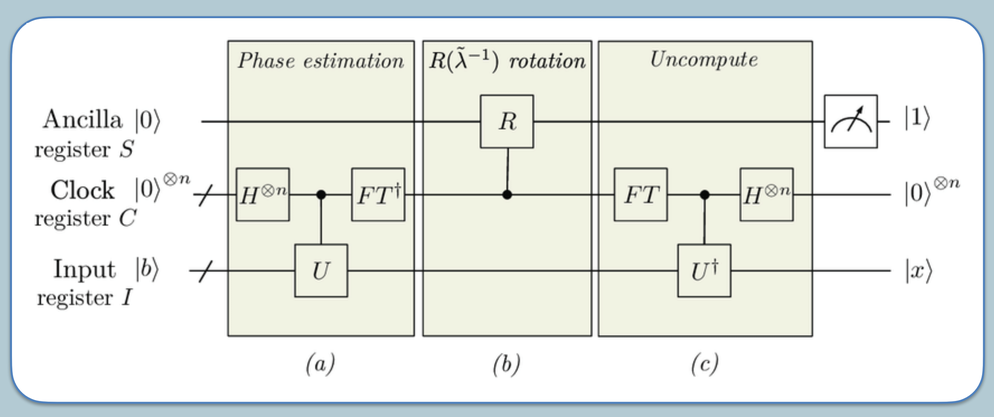
Solving linear systems with HHL quantum algorithm on GPUs

Dhruv Sood[1], Manish Modani[2], Nilmani Mathur[1], Vikram Tripathi[1], Andreas Hehn[2]
[1]Department of Theoretical Physics, Tata Institute of Fundamental Research, Mumbai;
[2]NVIDIA India; [3]NVIDIA Switzerland.

Introduction

- Linear Systems of equations are ubiquitous in all fields of science and often arise as intermediary steps in studies of physical systems. Classical methods for such problems take time that scales polynomially with the problem size.
- The HHL algorithm [1] is a quantum algorithm that is able to solve such systems in time that that scales logarithmically with the size. Below is a schematic diagram of the quantum circuit corresponding to it.
- Thus the HHL algorithm can be used to study physical systems more efficiently once satisfactory quantum hardware is fabricated. We aim to use this and other such algorithms for studying quantum many body systems, which are often challenging to study via classical means.
- Quantum simulation can be used to test and refine the implementation of the algorithm, as well as optimise it for relevant applications.



- Further, NVIDIA's cuQuantum SDK allows us to speed up such simulations. It has been demonstrated to provide significant speedups to routine like Shor's Algorithm, the Quantum Fourier Transform, etc. [2,3] to create and we believe it will allow us to better explore the capabilities for when suitable quantum hardware is available.
- We thus chose to use cuQuantum to accelerate the simulations of the HHL algorithm with the goal of optimising it and making it scalable to higher system sizes and number of qubits.
- In this poster, we present results regarding the implementation of the HHL algorithm and the speedups gained by GPU enablement via cuQuantum.

A Test : Diagonal Matrices

We have tested the HHL algorithm for various matrix sizes and complexities. All runs were conducted first using only CPU capabilities, and then using V100 and A100 NVIDIA GPUs - where we expect the cuQuantum SDK to speed up the simulation of the quantum circuit.

We begin with systems where the matrix A is diagonal with entries corresponding to binary fractions upto a multiplicative constant. Such matrices were used to verify the that the implementation was satisfactory and that cuQuantum was able to use GPUs to speedup the runs.. A table of times taken for such runs is presented below.

System Size	Total Qubits	CPU Time (seconds)	GPU Time (V100) (seconds)	Speedup (%)	GPU Time (A100) (seconds)	Speedup (%)
2	4	5.773	5.231	9.3	5.657	2.0
4	8	11.362	11.022	2.9	10.943	3.6
8	8	7.797	7.638	2.0	7.548	3.2
8	10	30.125	29.102	3.3	28.852	4.2
16	10	14.546	14.008	3.6	13.804	5.1
16	12	113.984	109.880	3.6	107.748	5.4
32	12	116.910	112.585	3.7	109.253	6.5

This serves as a demonstration that the HHL algorithm has been implemented correctly and cuQuantum has been correctly integrated. We now move on to matrices where the eigenvalues are not binary fractions and see how the algorithm fares there.

Irrational Eigenvalues

After establishing the viability of the implementation and demonstrating the small but visible speedup achieved using cuQuantum, we moved onto testing it on diagonal matrices where the eigenvalues are not handpicked, but generated randomly.

This is expected to interfere with the phase estimation routine in the HHL algorithm, increasing the circuit depth and hence the time taken by the simulation. A table detailing the times taken in such runs is presented opposite.

System Size	Total Qubits	CPU Time (seconds)	GPU Time (V100) (seconds)	Speedup (%)	GPU Time (A100) (seconds)	Speedup (%)
2	4	5.818	6.002	--	5.713	1.8
4	8	11.870	11.596	2.3	11.454	3.5
8	8	7.583	7.222	4.7	7.325	3.4
8	10	30.232	29.767	1.5	29.204	3.4
16	10	14.823	14.549	1.8	14.111	4.8
16	12	115.622	113.542	1.8	109.030	5.7

Thus cuQuantum is able to speed up the simulation by upto 5.7%. We can further see that this speedup increases as we increase the size of the problem, which we attribute to the fact that at lower qubits the problem does not utilise all available cores. Indeed it seems that for the smallest system, the CPU run is faster than GPU-enabled runs.

On an average we find a speedup of 2.42% using V100 GPUs and 4.52% using A100 GPUs. Another characteristic evident from the date is that while the speedup attained via V100 GPUs appears to stagnate, the A100 speedup appears to still be rising. We will explore this in future work and look at how the speedup scales with system size and number of qubits.

A Test : Diagonal Matrices

We now move on to working with matrices where the eigenvalues and elements are generated randomly. For such matrices, we start with a diagonal matrix with random entries and then perform the random rotation to make it non-sparse.

In such runs the circuit depth has the potential to grow rapidly and take a lot of time. These runs serve to verify that the implementation is viable. The times obtained for these runs are tabulated below.

System Size	Total Qubits	CPU Time (seconds)	GPU Time (V100) (seconds)	Speedup (%)	GPU Time (A100) (seconds)	Speedup (%)
2	4	5.520	5.625	--	5.618	--
4	8	17.552	17.218	1.9	16.886	3.8
8	10	103.294	100.602	2.6	98.542	4.6
16	10	158.302	153.713	2.9	150.228	5.1
16	12	753.482	727.864	3.4	703.844	6.7
32	12	1463.598	1412.370	3.5	1358.222	7.2

Again we see that a significant speedup is achieved in GPU-enable runs. Indeed, the average speedups of x% for V100 and x% for A100 are both higher their counterparts in the previous section. This is like due to the increased complexity of the matrix and the higher circuit depth demands more system resources and a GPU-enabled run is better at coping with that. We aim to pursue this effect as well in future work. The scaling of the speedup still follows the same pattern as above, and can be attributed to the same reasons.

These results demonstrate that one can try and solve actual problems upto a matrix size of 16 X 16. While there are some problems that can be explored at this size, we believe it is possible to further optimise both the implementation of the HHL algorithm and the compatibility with cuQuantum to obtain better speeds and scale these results further. Once that is achieved, we plan to try and apply this to studying physical models like the SSH Model, the Toric Code, etc.

Conclusions

- We have implemented the HHL algorithm and successfully integrated it with NVIDIA's cuQuantum SDK.
- We have demonstrated speedups in the quantum simulation of the HHL algorithm using cuQuantum for sparse matrices.
- We have found that the percentage speedups increase with system size, a fact that we will exploit in increasing the scalability of the implementation.
- Our implementation of the HHL algorithm is so far compatible with nearly sparse hermitian positive definite matrices with random eigenvalues.
- We plan to optimise it further to extend it's compatibility to dense matrices that possess highly entangled eigensystems.
- With this, we envisage to study entangled quantum many body systems in the near future, which are otherwise difficult to investigate.

References

1. Harrow, A. W., Hassidim, A., & Lloyd, S. (2009). Quantum Algorithm for Linear Systems of Equations. Phys. Rev. Lett., 103150502.
2. M. Modani, A. Banerjee and A. Das, "Performance analysis of quantum algorithms on Param Siddhi-AI system," 2022 International Conference on Trends in Quantum Computing and Emerging Business Technologies (TQCEBT), Pune, India, 2022.
3. Modani, M., Banerjee, A., Das, A. (2023). Multi-GPU-Enabled Quantum Circuit Simulations on HPC-AI System. In: Sharma, N., Goje, A., Chakrabarti, A., Bruckstein, A.M. (eds) Data Management, Analytics and Innovation. ICDMAI 2023. Lecture Notes in Networks and Systems, vol 662. Springer, Singapore.