

ĐẠI HỌC QUỐC GIA TP.HCM
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA KHOA HỌC VÀ KỸ THUẬT MÁY TÍNH



BÁO CÁO
ĐỒ ÁN TỐT NGHIỆP

NGHIÊN CỨU VÀ PHÁT TRIỂN GIẢI PHÁP ĐỊNH
THỜI TÀI NGUYÊN CHO TÍNH TOÁN LƯỢNG TỬ

Ngành: KỸ THUẬT MÁY TÍNH

HỘI ĐỒNG : KỸ THUẬT MÁY TÍNH 7

GVHD : PGS. TS. THOẠI NAM

: ThS. HOÀNG LÊ HẢI THANH

: TS. DIỆP THANH ĐĂNG

GVPB : ThS. NGUYỄN PHƯƠNG DUY

—————o0o—————

SVTH 1 : NGUYỄN XUÂN TRIỀU (2110610)

SVTH 2 : VÕ TRẦN NHÃ LINH (2111654)

TP. HỒ CHÍ MINH, 05/2025

ĐẠI HỌC QUỐC GIA TP.HCM
TRƯỜNG ĐẠI HỌC BÁCH KHOA

CỘNG HÒA XÃ HỘI CHỦ NGHĨA VIỆT NAM
Độc lập - Tự do - Hạnh phúc

KHOA: KH & KT Máy tính
BỘ MÔN: HT & MMT

NHIỆM VỤ LUẬN VĂN/ ĐỒ ÁN TỐT NGHIỆP
Chú ý: Sinh viên phải dán tờ này vào trang nhất của bản thuyết trình

HỌ VÀ TÊN: NGUYỄN XUÂN TRIỀU
HỌ VÀ TÊN: VÕ TRẦN NHẢ LINH
NGÀNH: KỸ THUẬT MÁY TÍNH

MSSV: 2110610
MSSV: 2111654
LỚP:

1. Đầu đề luận văn/ đồ án tốt nghiệp:

- Tiếng Việt: Nghiên cứu và phát triển giải pháp định thời tài nguyên cho tính toán lượng tử
- Tiếng Anh: Studying and developing resource scheduling for quantum computing

2. Nhiệm vụ (yêu cầu về nội dung và số liệu ban đầu):

- Triển khai các giải pháp định thời được chọn trên hệ mô phỏng lượng tử.
- Thiết lập môi trường đánh giá các giải pháp định thời.
- Xác định các tiêu chí đánh giá.
- Đề xuất các kịch bản đánh giá.
- Tiến hành đánh giá các giải pháp định thời.
- Đề xuất giải thuật định thời mới cho tính toán lượng tử (bonus).

3. Ngày giao nhiệm vụ: 06/01/2025

4. Ngày hoàn thành nhiệm vụ: 25/05/2025

5. Họ tên giảng viên hướng dẫn:

- 1) PGS. TS. Thoại Nam
- 2) ThS. Hoàng Lê Hải Thanh
- 3) TS. Diệp Thanh Đăng

Phản hướng dẫn:

Thảo luận
Thảo luận
Thảo luận, sửa lỗi trình bày

Nội dung và yêu cầu LVTN/ ĐATN đã được thông qua Bộ môn.

Ngày tháng năm
CHỦ NHIỆM BỘ MÔN
(Ký và ghi rõ họ tên)


Nguyễn Đức Thái

PHÂN DÀNH CHO KHOA, BỘ MÔN:

Người duyệt (chấm sơ bộ):


Đơn vị:

Ngày bảo vệ:

Điểm tổng kết:

Nơi lưu trữ LVTN/ĐATN:

GIẢNG VIÊN HƯỚNG DẪN CHÍNH
(Ký và ghi rõ họ tên)


Diệp Thanh Đăng

Ngày 15 tháng 05 năm 2025

PHIẾU ĐÁNH GIÁ LUẬN VĂN/ ĐỒ ÁN TỐT NGHIỆP
(Dành cho người hướng dẫn/phản biện)

1. Họ và tên SV: Nguyễn Xuân Triều, Võ Trần Nhã Linh
MSSV: 2110610, 2111654 Ngành (chuyên ngành): Kỹ Thuật Máy Tính
2. Đề tài: Nghiên cứu và phát triển giải pháp định thời tài nguyên cho tính toán lượng tử

3. Họ tên người hướng dẫn: Diệp Thanh Đăng

4. Tổng quát về bản thuyết minh:

Số trang: 134

Số chương: 8

Số bảng số liệu: 15

Số hình vẽ: 44

Số tài liệu tham khảo: 49

Phần mềm tính toán: 0

Hiện vật (sản phẩm): 0

5. Những ưu điểm chính của LV/ĐATN:

- Xây dựng môi trường thực nghiệm có khả năng mô phỏng và đánh giá giải thuật lập lịch lượng tử.
- Thiết lập bộ tiêu chí khá đầy đủ (bao gồm 8) cho việc đánh giá và so sánh các giải thuật lượng tử.
- Cung cấp các so sánh và phân tích kết quả thực nghiệm của một vài giải thuật tính toán lượng tử phổ biến.
- Báo cáo ĐATN khá chi tiết.

6. Những thiếu sót chính của LV/ĐATN:

- Các kết quả thực nghiệm vẫn chưa được trình bày tốt.
- Một vài giải thuật lập lịch lượng tử liên quan khác vẫn chưa được xem xét.

7. Đề nghị: Được bảo vệ ☒ Bổ sung thêm để bảo vệ ☐ Không được bảo vệ ☐

8. Các câu hỏi SV phải trả lời trước Hội đồng:

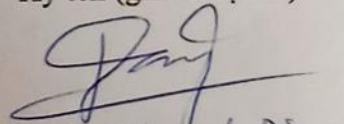
a. Thách thức chính của ĐATN này là gì?

b. Sự khác nhau giữa multithreading và circuit knitting là gì?

c. Nêu một giải thuật lập lịch lượng tử tiềm năng mà SV chưa xem xét trong ĐATN này?

9. Đánh giá chung (bằng chữ: Xuất sắc, Giỏi, Khá, TB): Xuất sắc Điểm: 9.1/10

Ký tên (ghi rõ họ tên)


Diệp Thanh Đăng

TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA KH & KT MÁY TÍNH

CỘNG HÒA XÃ HỘI CHỦ NGHĨA VIỆT NAM
Độc lập - Tự do - Hạnh phúc

Ngày 16 tháng 5 năm 2025

PHIẾU ĐÁNH GIÁ LUẬN VĂN/ ĐỒ ÁN TỐT NGHIỆP
(Dành cho người hướng dẫn/phản biện)

1. Họ và tên SV: Võ Trần Nhã Linh

MSSV: 2111654

Ngành (chuyên ngành): Kỹ thuật máy tính

Họ và tên SV: Nguyễn Xuân Triều

MSSV: 2110610

Ngành (chuyên ngành): Kỹ thuật máy tính

2. Đề tài: Nghiên cứu và phát triển giải pháp định thời tài nguyên cho tính toán lượng tử
(Studying and developing resource scheduling for quantum computing)

3. Họ tên người hướng dẫn/phản biện: Nguyễn Phương Duy

4. Tổng quát về bản thuyết minh:

Số trang: 123

Số chương: 8

Số bảng số liệu: 15

Số hình vẽ: 44

Số tài liệu tham khảo: 49

Phần mềm tính toán:

Hiện vật (sản phẩm)

5. Những ưu điểm chính của LV/ĐATN:

Nhóm sinh viên thực hiện các nghiên cứu có ý nghĩa đi đầu tạo một hệ thống tính toán lượng tử và triển khai các giải pháp định thời tài nguyên cho hệ thống tự xây dựng (dựa trên một hệ thống có sẵn nhưng chưa hoàn chỉnh).

Đóng góp của sinh viên hoàn thiện các giải pháp khác nhau và tích hợp chung vào một hệ thống từ đó đưa ra các đánh giá đo đạc dựa trên thực nghiệm, bao gồm các hiện thực:

- Hệ thống thực hiện nhiều giải pháp định thời khác nhau: MILQ, SCIM MILQ, NoTADS, MTMC
- Hệ thống thực hiện hai giải pháp hiện thực hệ thống lượng tử: CutQC và mở rộng QOS

Kết quả của đề tài là cả hai hệ thống đều có ý nghĩa thực tế và xây dựng nền tảng cho hệ thống tính toán có tính mới là hệ thống máy tính lượng tử.

6. Những thiếu sót chính của LV/ĐATN:

Sinh viên chưa làm rõ đóng góp của luận văn là hai kết quả riêng biệt về hiện thực hệ thống lượng tử và hiện thực các giải thuật định thời.

- Nếu hệ thống chỉ đánh giá các giải thuật định thời thì phần hiện thực hệ thống nên là một mô tả testbed thí nghiệm.
- Nếu hệ thống có cả hai kết quả thì nội dung hiện thực hệ thống chưa được đánh giá đầy đủ các kết quả của phép cắt CutQC và có nhiều giải pháp hệ thống lượng tử ngoài QOS, nên có một so sánh hệ thống của đề tài đã hiện thực với các hệ thống hiện có

Sinh viên có hiện thực nhầm lẫn cơ chế song song trong giải thuật MTMC cần kiểm tra lại.

Sinh viên có sai sót trong định nghĩa các tiêu chí định thời dẫn đến kết quả so sánh giữa giải pháp MTMC và NoTADS chưa phù hợp với đặc điểm so sánh giữa hai lời giải sử dụng bộ giải tối ưu (optimization solver) và giải pháp heuristic.

7. Đề nghị: Được bảo vệ [x] Bổ sung thêm đề bảo vệ ○

Không được bảo vệ ○


8. Các câu hỏi SV phải trả lời trước Hội đồng:

a. Sinh viên có thể đánh giá các đóng góp trong việc xây dựng hệ thống lượng tử ở các nội dung CurQC và QOS?

b. Sinh viên có thể cập nhật kết quả so sánh MTMC có tốt hơn MILQ sau khi đã thực hiện kiểm tra và hiệu chỉnh các hiện thực của các giải thuật?

9. Đánh giá chung (bằng chữ: Xuất sắc, Giỏi, Khá, TB): Giỏi Điểm : 9.1 /10

Ký tên (ghi rõ họ tên)



Nguyễn Phương Dao

Lời cam đoan

Nhóm nghiên cứu tuyên bố rằng nhóm chỉ thực hiện nghiên cứu này, dưới sự hướng dẫn của PGS.TS. Thoại Nam, Khoa Khoa học máy tính và Kỹ thuật, Đại học Quốc gia thành phố Hồ Chí Minh - Trường Đại học Bách Khoa.

Nhóm đã cẩn thận ghi nhận và ghi chú đầy đủ tất cả các nguồn và tài liệu tham khảo bên ngoài được sử dụng trong dự án.

Nếu có bất kỳ dấu hiệu đạo văn nào, nhóm sẵn sàng chấp nhận hậu quả.

Đại học Bách khoa Thành phố Hồ Chí Minh - Đại học Quốc gia Việt Nam TP.HCM sẽ không chịu trách nhiệm về bất kỳ hành vi vi phạm bản quyền nào có thể xảy ra trong quá trình nghiên cứu của tôi.

Lời cảm ơn

Nhóm tác giả xin chân thành cảm ơn tất cả mọi người đã hỗ trợ cho đề tài nghiên cứu này.

Đầu tiên và trước hết, nhóm xin được gửi lời cảm ơn sâu sắc nhất tới thầy Thoại Nam, thầy Diệp Thanh Đăng và thầy Hoàng Lê Hải Thanh đã hướng dẫn tận tình trong thời gian qua. Những góp ý và kinh nghiệm từ các thầy đã định hướng và dẫn dắt đề tài của nhóm.

Tiếp theo, nhóm xin được phép cảm ơn Trường Đại học Bách Khoa, Đại học Quốc gia TP.HCM và Khoa Khoa học và Kỹ thuật Máy tính đã cung cấp tài nguyên và môi trường thích hợp cho đề tài này.

Cuối cùng, nhóm xin biết ơn gia đình và bạn bè của cả hai vì sự hỗ trợ và sát cánh tận tình.

Cảm ơn tất cả mọi người.

Tóm tắt

Định thời tài nguyên lượng tử hiệu quả là một nhiệm vụ cần thiết trong bối cảnh thiết bị lượng tử đang còn hạn chế về tài nguyên và hiệu suất. Hiện nay đã và đang có rất nhiều nghiên cứu về đề tài này, từ việc xây dựng một trình định thời cho tới xây dựng một hệ điều hành lượng tử. Tuy vậy, vẫn chưa có nghiên cứu nào hỗ trợ so sánh các giải thuật định thời tài nguyên lượng tử hay đưa ra các đánh giá, so sánh các nghiên cứu này với nhau.

Luận văn này tập trung vào hai nhóm nhiệm vụ chính, bao gồm: xây dựng một nền tảng cho phép mô phỏng các giải thuật định thời trong môi trường lượng tử; đồng thời đưa ra những so sánh, đánh giá đối với một số nghiên cứu định thời lượng tử trước đó dựa trên một số tiêu chí chọn lọc.

Thông qua đề tài này, người đọc sẽ có một cái nhìn chi tiết hơn về các giải thuật định thời tài nguyên lượng tử cũng như các nghiên cứu về đề tài này hiện nay. Công cụ mô phỏng mà nhóm đưa ra cung cấp khả năng tùy linh hoạt để người dùng có thể thực hiện so sánh hay có cái nhìn cụ thể hơn về giải thuật định thời tài nguyên lượng tử sử dụng.

Mục lục

1	Giới thiệu	1
1.1	Tổng quan	1
1.2	Lý do chọn đề tài	2
1.3	Mục tiêu đề tài	2
1.4	Phạm vi đề tài	3
1.4.1	Câu hỏi nghiên cứu	3
1.4.2	Giới hạn phạm vi	4
1.5	Đóng góp đề tài	4
1.6	Cấu trúc đề tài	5
2	Cơ sở lí thuyết	7
2.1	Tính toán lượng tử	7
2.1.1	Các tính chất cơ bản của hệ lượng tử	7
2.1.2	Qubit và các phép toán cơ bản	9
2.1.3	Cổng lượng tử	11
2.1.4	Mạch lượng tử (Quantum Circuit)	12
2.2	Dịch vụ lượng tử	14
2.2.1	Mô hình triển khai	14
2.2.2	So sánh tổng quan	18
2.3	Định thời	19
2.4	Kỹ thuật xử lý mạch lượng tử với tài nguyên hạn chế	22
2.4.1	Cắt mạch	22
2.4.2	Ghép mạch	27
2.4.3	Tổng hợp các giải pháp	29
2.4.4	Ứng dụng thực tế	29

2.5	Simulator và Emulator	30
2.5.1	Định nghĩa và phân biệt	30
2.5.2	Vai trò và ứng dụng	31
2.5.3	Mối quan hệ với phần cứng lượng tử	32
2.6	Thư viện mô phỏng lượng tử	32
2.6.1	Giới thiệu	32
2.6.2	Các thư viện mô phỏng lượng tử phổ biến	32
2.6.3	Sự lựa chọn	34
2.7	Tóm tắt chương	35
3	Nghiên cứu liên quan	36
3.1	MILQ - Multithreaded Parallelism for Heterogeneous Clusters of QPUs	36
3.2	SCIM MILQ	37
3.3	CutQC	38
3.4	QOS - Quantum Operating System	39
3.5	Bộ định thời phân bổ nhận biết độ nhiễu và thời gian (NoTADS)	40
3.6	MTMC - Multi-Task Multi-Chip Scheduling	42
3.7	Định thời nhận biết tài nguyên (Resource-aware scheduling)	43
3.8	Bảng tổng hợp các công trình nghiên cứu	44
3.9	Tóm tắt chương	46
4	Kiến trúc hệ thống	48
4.1	Kiến trúc tổng quan	48
4.2	Mô hình chi tiết	49
4.2.1	Input (Đầu vào)	50
4.2.2	Emulator Engine (Bộ máy mô phỏng)	52
4.2.3	Output (Kết quả đầu ra)	56
4.2.4	Visualization Module (Mô-đun trực quan hóa)	61
4.3	Workflow hệ thống	62
4.4	Tóm tắt chương	71
5	Tiêu chí đánh giá	72
5.1	Các tiêu chí đánh giá	72

5.1.1	Độ trung thực	73
5.1.2	Throughput	75
5.1.3	Response time	77
5.1.4	Turnaround time	78
5.1.5	Makespan	79
5.1.6	Scheduling latency	80
5.1.7	Machine Utilization	81
5.1.8	Cut Sampling Overhead	82
5.2	Tóm tắt chương	83
6	Giải thuật định thời lượng tử	84
6.1	Tổng quan	84
6.2	Mô tả giải thuật	85
6.2.1	First-Fit Decreasing Bin Packing	85
6.2.2	MILP (Mixed Integer Linear Programming)	87
6.2.3	Noise and Time Aware Distributed Scheduler (NoTADS)	93
6.2.4	Multi-Task Multi-Chip Scheduling (MTMC)	95
6.3	Bảng so sánh	100
6.4	Tóm tắt chương	100
7	Đánh giá giải thuật	101
7.1	Mô tả hệ thống thực nghiệm	101
7.2	Mô tả benchmark thực nghiệm	101
7.3	Đánh giá mối liên hệ đầu vào - chỉ số	102
7.4	Thử nghiệm	103
7.5	Kết quả đánh giá	106
7.6	Phân tích kết quả	106
7.7	Đánh giá tổng quan	107
7.8	Tóm tắt chương	108
8	Tổng kết	109
8.1	Kết luận	109
8.2	Hướng phát triển	110

Tài liệu tham khảo	111
Phụ lục A: Thiết lập môi trường	116
8.3 Tải Về và Kiểm Tra Công Cụ	117
8.4 Ví Dụ Sử Dụng Cơ Bản	117
Phụ lục B: Các công lượng tử	119

Danh sách hình vẽ

Hình 2.1	Trực quan hoá trạng thái qubit trên hình cầu Bloch. Nguồn: Smite-Meister, CC BY-SA 3.0 , via Wikimedia Commons.	10
Hình 2.2	Mạch lượng tử Quantum Fourier Transform (QFT) với 3 qubit.	13
Hình 2.3	Mạch lượng tử trong thuật toán Grover với 2 qubit.	13
Hình 2.4	Mạch lượng tử trong thuật toán Shor.	14
Hình 2.5	Sơ đồ minh họa HPQC.	15
Hình 2.6	Sơ đồ minh họa Quantum Cloud.	17
Hình 2.7	Giao diện IBM Quantum Experience [23].	18
Hình 2.8	Minh họa quá trình cắt mạch lượng tử thành các mạch con nhỏ hơn [37].	23
Hình 2.9	Lựa chọn điểm cắt trong mạch lượng tử, thường tại các cổng điều khiển như CNOT [36].	24
Hình 2.10	Mạch chuẩn bị trạng thái GHZ 4 qubit.	25
Hình 2.11	Cắt mạch GHZ 4 qubit thành hai mạch con 2 qubit.	26
Hình 2.12	Hai mạch lượng tử riêng biệt.	28
Hình 2.13	Hai mạch lượng tử riêng biệt đã được ghép lại.	29
Hình 4.1	Kiến trúc tổng quan hệ thống.	48
Hình 4.2	Mô hình chi tiết của hệ thống.	50
Hình 4.3	Ví dụ minh họa về Gantt Chart.	61
Hình 4.4	Workflow của hệ thống.	62
Hình 4.5	Kiến trúc backend (manila).	63
Hình 4.6	Circuit ghz.	64
Hình 4.7	Mạch con 2 của ghz theo half cut.	64
Hình 4.8	Mạch con 2 của ghz theo half cut.	65
Hình 4.9	Transpile theo vị trí [0,1,2,3].	65

Hình 4.10	Transpile theo vị trí [0,1,2,4].	66
Hình 4.11	Map mạch lên máy theo vị trí [0,1,2,3].	66
Hình 4.12	Map mạch lên máy theo vị trí [0,1,2,4].	67
Hình 4.13	Thời gian chạy trên máy.	67
Hình 4.14	Kết quả chạy measurement không có nhiễu.	68
Hình 4.15	Kết quả chạy measurement có nhiễu.	68
Hình 4.16	Kết quả scheduler ban đầu.	69
Hình 4.17	Hiệu chỉnh thực tế của scheduler cho phép multithreading. . . .	69
Hình 4.18	Hiệu chỉnh thực tế của scheduler không cho phép multithreading.	70
Hình 7.1	Biểu đồ chuẩn hóa của thuật toán FFD.	103
Hình 7.2	Biểu đồ chuẩn hóa của thuật toán MILQ.	104
Hình 7.3	Biểu đồ chuẩn hóa của thuật toán MTMC.	104
Hình 7.4	Biểu đồ chuẩn hóa của thuật toán NoTADS.	105
Hình 7.5	Biểu đồ chuẩn hóa tổng hợp kết quả các giải thuật.	105

Danh sách bảng

Bảng 2.1	So sánh các dịch vụ lượng tử hiện tại	18
Bảng 2.2	So sánh các phương pháp định thời: ILP, Heuristic và AI	21
Bảng 2.3	So sánh Circuit Cutting và Circuit Knitting	30
Bảng 2.4	So sánh Simulator và Emulator trong tính toán lượng tử	31
Bảng 2.5	Các thư viện mô phỏng lượng tử	33
Bảng 3.1	Tổng hợp so sánh các công trình nghiên cứu liên quan	45
Bảng 4.1	Danh sách Quantum Backend theo số qubit	51
Bảng 4.2	Ánh xạ benchmark_name và tên đầy đủ các benchmark	52
Bảng 6.1	So sánh các giải thuật lập lịch lượng tử	100
Bảng 7.1	Mối liên hệ giữa các biến đầu vào và các chỉ số đánh giá	102
Bảng 7.2	Xếp hạng thuật toán theo các tiêu chí	106
Bảng 8.1	Các cổng lượng tử cơ bản và đặc điểm của chúng (Gates on One Target Qubit)	119
Bảng 8.2	Các cổng xoay và cổng pha trong tính toán lượng tử	120
Bảng 8.3	Các cổng điều khiển một qubit (có một qubit điều khiển và một qubit mục tiêu)	121
Bảng 8.4	Cổng hoán đổi và các cổng xoay điều khiển trong tính toán lượng tử	122

Chương 1

Giới thiệu

Trong chương 1, tổng quan, lý do và mục tiêu của đề tài sẽ được trình bày nhằm cung cấp cho người đọc cái nhìn toàn diện và động lực thực hiện đề tài. Các phần phạm vi và cấu trúc sẽ giúp định hướng nội dung nghiên cứu.

1.1 Tổng quan

Tính toán lượng tử là một trong những lĩnh vực công nghệ tiên phong, được kỳ vọng sẽ mang lại đột phá cho các bài toán tính toán phức tạp mà máy tính cổ điển không thể giải quyết hiệu quả. Những ứng dụng tiềm năng bao gồm tối ưu hóa tổ hợp, mô phỏng hệ lượng tử trong hóa học và vật lý, phát triển thuật toán trí tuệ nhân tạo, cũng như đảm bảo an toàn thông tin trong môi trường truyền thông hiện đại [1].

Mặc dù các nguyên lý lý thuyết về tính toán lượng tử đã được nghiên cứu sâu rộng, phần lớn thiết bị hiện tại vẫn thuộc giai đoạn *Noisy Intermediate-Scale Quantum (NISQ)*. Trong giai đoạn này, các hệ thống lượng tử bị giới hạn bởi số lượng qubit hạn chế, kết nối qubit không đầy đủ, tỷ lệ lỗi cao và tính ổn định kém. Những hạn chế này đặt ra thách thức đáng kể trong việc triển khai các thuật toán lượng tử quy mô lớn và đạt được ưu thế lượng tử trong thực tiễn [1]–[3].

Bên cạnh đó, với sự xuất hiện của các dịch vụ điện toán lượng tử qua nền tảng đám mây và việc các trung tâm siêu máy tính bắt đầu tích hợp bộ xử lý lượng tử như một gia tốc phần cứng, nhu cầu truy cập tài nguyên lượng tử đang ngày càng gia tăng. Khi nhiều người dùng cùng gửi một loạt tác vụ lên hệ thống, vấn đề phân bổ tài nguyên và xử lý đồng thời trở nên phức tạp, đòi hỏi các cơ chế lập lịch hiệu quả để đảm bảo tính

công bằng, tối ưu và tăng độ hiệu quả cho các công việc.

Do đó, việc quản lý tài nguyên lượng tử hiệu quả - bao gồm lập lịch các tác vụ, phân bổ mạch lượng tử lên thiết bị phù hợp và tối ưu hóa độ trung thực kết quả - trở thành một yêu cầu cấp thiết. Đây chính là động lực chính để nghiên cứu và phát triển các chiến lược lập lịch tài nguyên lượng tử thông minh, nhằm khai thác tối đa hiệu năng của phần cứng hiện có trong kỷ nguyên NISQ.

1.2 Lý do chọn đề tài

Trong bối cảnh tính toán lượng tử đang bước vào kỷ nguyên NISQ (*Noisy Intermediate-Scale Quantum*), vấn đề lập lịch và quản lý tài nguyên lượng tử trở thành một trong những thách thức quan trọng nhằm khai thác tối ưu các thiết bị lượng tử có số lượng qubit và khả năng kết nối còn hạn chế. Nhiều nghiên cứu đã đề xuất các giải thuật lập lịch và ánh xạ qubit nhằm tối thiểu thời gian thực thi, độ tin cậy của kết quả [4]–[6].

Tuy nhiên, phần lớn các công trình nghiên cứu hiện nay chỉ tập trung đề xuất giải thuật hoặc cải tiến một thành phần cụ thể mà thiếu một nền tảng chung để so sánh, đánh giá khách quan giữa các giải thuật khác nhau trong cùng một môi trường thực nghiệm tổng quát [7]–[9]. Các kết quả đánh giá thường chỉ giới hạn trong phạm vi nội bộ, với tập dữ liệu thử nghiệm riêng và điều kiện giả lập không đồng nhất, dẫn đến khó khăn trong việc so sánh chéo và tổng hợp kết quả giữa các nghiên cứu độc lập. Hơn nữa, sự đa dạng về kiến trúc phần cứng lượng tử, đặc biệt trong các hệ thống lượng tử không đồng nhất, đòi hỏi một công cụ có khả năng mô phỏng và đánh giá linh hoạt các chiến lược lập lịch trong nhiều cấu hình và kịch bản khác nhau.

Chính vì vậy, đề tài được lựa chọn với mục tiêu xây dựng một nền tảng mô phỏng hệ thống lập lịch tài nguyên lượng tử có khả năng triển khai, tích hợp và đánh giá nhiều giải thuật định thời khác nhau trong một môi trường nhất quán, linh hoạt và có khả năng mở rộng, góp phần cung cấp một công cụ tham chiếu cho cộng đồng nghiên cứu.

1.3 Mục tiêu đề tài

Luận văn này hướng tới hai mục tiêu chính:

- Xây dựng một nền tảng mô phỏng linh hoạt, có khả năng thực hiện và đo lường

các giải thuật lập lịch tài nguyên lượng tử.

- Thực hiện so sánh, đánh giá một số giải thuật định thời tài nguyên lượng tử dựa trên các tiêu chí định lượng được lựa chọn.

Nền tảng mô phỏng được thiết kế theo cấu trúc, cho phép dễ dàng thêm mới giải thuật, thay đổi tham số và phân tích kết quả theo nhiều tiêu chí. Thông qua hệ thống này, nhóm tác giả tiến hành thực nghiệm các giải thuật, thu thập dữ liệu và đưa ra đánh giá so sánh, góp phần cung cấp góc nhìn tổng quan về hiệu quả của các giải thuật lập lịch trong bối cảnh tài nguyên lượng tử hạn chế.

1.4 Phạm vi đề tài

Phần này trình bày phạm vi và giới hạn nghiên cứu của đề tài, bao gồm các câu hỏi nghiên cứu chính và những ràng buộc, giả định trong quá trình thực hiện. Việc xác định phạm vi rõ ràng giúp đảm bảo tính tập trung và khả thi, đồng thời định hướng cho phương pháp và kết quả nghiên cứu.

1.4.1 Câu hỏi nghiên cứu

Đề tài tập trung trả lời câu hỏi nghiên cứu chính sau:

"Những thuật toán định thời tài nguyên lượng tử nào được xem là phù hợp nhất trong môi trường đa máy lượng tử không đồng nhất, và các thuật toán này đã được đánh giá và so sánh như thế nào trong các nghiên cứu hiện tại?"

Câu hỏi này hướng tới việc xác định sự phù hợp và hiệu quả của các giải thuật lập lịch tài nguyên lượng tử khi áp dụng trong một hệ thống giả lập gồm nhiều máy tính lượng tử với cấu hình và khả năng không đồng nhất. Đồng thời, đề tài đặt mục tiêu xây dựng bộ tiêu chí đánh giá khách quan để so sánh các giải thuật trong một môi trường thực nghiệm thống nhất.

1.4.2 Giới hạn phạm vi

Đề tài giới hạn phạm vi nghiên cứu trong các khía cạnh sau:

- Để đảm bảo tính khả thi của đề tài, môi trường thực nghiệm được thực hiện hoàn toàn trên môi trường giả lập, sử dụng trình giả lập do Qiskit SDK [10] của IBM cung cấp giúp người dùng có thể linh động trong việc chạy mô phỏng và thư viện Qiskit có các thành phần hỗ trợ đầy đủ cho việc mô phỏng sát với thực tế.
- Nhằm không phân tán mục tiêu nghiên cứu, nhóm không đi sâu vào khía cạnh vật lý phần cứng lượng tử, không đề cập đến tối ưu thiết kế cổng lượng tử hay đặc tính vật lý qubit.
- Giả định môi trường hệ thống gồm nhiều máy tính lượng tử (multi-QPU) với kiến trúc không đồng nhất về số lượng qubit, topology kết nối và độ tin cậy. Điều này giúp cho người sử dụng có thể thay đổi linh hoạt phần cứng để có thể đa dạng quá trình mô phỏng.
- Ngoài ra, nhóm không đề cập tới các giải thuật định thời lai giữa lượng tử và cổ điển; tập trung hoàn toàn vào các giải thuật định thời lượng tử thuần túy trên multi-QPU. Vì nhóm đề tài muốn xử lý phần nhánh khi các công việc lượng tử được sắp xếp đưa vào máy.

1.5 Đóng góp đề tài

Đề tài của chúng tôi đóng góp một môi trường thực nghiệm linh hoạt, cho phép triển khai và đo lường hiệu quả các giải thuật lập lịch tài nguyên lượng tử.

Cụ thể, đề tài có ba đóng góp chính:

- Xây dựng môi trường thực nghiệm có khả năng mô phỏng và đánh giá giải thuật lập lịch lượng tử.
- Xác định và tổng hợp các tiêu chí đánh giá tổng quan cho giải thuật lập lịch trong môi trường lượng tử.
- Thực hiện so sánh tổng quan một số giải thuật lập lịch tài nguyên lượng tử, qua đó rút ra nhận xét về ưu nhược điểm và khả năng ứng dụng của từng giải thuật.

1.6 Cấu trúc đề tài

Để cung cấp một cái nhìn toàn diện và có hệ thống cho người đọc, luận văn được tổ chức thành 8 chương chính. Mỗi chương được xây dựng với một vai trò và mục tiêu riêng, nhưng có mối liên hệ chặt chẽ, đảm bảo tính liền mạch trong việc giải quyết vấn đề nghiên cứu. Phần còn lại của đề tài có cấu trúc chi tiết như sau:

- **Chương 2 – Cơ sở lý thuyết:** Trình bày các kiến thức nền tảng liên quan đến tính toán lượng tử, các tính chất cơ bản của hệ lượng tử, qubit, cổng lượng tử, mạch lượng tử, dịch vụ lượng tử, định thời tài nguyên (scheduling) và các mô phỏng (simulator) trong môi trường lượng tử. Đây là phần đặt nền móng lý thuyết cho các chương tiếp theo.
- **Chương 3 – Nghiên cứu liên quan:** Tổng hợp, phân tích và đánh giá các công trình nghiên cứu trong lĩnh vực lập lịch và quản lý tài nguyên lượng tử. Chương này giúp làm rõ những hướng đi đã có, những hạn chế còn tồn tại, và khoảng trống nghiên cứu mà luận văn hướng đến.
- **Chương 4 – Kiến trúc hệ thống:** Trình bày kiến trúc tổng quan và chi tiết của hệ thống mô phỏng được xây dựng, bao gồm các bước xử lý chính: thiết lập dữ liệu đầu vào, tạo benchmark, cắt mạch lượng tử, thực thi giải thuật lập lịch, biên dịch mạch lượng tử (transpile), mô phỏng thiết bị giả lập, tính toán chỉ số hiệu suất, và hiển thị kết quả.
- **Chương 5 – Tiêu chí đánh giá:** Xây dựng và giải thích hệ thống các tiêu chí đánh giá hiệu quả của các giải thuật lập lịch trong môi trường lượng tử. Các chỉ số bao gồm: độ trung thực (fidelity), thông lượng (throughput), thời gian phản hồi (response time), thời gian hoàn thành (turnaround time), tổng thời gian thực hiện (makespan), độ trễ lập lịch (scheduling latency), hiệu suất sử dụng (utilization), và chi phí phát sinh do cắt mạch (cut sampling overhead).
- **Chương 6 – Giải thuật định thời lượng tử:** Trình bày chi tiết các giải thuật lập lịch được lựa chọn để thử nghiệm và đánh giá, bao gồm: First-Fit Decreasing Bin Packing [11], MILP (Mixed Integer Linear Programming) [9], Noise and Time

Aware Distributed Scheduler (NoTaDS) [7], Multi-Task Multi-Chip Scheduling (MTMC) [12].

- **Chương 7 – Đánh giá giải thuật:** Trình bày kết quả thực nghiệm từ việc triển khai các giải thuật trên hệ mô phỏng. Bao gồm mô tả hệ thống chạy thực nghiệm, benchmark thử nghiệm, đánh giá vai trò các đầu vào, kịch bản đánh giá, kết quả đo lường, phân tích so sánh, nhận xét tổng quan và đưa ra đánh giá chung.
- **Chương 8 – Tổng kết:** Tổng hợp những kết quả chính của đề tài, đưa ra kết luận, thảo luận các vấn đề mở, đề xuất hướng nghiên cứu và phát triển trong tương lai.

Chương 2

Cơ sở lý thuyết

Trong chương 2, các kiến thức nền tảng liên quan đến đề tài sẽ được trình bày nhằm cung cấp cơ sở lý thuyết và các công cụ cần thiết phục vụ cho quá trình nghiên cứu và thực hiện đề tài. Những kiến thức này nhằm giúp người đọc hiểu rõ hơn về bối cảnh và cơ sở khoa học của đề tài giúp cho việc hiểu đề tài một cách dễ dàng hơn.

2.1 Tính toán lượng tử

Tính toán lượng tử (*quantum computing*) là một hướng đi đột phá trong khoa học máy tính và công nghệ thông tin, mở ra tiềm năng giải quyết những bài toán phức tạp mà máy tính cổ điển gặp nhiều hạn chế [3]. Khác với hệ thống cổ điển sử dụng bit nhị phân (0 hoặc 1), tính toán lượng tử vận hành trên các đơn vị thông tin lượng tử gọi là *qubit*, tuân theo các nguyên lý cơ học lượng tử như **chồng chập (superposition)**, **rối lượng tử (entanglement)** và **khử kết hợp (decoherence)**. Những đặc tính này mang lại khả năng xử lý song song vượt trội và mở ra nhiều ứng dụng chưa từng có tiền lệ.

Trong phần này, chúng tôi sẽ lần lượt trình bày các đặc tính cơ bản của hệ lượng tử, khái niệm qubit, các cổng lượng tử, và cấu trúc mạch lượng tử. Việc hiểu rõ những thành phần này là nền tảng để xây dựng, vận hành các thuật toán lượng tử.

2.1.1 Các tính chất cơ bản của hệ lượng tử

Cơ học lượng tử cung cấp nền tảng lý thuyết cho tính toán lượng tử, nổi bật với ba đặc tính then chốt: **chồng chập (superposition)**, **rối lượng tử (entanglement)**, và **khử kết**

hợp (decoherence). Các tính chất này không chỉ tạo nên sự khác biệt về lý thuyết mà còn quyết định khả năng và giới hạn của hệ lượng tử.

Chồng chập (Superposition)

Chồng chập cho phép một qubit tồn tại đồng thời ở nhiều trạng thái. Nếu một bit cổ điển chỉ mang giá trị 0 hoặc 1, thì qubit có thể biểu diễn dưới dạng tổ hợp tuyến tính:

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle \quad (2.1)$$

với $\alpha, \beta \in \mathbb{C}$ là các số phức thỏa mãn điều kiện chuẩn hóa:

$$|\alpha|^2 + |\beta|^2 = 1 \quad (2.2)$$

Nhờ chồng chập, một hệ n qubit có thể biểu diễn 2^n trạng thái cùng lúc:

$$|\psi_n\rangle = \sum_{i=0}^{2^n-1} c_i|i\rangle, \quad \sum_{i=0}^{2^n-1} |c_i|^2 = 1 \quad (2.3)$$

Tính chất này mang lại khả năng xử lý song song, là cơ sở sức mạnh của tính toán lượng tử.

Rối lượng tử (Entanglement)

Rối lượng tử là hiện tượng hai hoặc nhiều qubit liên kết chặt chẽ đến mức trạng thái của chúng không thể mô tả riêng lẻ, mà chỉ có thể mô tả dưới dạng tổng thể.

Ví dụ, một trạng thái Bell:

$$|\Phi^+\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle) \quad (2.4)$$

Ngoài ra còn có các trạng thái Bell khác:

$$|\Phi^-\rangle = \frac{1}{\sqrt{2}}(|00\rangle - |11\rangle) \quad (2.5)$$

$$|\Psi^+\rangle = \frac{1}{\sqrt{2}}(|01\rangle + |10\rangle) \quad (2.6)$$

$$|\Psi^-\rangle = \frac{1}{\sqrt{2}}(|01\rangle - |10\rangle) \quad (2.7)$$

Khi đo một qubit trong cặp rồi, kết quả của qubit còn lại sẽ được xác định ngay lập tức, bất kể khoảng cách giữa chúng. Tính chất này đóng vai trò thiết yếu trong các thuật toán lượng tử, truyền thông lượng tử và bảo mật lượng tử.

Khử kết hợp (Decoherence)

Khử kết hợp là hiện tượng qubit mất trạng thái lượng tử do tương tác với môi trường, làm cho hệ lượng tử dần trở nên cổ điển hóa. Trạng thái thuần (pure state) ban đầu sẽ trở thành trạng thái hỗn hợp (mixed state), được mô tả bằng ma trận mật độ ρ :

$$\rho = \sum_i p_i |\psi_i\rangle \langle \psi_i| \quad (2.8)$$

Với p_i là xác suất của trạng thái $|\psi_i\rangle$. Nếu không có khử kết hợp, hệ sẽ duy trì trạng thái thuần:

$$\text{Tr}(\rho^2) = 1 \quad (2.9)$$

Nhưng khi chịu tác động của môi trường:

$$\text{Tr}(\rho^2) < 1 \quad (2.10)$$

Khử kết hợp là thách thức lớn trong việc duy trì trạng thái lượng tử đủ lâu để thực hiện các phép tính phức tạp. Các giải pháp như sửa lỗi lượng tử, che chắn nhiễu, và tối ưu thiết kế mạch đang là trọng tâm trong nghiên cứu phần cứng lượng tử.

Ba đặc tính này không chỉ là lý thuyết nền tảng mà còn trực tiếp ảnh hưởng đến hiệu năng, độ tin cậy và khả năng hiện thực của các mạch lượng tử, từ đó tác động đến các giải thuật định thời và lập lịch trong môi trường lượng tử.

2.1.2 Qubit và các phép toán cơ bản

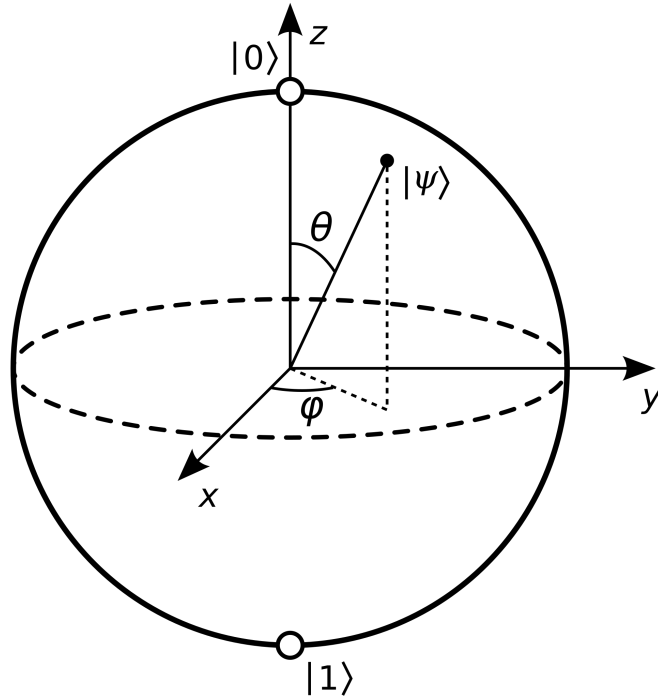
Một qubit là đơn vị thông tin cơ bản trong tính toán lượng tử, được mô tả bởi một vectơ trong không gian Hilbert hai chiều:

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle \quad (2.11)$$

Với điều kiện chuẩn hóa:

$$|\alpha|^2 + |\beta|^2 = 1 \quad (2.12)$$

Trạng thái này được minh hoạ trực quan trong *không gian Bloch* (Hình 2.1), nơi góc quay thể hiện pha và biên độ.



Hình 2.1: Trực quan hoá trạng thái qubit trên hình cầu Bloch. Nguồn: [Smite-Meister](#), CC BY-SA 3.0, via Wikimedia Commons.

Để thao tác trên qubit, các **cổng lượng tử (quantum gate)** được sử dụng. Mỗi cổng là một phép biến đổi đơn vị tác động lên không gian trạng thái, tương tự như cổng logic trong máy tính cổ điển. Các cổng này được phân loại theo số qubit tác động và đặc tính toán học.

Ví dụ, một số cổng cơ bản:

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \quad Y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}, \quad Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}, \quad H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}, \quad (2.13)$$

Trong đó:

- X : cổng NOT lượng tử (Pauli- X), đảo trạng thái $|0\rangle \leftrightarrow |1\rangle$.
- Y : cổng Pauli- Y , thay đổi pha trạng thái $|0\rangle$ thành $i|1\rangle$ và $|1\rangle$ thành $-i|0\rangle$.
- Z : cổng Pauli- Z , thay đổi pha trạng thái $|1\rangle$.
- H : cổng Hadamard, tạo trạng thái chồng chập từ $|0\rangle$ hoặc $|1\rangle$.

Các cổng này là thành phần cơ bản trong việc xây dựng mạch lượng tử (quantum circuit) và triển khai các thuật toán lượng tử.

2.1.3 Cổng lượng tử

Các cổng lượng tử (quantum gates) là các phép toán tuyến tính, đơn vị (unitary operations) tác động lên không gian trạng thái của qubits, đóng vai trò là các khối xây dựng cơ bản trong việc thao tác và xử lý thông tin lượng tử. Mỗi cổng lượng tử tương ứng với một ma trận đơn vị, đảm bảo bảo toàn chuẩn hoá của trạng thái lượng tử sau phép biến đổi [13].

Tương tự như các cổng logic trong tính toán cổ điển, cổng lượng tử cho phép thao tác trên một hoặc nhiều qubit, nhưng với khả năng xử lý các trạng thái chồng chập và rối lượng tử. Một mạch lượng tử phức tạp có thể được biểu diễn bằng sự kết hợp tuần tự và song song của các cổng lượng tử cơ bản.

Trong phần này, nhóm trình bày các loại cổng lượng tử thường gặp, phân loại theo số lượng qubit tác động và đặc điểm toán học của chúng:

- **Cổng tác động trên một qubit:** bao gồm các cổng Hadamard, Pauli (X , Y , Z), Identity, và các cổng xoay quanh các trục Bloch (RX , RY , RZ). Các cổng này được liệt kê trong Bảng 8.1 và Bảng 8.2, với biểu diễn ma trận và thuộc tính đặc trưng của từng cổng.
- **Cổng điều khiển (controlled gates):** thao tác trên hai qubit, trong đó một qubit đóng vai trò điều khiển, quyết định tác động lên qubit mục tiêu. Ví dụ, các cổng CNOT (CX), CY , CZ , CH được trình bày trong Bảng 8.3.

- **Cổng swap và xoay điều khiển:** bao gồm cổng SWAP hoán đổi hai qubit, và các cổng xoay điều khiển CRX, CRY, CRZ, được liệt kê trong Bảng 8.4.

Mỗi cổng lượng tử mang đặc tính toán học riêng, chẳng hạn tự nghịch đảo (involutory), thuộc nhóm Pauli (Pauli group), hoặc đơn vị đặc biệt (special unitary, định thức bằng 1). Việc lựa chọn và kết hợp các cổng này là trọng tâm trong thiết kế thuật toán lượng tử, đảm bảo thực hiện các phép biến đổi trạng thái mong muốn trên hệ lượng tử.

Các bảng từ Bảng 8.1 đến Bảng 8.4 tổng hợp chi tiết biểu diễn ma trận, số qubit tác động, và các thuộc tính nổi bật của từng loại cổng lượng tử, cung cấp cơ sở tham khảo cho việc xây dựng và phân tích mạch lượng tử.

2.1.4 Mạch lượng tử (Quantum Circuit)

Mạch lượng tử là một mô hình toán học nền tảng trong tính toán lượng tử, mô tả quá trình biến đổi trạng thái của hệ qubit thông qua chuỗi các phép toán đơn vị (unitary operations) và các phép đo (measurements). Tương tự chương trình trong máy tính cổ điển, một mạch lượng tử là sự kết hợp của các cổng lượng tử tác động lên một hoặc nhiều qubit theo thứ tự thời gian.

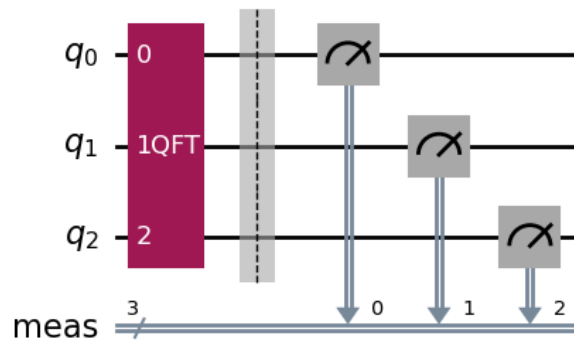
Cấu trúc cơ bản của một mạch lượng tử bao gồm ba giai đoạn:

1. **Chuẩn bị trạng thái đầu vào (State Preparation):** Các qubit được khởi tạo ở trạng thái $|0\rangle$ hoặc trạng thái chồng chập tùy theo yêu cầu bài toán.
2. **Thao tác với các cổng lượng tử (Quantum Gate Operations):** Áp dụng tuần tự các phép biến đổi đơn vị để thao tác lên các qubit, tạo ra sự tiến hóa trạng thái của hệ.
3. **Đo lường (Measurement):** Thực hiện phép đo trên một số hoặc toàn bộ qubit để rút ra kết quả cổ điển.

Mạch lượng tử thường được biểu diễn dưới dạng sơ đồ tuyến tính (quantum circuit diagram), trong đó mỗi đường ngang biểu diễn một qubit, còn các cổng lượng tử là các ký hiệu hình học tác động lên một hoặc nhiều đường qubit.

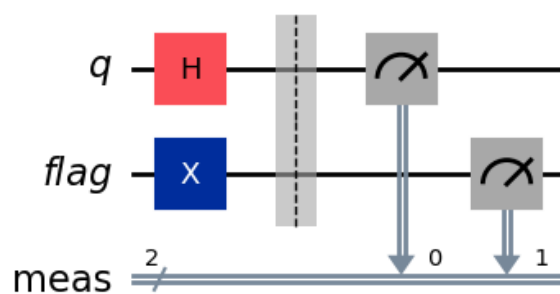
Trong lịch sử phát triển tính toán lượng tử, nhiều mạch lượng tử nổi tiếng đã được đề xuất, ứng dụng vào các bài toán mà máy tính cổ điển khó giải quyết hiệu quả. Dưới đây là tổng hợp một số mạch nổi bật:

- **Quantum Fourier Transform (QFT) [14]:** là một phiên bản lượng tử của biến đổi Fourier rời rạc (Discrete Fourier Transform - DFT), được sử dụng trong tính toán lượng tử. Phép biến đổi này sẽ ánh xạ một trạng thái cơ sở $|x\rangle$ sang một tổ hợp tuyến tính của các trạng thái cơ sở khác. QFT được sử dụng trong một số thuật toán lượng tử và phép biến đổi này có độ phức tạp $O(n^2)$ [15].



Hình 2.2: Mạch lượng tử Quantum Fourier Transform (QFT) với 3 qubit.

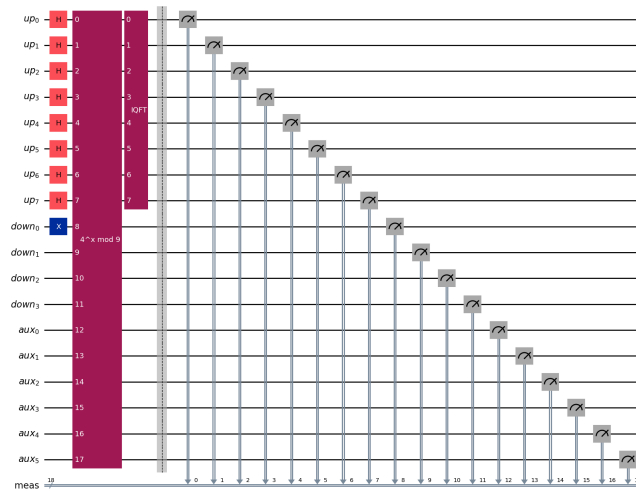
- **Grover's Circuit:** mạch lượng tử trong thuật toán Grover, dùng để tìm kiếm phần tử trong không gian không có cấu trúc với độ phức tạp $O(\sqrt{N})$ so với $O(N)$ của thuật toán tìm kiếm tuyến tính cổ điển. Thuật toán này mang lại tốc độ nhanh hơn khoảng $\sim 100\%$ trên không gian lớn[16].



Hình 2.3: Mạch lượng tử trong thuật toán Grover với 2 qubit.

- **Shor's Circuit:** mạch lượng tử được thiết kế cho thuật toán Shor – thuật toán phân tích thừa số nguyên. Shor's algorithm có khả năng phân tích một số nguyên N với độ phức tạp $O((\log N)^3)$, vượt trội so với thuật toán cổ điển tốt nhất có độ phức

tập $O(e^{(1.9(\log N)^{1/3}(\log \log N)^{2/3})})$. Ứng dụng của Shor có tiềm năng phá vỡ nhiều hệ mật mã dựa trên tính khó phân tích thừa số, như RSA [17], [18].



Hình 2.4: Mạch lượng tử trong thuật toán Shor.

Các mạch lượng tử trên minh chứng tiềm năng vượt trội của tính toán lượng tử trong việc giải quyết các bài toán phức tạp. Tuy nhiên, thách thức hiện tại vẫn nằm ở việc hiện thực hoá các mạch này trên phần cứng với số lượng qubit đủ lớn và duy trì độ trung thực của kết quả.

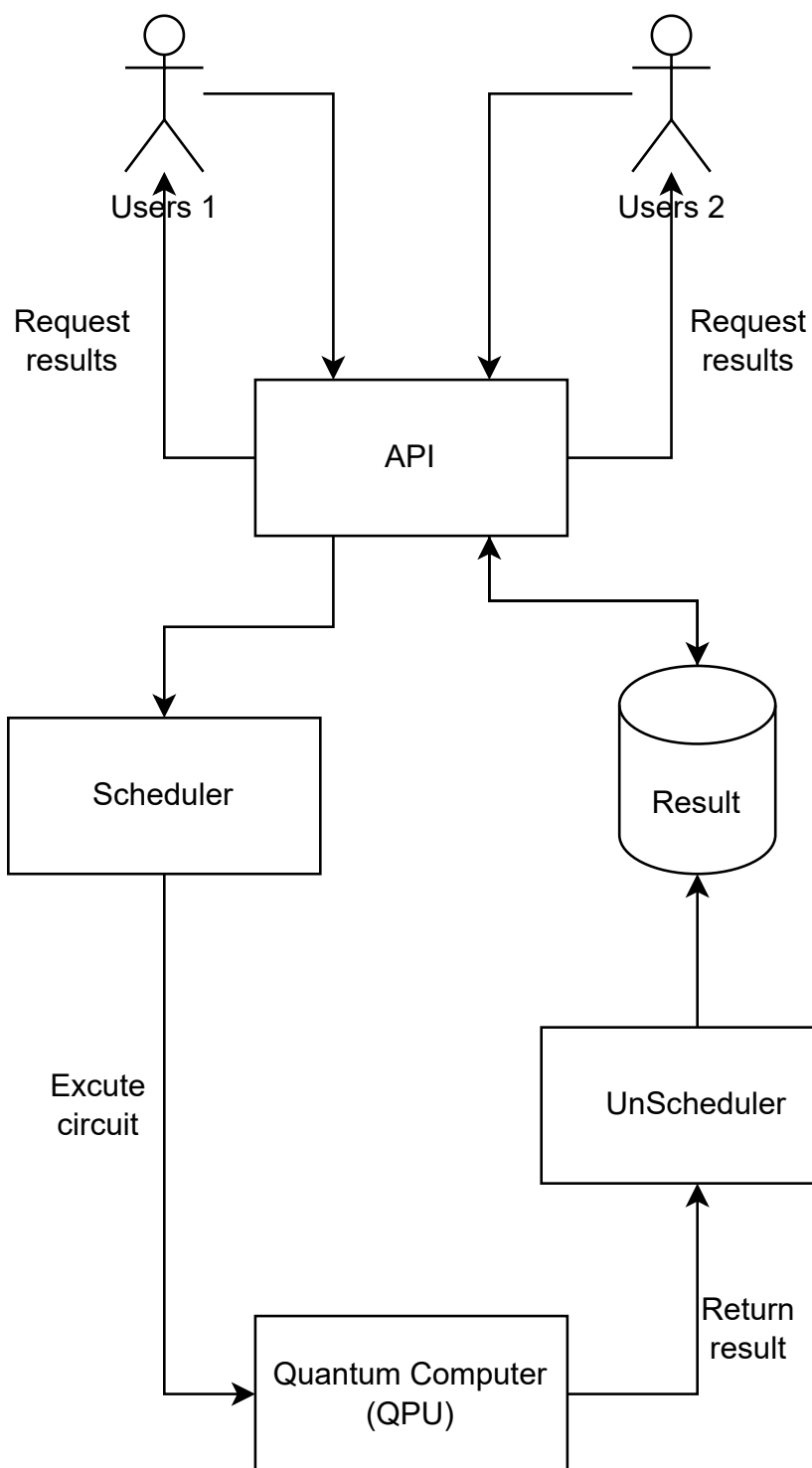
2.2 Dịch vụ lượng tử

Sau khi thiết kế mạch lượng tử, việc thực thi đòi hỏi phần cứng chuyên biệt. Tuy nhiên, bộ xử lý lượng tử (Quantum Processing Unit – QPU) hiện còn đắt đỏ, khó tiếp cận, và yêu cầu điều kiện vận hành phức tạp [19], [20]. Để khắc phục, hai mô hình triển khai chính đã phát triển: **hệ thống QPU tích hợp dạng tăng tốc phần cứng (accelerator)** và **dịch vụ điện toán đám mây lượng tử (quantum cloud service)**.

Phần này tổng quan hai mô hình và các nền tảng dịch vụ lượng tử hiện tại, cung cấp cơ sở lựa chọn công cụ triển khai phù hợp cho đề tài.

2.2.1 Mô hình triển khai

Hệ thống High-Performance Quantum Computing (HPQC) Hệ thống *HPQC 2.5* kết hợp QPU và CPU/GPU trong một nền tảng siêu máy tính, với QPU đóng vai trò bộ



Hình 2.5: Sơ đồ minh họa HPQC.

gia tốc – tương tự GPU trong High-Performance Computing (HPC). Người dùng lập trình trên CPU, “offload” phép toán lượng tử sang QPU xử lý, giúp giảm độ trễ truyền thông và tối ưu hóa mô hình lai (hybrid quantum-classical).

Một số dự án tiêu biểu:

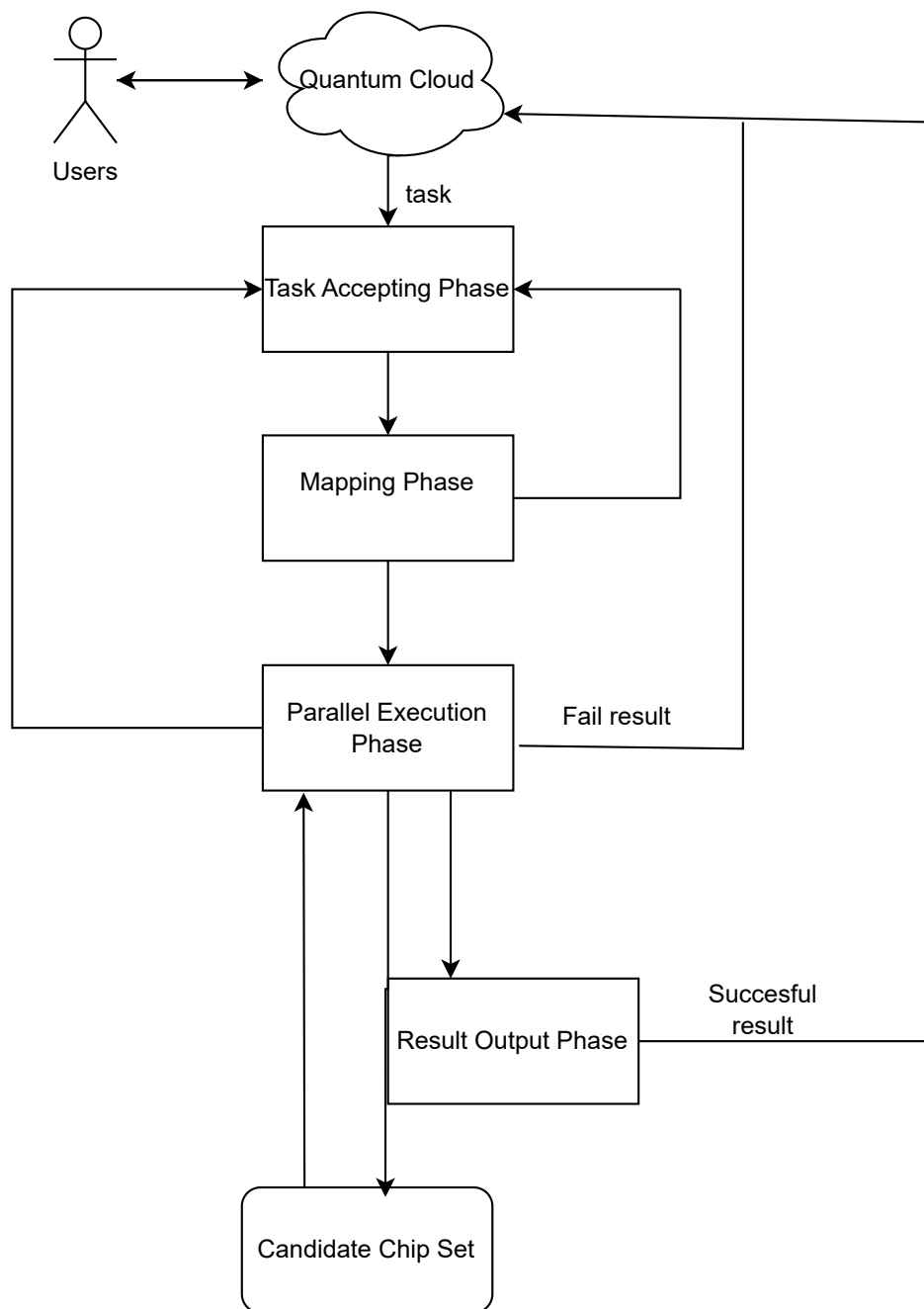
- **Fujitsu Digital Annealer:** giải bài toán tối ưu tổ hợp theo mô hình annealing lượng tử. Dự án này hứa hẹn có thể tính toán song song, xử lý các bài toán tối ưu mà máy tính cổ điển không thể làm được [21].
- **D-Wave Advantage:** sử dụng mô hình adiabatic, tích hợp accelerator cho tối ưu hóa. Có khả năng xử lý bài toán 120 qubit mà phiên bản tiền nhiệm không xử lý được [22].

Do chi phí và hạ tầng đặc thù, HPQC chủ yếu dùng trong nghiên cứu và doanh nghiệp lớn.

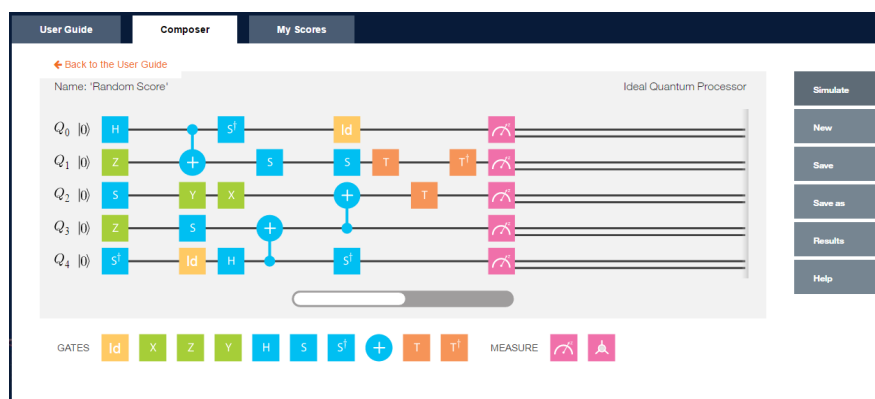
Dịch vụ điện toán lượng tử đám mây (quantum cloud service)

Dịch vụ điện toán lượng tử đám mây Dịch vụ đám mây lượng tử 2.6 cho phép truy cập phần cứng thực hoặc giả lập từ xa mà không cần sở hữu QPU. Các tập đoàn công nghệ lớn đã phát triển nền tảng kèm SDK hỗ trợ lập trình. Một số nền tảng tiêu biểu:

- **IBM Quantum Experience:** truy cập miễn phí phần cứng IBM Falcon (27 qubit), IBM Eagle (127 qubit); lập trình qua **Qiskit**; hỗ trợ tài liệu và cộng đồng.
- **Microsoft Azure Quantum:** hỗ trợ QPU từ IonQ, Quantinuum; lập trình bằng **Q#** [24] trong Visual Studio; tích hợp workflow hybrid.
- **Rigetti Computing:** dịch vụ **QCS** kết nối trực tiếp QPU; SDK **Forest** hỗ trợ Python, Quil [25]; phần cứng dựa trên qubit siêu dẫn.
- **Amazon Braket** [26]: hỗ trợ QPU đa dạng: IonQ (bẫy ion), Rigetti (siêu dẫn), D-Wave (adiabatic); môi trường mô phỏng hàng nghìn qubit; tích hợp AWS.
- **Intel Quantum:** SDK **Intel Quantum SDK** (C++) [27]; tập trung nghiên cứu qubit silicon spin; chưa cung cấp đám mây công khai.



Hình 2.6: Sơ đồ minh họa Quantum Cloud.



Hình 2.7: Giao diện IBM Quantum Experience [23].

- **Google Quantum AI:** phần cứng **Sycamore** đạt “quantum supremacy” (2019); thư viện **Cirq** [28] (Python); ứng dụng: mô phỏng vật liệu, tối ưu hóa, học máy.

2.2.2 So sánh tổng quan

Bảng 2.1 tóm tắt các nền tảng lượng tử chính về ngôn ngữ, SDK và khả năng cung cấp đám mây.

Bảng 2.1: So sánh các dịch vụ lượng tử hiện tại

Nhà cung cấp	Ngôn ngữ	SDK	Cung cấp đám mây
IBM Quantum Platform	Python	Qiskit [10]	Miễn phí (giới hạn qubit)
Microsoft Azure Quantum	Q#	QDK [24]	Trả phí
Rigetti Computing	Python	Forest SDK [25]	Đăng ký truy cập
Amazon Braket	Python	Braket SDK [26]	Trả phí
Intel Quantum	C++	Intel Quantum SDK [27]	Chưa cung cấp
Google Quantum AI	Python	Cirq [28]	Đăng ký truy cập

Việc lựa chọn nền tảng phụ thuộc mục tiêu, ngân sách, loại phần cứng. Trong đề tài này, nhóm sử dụng **IBM Quantum Experience** và **Qiskit** nhờ truy cập miễn phí, tài liệu phong phú, cộng đồng lập trình hỗ trợ rộng rãi.

2.3 Định thời

Trong các hệ thống tính toán lượng tử hiện đại, đặc biệt là mô hình High-Performance Quantum Computing (HPQC) và dịch vụ đám mây lượng tử (quantum cloud services), quá trình thực thi các mạch lượng tử thường được tổ chức theo dạng **danh sách các job** (job queue) [29], [30]. Mỗi job đại diện cho một mạch lượng tử được người dùng gửi đến hệ thống. Với hạn chế về số lượng qubit, độ sâu mạch (circuit depth) và thời gian decoherence, việc quản lý thứ tự và tài nguyên xử lý các job này trở thành một bài toán quan trọng.

Một vấn đề cốt lõi là **việc định thời (scheduling)** – quá trình gán tài nguyên (QPU, qubit, thời gian đo, kênh readout) cho các job sao cho đạt được các mục tiêu tối ưu hóa nhất định. Theo một định nghĩa tổng quát, định thời là bài toán gán tài nguyên cần thiết cho các tác vụ để thực thi trong một không gian tài nguyên hữu hạn [31]. Trong bối cảnh lượng tử, bài toán trở nên phức tạp hơn bởi các ràng buộc vật lý: qubit không đồng nhất, lỗi rẽ nhánh (crosstalk), giới hạn connectivity trong kiến trúc phần cứng, và thời gian decoherence ngắn. Hơn nữa, việc lập lịch còn phải tính đến các giới hạn thực tế của hệ thống cloud, như chia sẻ hạ tầng giữa nhiều người dùng.

Từ góc độ lý thuyết, định thời là một bài toán NP-hard, đặc biệt trong môi trường lượng tử nơi cần đồng thời tối ưu đa mục tiêu (multi-objective optimization).

Các dạng giải thuật định thời

Giải thuật định thời chính là trái tim của bài toán định thời tài nguyên. Tùy theo mục tiêu và môi trường triển khai, các giải thuật được thiết kế để hướng đến các tiêu chí tối ưu khác nhau: tiết kiệm năng lượng, giảm thiểu lỗi, tối đa throughput, hoặc giảm chi phí dịch vụ đám mây [32].

Trong bối cảnh định thời tài nguyên lượng tử, các giải thuật có thể phân thành ba nhóm chính:

- **ILP (Integer Linear Programming):** phương pháp tối ưu hóa toán học, mô hình bài toán bằng hệ phương trình tuyến tính nguyên. ILP cho phép tìm lời giải tối ưu toàn cục, thích hợp cho các hệ thống nhỏ hoặc nghiên cứu học thuật. Tuy nhiên,

độ phức tạp tăng nhanh theo kích thước bài toán khiến ILP ít phù hợp với các hệ thống lớn hoặc yêu cầu thời gian thực [33], [34].

- **Heuristic:** phương pháp gần đúng dựa trên quy tắc thiết kế sẵn hoặc kinh nghiệm. Heuristic phù hợp trong môi trường yêu cầu tính toán nhanh, ít tài nguyên, chẳng hạn hệ thống IoT hoặc các dịch vụ edge computing. Ví dụ: giải thuật FIFO (First In First Out), Round Robin, hoặc Greedy-based scheduling [35].
- **AI (Trí tuệ nhân tạo):** nhóm phương pháp dựa trên khả năng học từ dữ liệu, bao gồm học máy (Machine Learning), học sâu (Deep Learning), hoặc học tăng cường (Reinforcement Learning). Các giải thuật AI có khả năng thích nghi với thay đổi của môi trường, học từ lịch sử vận hành, và tối ưu hoá theo thời gian. Tuy nhiên, chi phí huấn luyện cao, yêu cầu dữ liệu lớn, và độ phức tạp triển khai cũng là thách thức.

Mỗi phương pháp có ưu điểm, nhược điểm và phạm vi ứng dụng riêng, được tóm tắt trong Bảng 2.2.

Bảng 2.2: So sánh các phương pháp định thời: ILP, Heuristic và AI

Tiêu chí	ILP (Integer Linear Programming)	Heuristic	AI (Trí tuệ nhân tạo)
Độ chính xác	Cao, thường đạt tối ưu toàn cục	Trung bình, giải gần đúng	Trung bình đến cao (tùy dữ liệu huấn luyện)
Tốc độ thực thi	Chậm, đặc biệt với bài toán lớn	Nhanh, phù hợp thời gian thực	Trung bình, phụ thuộc vào mô hình
Khả năng mở rộng	Kém, khó mở rộng với số biến lớn	Tốt, mở rộng dễ dàng	Tốt, có thể học được từ dữ liệu lớn
Khả năng thích nghi	Kém, phải xây dựng lại mô hình nếu thay đổi hệ thống	Thấp, không phản ứng với thay đổi môi trường	Cao, có thể học và thích nghi theo thời gian
Độ phức tạp triển khai	Cao, yêu cầu kiến thức toán học	Thấp đến trung bình, dễ triển khai	Cao, cần dữ liệu và kỹ thuật huấn luyện
Ứng dụng điển hình	Hệ thống tối ưu hóa toán học, nghiên cứu lý thuyết	IoT, thiết bị nhúng, hệ thống thời gian thực	Cloud, học máy lượng tử, hệ thống thích nghi

Hiện nay, trong bối cảnh dịch vụ lượng tử trên cloud, việc định thời không chỉ giới hạn ở việc gán qubit vật lý mà còn bao gồm việc quản lý thời gian xếp hàng, ưu tiên người dùng, và tối ưu chi phí truy cập dịch vụ. Điều này đặt ra yêu cầu phát triển các giải thuật định thời kết hợp đa mục tiêu, có khả năng mở rộng, thích nghi và đáp ứng yêu cầu chất lượng dịch vụ (QoS).

Nhóm xây dựng một hệ mô phỏng quy trình chạy job lượng tử, với khả năng thay đổi linh hoạt giữa các giải thuật định thời (ILP, heuristic, AI). Mục tiêu của hệ thống là đánh giá hiệu suất của các phương pháp định thời dưới các điều kiện khác nhau, làm cơ sở cho việc lựa chọn thuật toán phù hợp nhất theo đặc thù ứng dụng.

2.4 Kỹ thuật xử lý mạch lượng tử với tài nguyên hạn chế

Trong kỷ nguyên NISQ (*Noisy Intermediate-Scale Quantum*), các thiết bị lượng tử hiện nay vẫn còn hạn chế: số lượng qubit hữu dụng thấp, độ sâu mạch bị giới hạn, và độ tin cậy phép đo chưa cao. Để có thể thực thi các mạch lượng tử quy mô lớn trên phần cứng giới hạn, nhiều kỹ thuật đã ra đời nhằm thao tác ở mức cấu trúc mạch, cho phép chia nhỏ các mạch để có thể chạy trên phần cứng hạn chế. Ngược lại, nhiều công trình nghiên cứu nỗ lực dẫn tới việc nhiều mạch lượng tử nhỏ được thử nghiệm, dẫn tới việc sử dụng tài nguyên không hiệu quả. Việc kết nối các mạch lượng tử nhỏ lại giúp chạy cùng lúc tăng độ hiệu quả sử dụng tài nguyên. Điều đó dẫn tới hai kỹ thuật tương tác với mạch là:

Hai kỹ thuật tiêu biểu trong nhóm này là:

1. **Cắt mạch (Circuit Cutting)** – chia mạch lượng tử thành các mạch con độc lập, thực thi riêng lẻ rồi ghép kết quả hậu xử lý.
2. **Ghép mạch (Circuit Knitting)** – ghép nối mạch lượng tử từ các phần nhỏ hơn thông qua kỹ thuật tổng hợp hoặc giả lập.

Phần này sẽ trình bày chi tiết hai kỹ thuật, bao gồm định nghĩa, cơ chế, ưu nhược điểm, và ví dụ minh họa.

2.4.1 Cắt mạch

Trong tính toán lượng tử, cắt mạch [36] là một kỹ thuật giúp các phép toán có số lượng qubit lớn thực thi được ở các máy tính lượng tử có số qubit nhỏ hơn thông qua việc chia nhỏ số qubit của phép toán thành số qubit mà máy tính có thể xử lý được. Phần dưới đây sẽ trình bày về các kiến thức của kỹ thuật này.

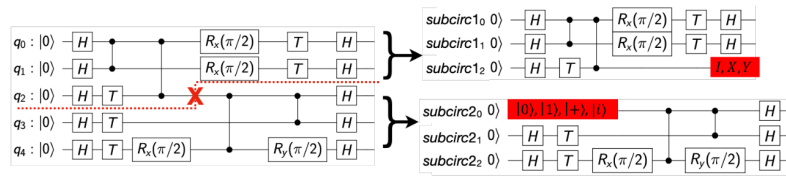
Định nghĩa và bối cảnh

Trong bối cảnh các thiết bị lượng tử hiện nay còn hạn chế về số lượng qubit vật lý, độ sâu mạch tối đa, và chịu ảnh hưởng mạnh của nhiễu (noise), việc thực thi trực tiếp các

mạch lượng tử lớn trên phần cứng thực tế là không khả thi. Đặc biệt trong kỷ nguyên NISQ (*Noisy Intermediate-Scale Quantum*) [3], số lượng qubit khả dụng thường chỉ dao động từ vài chục đến vài trăm qubit, trong khi nhiều thuật toán lượng tử yêu cầu số lượng lớn hơn đáng kể.

Để giải quyết hạn chế này, một kỹ thuật được phát triển nhằm cho phép thực thi các mạch lớn trên phần cứng nhỏ hơn, gọi là cắt mạch lượng tử (*Quantum Circuit Cutting*). Ý tưởng cốt lõi là phân tách một mạch lượng tử phức tạp thành nhiều mạch con nhỏ hơn, mỗi mạch con có số qubit vừa đủ để chạy trên phần cứng hiện tại. Sau đó, kết quả từ các mạch con này được kết hợp lại để tái tạo kết quả tương đương với mạch ban đầu [37], [38].

Kỹ thuật này đóng vai trò quan trọng trong việc mở rộng khả năng tính toán của các thiết bị lượng tử hiện tại, đồng thời cũng tạo ra hướng đi mới cho các mô hình lai (hybrid quantum-classical). Minh họa khái niệm được trình bày trong Hình 2.8.



Hình 2.8: Minh họa quá trình cắt mạch lượng tử thành các mạch con nhỏ hơn [37].

Cơ chế hoạt động

Kỹ thuật cắt mạch lượng tử bao gồm ba bước chính: **phân tách (decomposition)**, **thực thi (execution)** và **tái tạo (reconstruction)**.

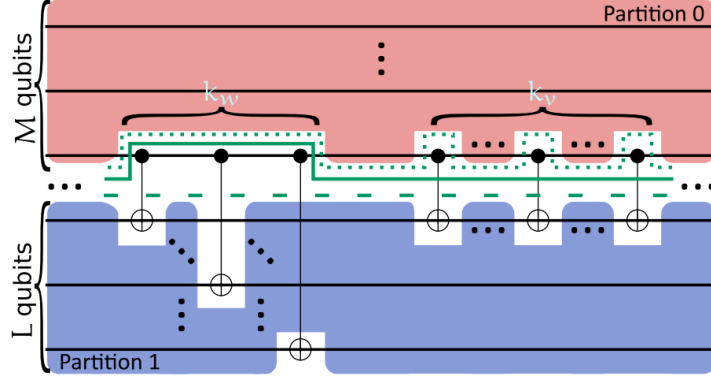
Phân tách (Decomposition)

Bước đầu tiên là xác định các *điểm cắt* (cut points) trong mạch lượng tử. Tại các điểm này, trạng thái lượng tử sẽ được thay thế bằng một biểu diễn trung gian dưới dạng đo (measurement) và tái chuẩn bị (preparation), tách mạch thành hai phần độc lập. Thông thường, các điểm cắt được chọn tại các liên kết quan trọng như cổng hai qubit (ví dụ CNOT) hoặc đường qubit giao nhau.

Về mặt toán học, việc cắt một qubit tương đương với việc chèn một phép chiếu (projection) và phép tái chuẩn bị trạng thái (state preparation):

$$I = \sum_m M_m \otimes P_m$$

Trong đó M_m là phép đo (measurement operator) và P_m là phép tái chuẩn bị trạng thái dựa trên kết quả đo m . Hình 2.9 minh họa cách chọn các điểm cắt trên mạch.



Hình 2.9: Lựa chọn điểm cắt trong mạch lượng tử, thường tại các cổng điều khiển như CNOT [36].

Sau khi cắt, mỗi phần mạch con sẽ được biểu diễn độc lập, giảm số qubit vật lý cần thiết để thực thi.

Thực thi (Execution)

Mỗi mạch con sau khi phân tách sẽ được thực thi độc lập trên phần cứng lượng tử (hoặc giả lập). Vì các mạch con không còn liên kết trực tiếp, các qubit bị cắt sẽ được xử lý bằng cách thực hiện tất cả các tổ hợp phép đo và chuẩn bị trạng thái (tùy thuộc vào cơ sở đo).

Ví dụ, nếu cắt một qubit, cần thực hiện 2^n tổ hợp đo và chuẩn bị cho n qubit bị cắt. Số lượng phép đo tăng theo hàm mũ đối với số điểm cắt, đây là một trong những thách thức chính của kỹ thuật này.

Tái tạo (Reconstruction)

Sau khi thu thập kết quả từ tất cả mạch con, quá trình tái tạo được thực hiện dựa trên công thức:

$$\langle O \rangle = \sum_{\vec{m}} w(\vec{m}) \prod_i \langle O_i \rangle_{\vec{m}}$$

Trong đó:

- $\langle O \rangle$: kỳ vọng của observable O trên toàn mạch.
- $\langle O_i \rangle_{\vec{m}}$: kỳ vọng của observable O_i trên mạch con i với tổ hợp đo \vec{m} .
- $w(\vec{m})$: hệ số trọng số (weight) phụ thuộc vào cơ sở đo và phép tách.

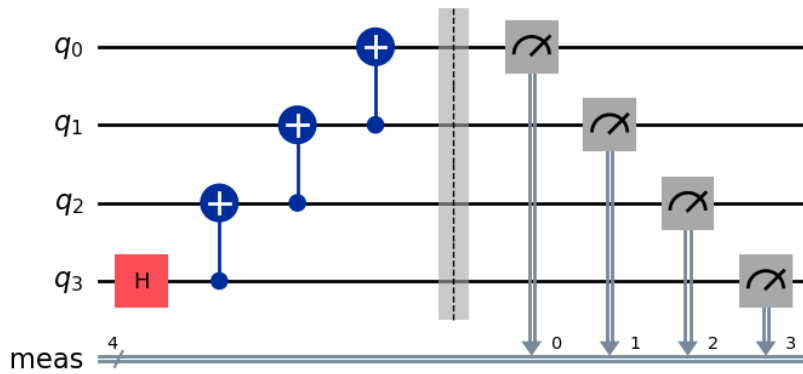
Các phương pháp phổ biến để tái tạo bao gồm: ma trận sản phẩm trạng thái (Matrix Product State), mạng tensor (Tensor Network) [39], hoặc các thuật toán tái cấu trúc dựa trên thống kê Monte Carlo.

Ví dụ minh họa

Để minh họa kỹ thuật cắt mạch lượng tử, xét một mạch lượng tử 4 qubit chuẩn bị trạng thái GHZ (Greenberger-Horne-Zeilinger) [40]:

$$|GHZ\rangle = \frac{1}{\sqrt{2}}(|0000\rangle + |1111\rangle)$$

Mạch này bao gồm một cổng Hadamard trên qubit đầu tiên và ba cổng CNOT liên tiếp để entangle các qubit còn lại (Hình 2.10).

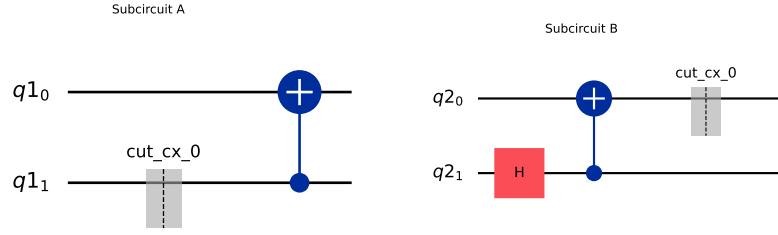


Hình 2.10: Mạch chuẩn bị trạng thái GHZ 4 qubit.

Giả sử thiết bị lượng tử hiện tại chỉ hỗ trợ tối đa 2 qubit, ta cần cắt mạch tại vị trí giữa qubit 2 và qubit 3 (giữa CNOT thứ hai và thứ ba), như trong Hình 2.11.

Sau khi cắt, mạch được chia thành:

- **Mạch con A (qubit 0, 1, 2):** thực thi Hadamard, CNOT(0,1), CNOT(1,2).



Hình 2.11: Cắt mạch GHZ 4 qubit thành hai mạch con 2 qubit.

- **Mạch con B (qubit 3, input 2):** thực thi CNOT(2,3), nhưng qubit 2 được thay thế bởi các trạng thái chuẩn bị (preparation) theo kết quả đo của mạch con A.

Để tái tạo kỳ vọng một observable O , cần thực hiện đo trên mạch con A theo tất cả các cơ sở đo (Z, X, Y) trên qubit cắt (qubit 2), và chuẩn bị trạng thái tương ứng trên mạch con B. Tổng số lượng chạy cần thiết tỉ lệ 4^k với k điểm cắt, ở đây $k = 1$, tức 4 phép đo cơ sở.

Sau khi thu thập dữ liệu từ các mạch con, giá trị kỳ vọng được tính theo công thức tái tạo:

$$\langle O \rangle = \sum_{m \in \{I, X, Y, Z\}} w_m \cdot \langle O_A^{(m)} \rangle \cdot \langle O_B^{(m)} \rangle$$

Với w_m là trọng số phụ thuộc cơ sở đo m , và $\langle O_A^{(m)} \rangle, \langle O_B^{(m)} \rangle$ là kỳ vọng trên mạch con A, B ứng với cơ sở m .

Kỹ thuật này cho phép mạch GHZ 4 qubit được chạy trên hai QPU 2 qubit, hoặc trên cùng một QPU nhưng với giới hạn 2 qubit/lượt. Tuy nhiên, chi phí phép đo tăng dần theo số điểm cắt, nhấn mạnh sự đánh đổi giữa số mạch cắt được và số phép đo để xây dựng lại kết quả.

Ưu nhược điểm

- **Ưu điểm:**
 - Cho phép thực thi các mạch lớn trên phần cứng nhỏ hơn.

- Khả năng song song hoá thực thi mạch con.
- Giảm nhu cầu phát triển thiết bị lớn trong giai đoạn NISQ.

• **Nhược điểm:**

- Chi phí phép đo tăng theo hàm mũ số điểm cắt.
- Độ chính xác phụ thuộc vào nhiễu phép đo và lỗi chuẩn bị trạng thái.
- Quá trình tái tạo đòi hỏi tính toán hậu xử lý phức tạp.

2.4.2 Ghép mạch

Circuit Knitting cung cấp cơ chế để ghép lại các kết quả lượng tử từ nhiều thành phần nhỏ, tạo thành lời giải đầy đủ cho một bài toán lớn hơn.

Kỹ thuật knitting không chỉ đơn thuần là bước nối các mạch đã chia, mà còn là một phần thiết yếu của các chiến lược tối ưu hóa cho hệ thống lượng tử phân tán, tính toán lai (hybrid quantum-classical), và ứng dụng thực tế yêu cầu mô hình hoá hệ lớn.

Định nghĩa và bối cảnh

Circuit Knitting là một nhóm kỹ thuật xử lý mạch lượng tử cho phép kết hợp (knit) các kết quả từ nhiều mạch con hoặc subroutine lượng tử, nhằm đạt được kết quả tương đương với một mạch lượng tử lớn hơn. Khác với Circuit Cutting vốn tập trung “chia nhỏ”, Circuit Knitting nhấn mạnh “kết hợp” các thành phần (có thể được xử lý trên phần cứng khác nhau hoặc ở các thời điểm khác nhau) thành một bài toán thống nhất [38], [41].

Kỹ thuật này đặc biệt hữu ích trong các ứng dụng như:

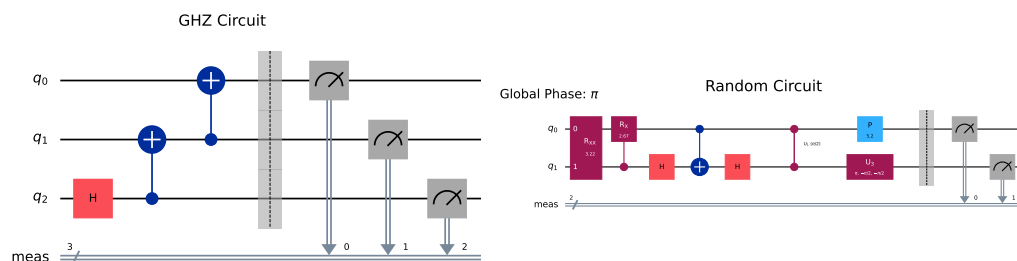
- Mô phỏng hệ vật lý với nhiều phân vùng.
- Chạy hybrid quantum-classical: chia thuật toán thành phần lượng tử và cổ điển xen kẽ.
- Thực thi mạch lượng tử trên nhiều QPU vật lý (quantum multi-processing).

Cơ chế hoạt động

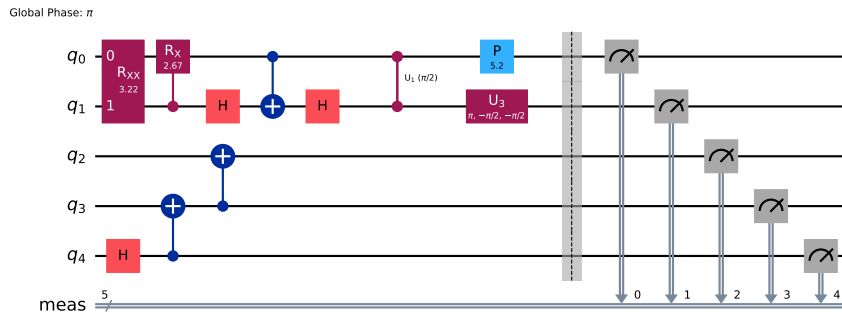
Các kỹ thuật knitting thường sử dụng các phương pháp sau:

1. **Subsystem embedding (nhúng hệ con):** chia hệ lượng tử thành các subsystem có tương tác yếu, sau đó mô phỏng subsystem trên QPU và kết hợp bằng classical embedding.
2. **Tensor Network stitching (kết nối mạng tensor):** biểu diễn trạng thái lượng tử bằng mạng tensor, ghép nối bằng contraction các tensor nhỏ hơn.
3. **Probabilistic stitching (ghép xác suất):** kết hợp kết quả từ nhiều mạch con dựa trên xác suất xảy ra các trạng thái đo, tương tự circuit cutting nhưng ở mức module thay vì qubit.

Một ví dụ minh họa được trình bày trong Hình 2.13, cho thấy cách ghép hai subsystem 2-qubit thành một hệ tổng 4-qubit.



Hình 2.12: Hai mạch lượng tử riêng biệt.



Hình 2.13: Hai mạch lượng tử riêng biệt đã được ghép lại.

Ưu nhược điểm

• Ưu điểm:

- Cho phép xử lý các subsystem độc lập, tận dụng tối đa QPU nhỏ.
- Giảm thiểu độ sâu mạch trên phần cứng.
- Kết hợp tự nhiên với các thuật toán hybrid (VD: VQE, QAOA).

• Nhược điểm:

- Đòi hỏi tính toán classical hậu xử lý phức tạp.
- Khó khăn khi hệ con có tương tác mạnh (rối lượng tử nhiều).
- Không áp dụng trực tiếp cho mọi loại bài toán lượng tử.

2.4.3 Tổng hợp các giải pháp

Bảng 2.3 giúp làm rõ sự khác biệt bản chất và phạm vi ứng dụng của hai kỹ thuật.

2.4.4 Ứng dụng thực tế

Circuit Knitting hiện đang được tích cực nghiên cứu trong các lĩnh vực:

- Mô phỏng hóa học lượng tử trên thiết bị NISQ.
- Thiết kế phần mềm lượng tử phân tán.
- Tối ưu hóa lập lịch trên hệ multi-QPU.

Bảng 2.3: So sánh Circuit Cutting và Circuit Knitting

Tiêu chí	Circuit Cutting	Circuit Knitting
Mục tiêu	Chia mạch lớn thành mạch con nhỏ hơn để chạy	Kết hợp các module/mạch con thành mạch tổng
Đơn vị thao tác	Qubit/cổng lượng tử	Module/subsystem/tensor block
Chi phí hậu xử lý	Cao (tăng hàm mũ theo số điểm cắt)	Cao (tăng theo kích thước tensor hoặc embedding)
Ứng dụng chính	Thực thi mạch lớn trên QPU nhỏ	Kết nối nhiều QPU; hybrid quantum-classical

Kỹ thuật này là nền tảng cho một số framework như *MIT Super.tech's Superstaq* [42] hoặc các nghiên cứu về distributed quantum computing.

2.5 Simulator và Emulator

Sự phát triển của các nền tảng mô phỏng lượng tử là yếu tố then chốt giúp thúc đẩy nghiên cứu và phát triển thuật toán trong bối cảnh phần cứng lượng tử vẫn đang hoàn thiện. Để phục vụ các mục tiêu khác nhau – từ thử nghiệm lý thuyết đến kiểm tra hiệu suất thực tế – hai loại công cụ chính đã được xây dựng: Simulator và Emulator.

Cả hai đều hoạt động trên máy tính cổ điển, nhưng khác biệt ở mức độ mô phỏng, các giả định vật lý được áp dụng, và khả năng phản ánh hiện thực của phần cứng lượng tử.

2.5.1 Định nghĩa và phân biệt

Simulator và **Emulator** đều là các công cụ phần mềm cho phép mô phỏng hoạt động của một hệ thống lượng tử trên phần cứng cổ điển [43]. Tuy nhiên, chúng khác nhau về mục tiêu, cách tiếp cận và độ trung thực trong mô phỏng.

- **Simulator:** Mô phỏng lý tưởng trạng thái lượng tử, bỏ qua hoặc đơn giản hóa tác động của nhiễu, lỗi đo, và các hạn chế vật lý của phần cứng. Simulator thường được sử dụng để nghiên cứu lý thuyết, kiểm thử thuật toán hoặc trực quan hóa hiện tượng lượng tử mà không chịu ảnh hưởng của lỗi thực tế.
- **Emulator:** Mô phỏng trạng thái lượng tử với mục tiêu tái hiện chính xác hành

vi của một thiết bị lượng tử cụ thể, bao gồm các thông số nhiễu, decoherence, crosstalk và lỗi vận hành. Emulator gần với điều kiện thực tiễn hơn, hỗ trợ đánh giá hiệu suất thuật toán trên phần cứng cụ thể trước khi triển khai thực tế.

2.5.2 Vai trò và ứng dụng

Simulator và emulator đóng vai trò quan trọng trong phát triển ứng dụng lượng tử, đặc biệt trong giai đoạn NISQ khi phần cứng còn hạn chế:

- **Simulator:**

- Kiểm tra tính đúng đắn của thuật toán lượng tử trong môi trường lý tưởng.
- Hỗ trợ giảng dạy, trực quan hóa các hiện tượng lượng tử.
- Thử nghiệm các kỹ thuật tối ưu mạch (circuit optimization) mà không bị giới hạn phần cứng.

- **Emulator:**

- Đánh giá tác động của nhiễu, lỗi phép đo trên hiệu suất thuật toán.
- Giúp lựa chọn kiến trúc phần cứng phù hợp cho một bài toán cụ thể.
- Chuẩn bị mạch lượng tử gần nhất với khả năng thực thi của một QPU cụ thể.

Bảng 2.4 tóm tắt sự khác biệt giữa simulator và emulator:

Bảng 2.4: So sánh Simulator và Emulator trong tính toán lượng tử

Tiêu chí	Simulator	Emulator
Mục tiêu	Mô phỏng lý tưởng trạng thái lượng tử	Mô phỏng hành vi thiết bị lượng tử thực tế
Mô hình lỗi	Không (hoặc tối giản)	Bao gồm nhiễu, decoherence, crosstalk
Độ chính xác vật lý	Cao trong điều kiện lý tưởng	Gần với thực tế phần cứng
Ứng dụng chính	Kiểm thử thuật toán lý thuyết, giảng dạy	Đánh giá thuật toán trên phần cứng cụ thể
Yêu cầu tài nguyên	Tăng theo số qubit (exponential)	Tăng theo số qubit và phức tạp lỗi

2.5.3 Mỗi quan hệ với phần cứng lượng tử

Cả simulator và emulator đều là công cụ trung gian giữa lý thuyết và phần cứng thực tế. Trong quy trình phát triển ứng dụng lượng tử:

1. Thuật toán được kiểm nghiệm ban đầu trên simulator để đảm bảo tính chính xác toán học.
2. Sau khi tinh chỉnh, thuật toán được thử nghiệm trên emulator để đánh giá khả năng hoạt động trong điều kiện phần cứng cụ thể.
3. Cuối cùng, thuật toán được triển khai trên phần cứng lượng tử thực tế.

Sự kết hợp giữa simulator và emulator giúp rút ngắn chu trình phát triển, giảm rủi ro khi triển khai và tối ưu hóa hiệu suất ngay từ giai đoạn thiết kế.

2.6 Thư viện mô phỏng lượng tử

Sự đa dạng về nhu cầu nghiên cứu, thử nghiệm thuật toán và mô phỏng phần cứng lượng tử đã thúc đẩy sự ra đời của nhiều thư viện phần mềm chuyên dụng. Các thư viện mô phỏng lượng tử không chỉ cung cấp công cụ để thao tác với mạch lượng tử ở mức logic, mà còn hỗ trợ khả năng mô phỏng hành vi của phần cứng thực, tích hợp các thông số nhiễu, giới hạn qubit, và lỗi cổng.

2.6.1 Giới thiệu

Trong bối cảnh phần cứng lượng tử còn đang phát triển, đặc biệt trong kỷ nguyên NISQ (*Noisy Intermediate-Scale Quantum*), các công cụ mô phỏng lượng tử đóng vai trò thiết yếu. Chúng không chỉ giúp kiểm nghiệm giải thuật trước khi triển khai trên phần cứng thực, mà còn là “phòng thí nghiệm ảo” cho những người bước đầu tiếp cận lĩnh vực đòi hỏi nền tảng toán học và vật lý chuyên sâu.

2.6.2 Các thư viện mô phỏng lượng tử phổ biến

Dưới đây là phần mô tả riêng cho một số thư viện hỗ trợ cho mô phỏng lượng tử

Bảng 2.5: Các thư viện mô phỏng lượng tử

Thư viện	Tính năng chính	Khả năng tích hợp	Hỗ trợ phần cứng	Ứng dụng chính
Qiskit [10]	Giải lập trạng thái, tối ưu hóa mạch, hỗ trợ qubit nhiều	Tích hợp IBM Quantum Experience	Hỗ trợ phần cứng IBM Quantum (Falcon, Hummingbird)	Nghiên cứu học thuật, tối ưu hóa thuật toán
Cirq [28]	Xây dựng và tối ưu hóa mạch lượng tử, hỗ trợ Sycamore	Tích hợp với Google Quantum AI	Phần cứng Google Sycamore	Tối ưu hóa, thử nghiệm thuật toán lượng tử
PennyLane [44]	Học máy lượng tử, tối ưu hóa biến phân	Tích hợp TensorFlow, PyTorch	Đa dạng phần cứng (IBM, Google, Rigetti)	Phát triển mô hình học máy lượng tử
Amazon Braket [26]	Mô phỏng mạch lượng tử, kết nối đa dạng phần cứng	Tích hợp AWS (Amazon Web Services)	IonQ, Rigetti, D-Wave	Ứng dụng công nghiệp, tối ưu hóa logistics
QuEST [45]	Hiệu năng cao trên siêu máy tính, hỗ trợ mô phỏng qubit lớn	Hỗ trợ giao tiếp phân tán	Không hỗ trợ phần cứng thực	Nghiên cứu mô phỏng quy mô lớn

- **Qiskit (IBM):** Thư viện mã nguồn mở do IBM phát triển, hỗ trợ lập trình, mô phỏng và thực thi mạch lượng tử [10]. Qiskit tích hợp trực tiếp với nền tảng IBM Quantum Experience, cho phép thử nghiệm trên phần cứng thật như Falcon, Hummingbird.
 - Tính năng: mô phỏng trạng thái lượng tử, tối ưu hóa mạch, mô hình hóa lỗi.
 - Ứng dụng: nghiên cứu thuật toán lượng tử, giáo dục, phát triển ứng dụng lượng tử thực nghiệm.
- **Cirq (Google):** Thư viện mã nguồn mở của Google, thiết kế để xây dựng và tối ưu mạch lượng tử trên chip Sycamore [28]. Cirq hỗ trợ mô phỏng cấu trúc cụ thể

của phần cứng thực tế.

- Tính năng: mô phỏng trạng thái lượng tử, mapping lên chip Sycamore, hỗ trợ tối ưu hóa low-level.
- Ứng dụng: phát triển thuật toán phù hợp với kiến trúc Google Quantum AI.
- **PennyLane (Xanadu):** Thư viện chuyên dụng cho học máy lượng tử và mạch biến phân, cho phép tích hợp với các nền tảng học máy cổ điển như PyTorch, TensorFlow [44].
 - Tính năng: hỗ trợ hybrid quantum-classical, tối ưu mạch biến phân, giao diện thống nhất nhiều backend.
 - Ứng dụng: xây dựng mô hình QML, VQE, QAOA và tối ưu hóa lượng tử.
- **Amazon Braket SDK:** Nền tảng mô phỏng và truy cập phần cứng lượng tử đa dạng trên dịch vụ AWS. Braket hỗ trợ mô phỏng mạch lượng tử trên cloud và kết nối nhiều backend như IonQ, Rigetti, D-Wave [46].
 - Tính năng: mô phỏng trên cloud, kết nối phần cứng đa nhà cung cấp, công cụ theo dõi hiệu năng mạch.
 - Ứng dụng: triển khai công nghiệp, tối ưu hóa logistics và bài toán tối ưu tổ hợp.
- **QuEST (Quantum Exact Simulation Toolkit):** Thư viện hiệu năng cao hỗ trợ mô phỏng hệ lượng tử lớn trên siêu máy tính hoặc môi trường phân tán [45].
 - Tính năng: mô phỏng trạng thái lý tưởng và có nhiễu, hỗ trợ phân tán, tối ưu cho HPC.
 - Ứng dụng: nghiên cứu lý thuyết, mô phỏng hóa học lượng tử, tối ưu hóa lượng tử quy mô lớn.

2.6.3 Sự lựa chọn

Sau khi phân tích và so sánh các thư viện mô phỏng lượng tử, nhóm quyết định lựa chọn **Qiskit** làm thư viện chính để triển khai các mô-đun trong hệ thống mô phỏng. Quyết định này dựa trên các lý do sau:

- **Tính phổ biến và cộng đồng hỗ trợ mạnh:** Qiskit có cộng đồng người dùng và nhà phát triển rộng lớn, tài liệu phong phú, dễ tra cứu và tiếp cận.
- **Khả năng tối ưu hóa mạch và mô hình hóa lỗi:** hỗ trợ nhiều tính năng cần thiết cho nghiên cứu lập lịch lượng tử, đặc biệt trong môi trường NISQ. Hỗ trợ tạo các máy mô phỏng giống các máy của hệ thống IBM.
- **Giao diện lập trình thân thiện:** dễ tích hợp với các thư viện Python khác, thuận tiện cho việc phát triển, trực quan hóa và mở rộng hệ thống.

Việc lựa chọn Qiskit phù hợp với mục tiêu của đề tài, đảm bảo tính linh hoạt, khả năng mở rộng và hỗ trợ các yêu cầu kỹ thuật đặt ra trong quá trình hiện thực hệ thống mô phỏng.

2.7 Tóm tắt chương

Chương này đã trình bày tổng quan về các công cụ mô phỏng lượng tử, bao gồm đặc điểm, tính năng, khả năng tích hợp và phạm vi ứng dụng của từng thư viện. Những kiến thức nền tảng này nhấn mạnh vai trò quan trọng của việc phát triển một hệ thống mô phỏng quá trình lập lịch trong môi trường lượng tử không đồng nhất. Chương tiếp theo sẽ trình bày các công trình nghiên cứu liên quan giúp người đọc có được góc nhìn chung về đề án.

Chương 3

Nghiên cứu liên quan

Trong chương 3, các công trình nghiên cứu liên quan sẽ được tổng hợp và phân tích nhằm cung cấp cái nhìn toàn diện về bối cảnh và các hướng tiếp cận hiện tại trong lĩnh vực tính toán lượng tử. Chương này trình bày các nghiên cứu nổi bật trong lĩnh vực định thời tài nguyên lượng tử.

3.1 MILQ - Multithreaded Parallelism for Heterogeneous Clusters of QPUs

Nghiên cứu giới thiệu MILQ, một framework kết hợp cắt mạch (circuit cutting) và Mixed Integer Linear Programming (MILP) để tối ưu hóa lập lịch mạch lượng tử trên các cụm QPUs khác nhau [9]. MILQ được thiết kế nhằm phân bổ các subcircuit lên các QPU khác nhau, đồng thời tận dụng đa luồng (multithreading) để xử lý song song các tác vụ, giảm tổng thời gian thực thi.

Tính năng nổi bật

- **Kết hợp cắt mạch và MILP** để lập lịch tối ưu các mạch lượng tử vượt quá khả năng phần cứng hiện tại.
- **Hỗ trợ multithreading**, xử lý nhiều instance mạch song song trên cùng một QPU.
- **Tích hợp circuit reassembly** thông qua hậu xử lý cổ điển để tái cấu trúc phân phối xác suất đầu ra.

- **Giảm makespan lên đến 26%** so với baseline trong thử nghiệm.

Hạn chế

- **Chi phí tính toán MILP cao**, thời gian giải MILP tăng nhanh theo số subcircuit và số QPU.
- **Hậu xử lý cổ điển tốn tài nguyên**, đặc biệt với các mạch có nhiều lần cắt.
- **Chưa kiểm chứng trên hệ thống đa backend thật**, chủ yếu thử nghiệm mô phỏng.

3.2 SCIM MILQ

SCIM MILQ đóng vai là một công cụ quản lý tài nguyên để định thời các công việc lượng tử trong hạ tầng tính toán hiệu năng cao [30]. Công cụ này kết hợp các kỹ thuật định thời HPC truyền thống với những phương pháp lượng tử như cắt mạch để quản lý và hiện thực hiệu quả các mạch lượng tử trên phần cứng đang có.

Tính năng nổi bật

- **Kết hợp các kỹ thuật định thời với các phương pháp lượng tử như cắt mạch.** Cụ thể là phương pháp *Scatter Search* và *Học tăng cường (Reinforcement Learning)* có bao gồm khả năng cắt mạch.
- **Sử dụng các đối tượng *proxy*** để xử lý hiệu quả hơn, bao gồm các thông tin quan trọng của mạch và ước lượng tài nguyên.
- **Sử dụng kỹ thuật *backfilling*** để các mạch nhỏ bỏ qua cơ chế định thời nếu như hàng đợi thiết bị có đủ không gian để thực thi.

Hạn chế

- **SCIM MILQ mới chỉ được kiểm thử trên môi trường độc lập với tải lượng công việc không cao.** Bước tiếp theo của nghiên cứu này là tích hợp vào các hệ

thống tính toán hiệu năng cao để kiểm tra hiệu năng dưới những điều kiện thực tế hơn.

- **Vẫn còn cần phải tối ưu, cải thiện hiệu năng cho SCIM MILQ.** Bao gồm tinh chỉnh giải thuật định thời cho một số trường hợp nhất định, áp dụng kỹ thuật cắt mạch nâng cao hơn hay tích hợp kỹ thuật giảm thiểu độ nhiễu sẽ giúp hiệu quả của hệ thống được nâng cao.

3.3 CutQC

CutQC là một cách tiếp cận mới trong tính toán lai lượng tử-truyền thống được thiết kế để đo lường các mạch lượng tử lớn không thể xử lý bằng máy tính truyền thống lẫn máy tính lượng tử hiện tại xử lý được [37].

Tính năng nổi bật

- **Mở rộng kích thước mạch lượng tử** có thể chạy trên thiết bị NISQ và giả lập truyền thống bằng cách kết hợp chúng lại. Phương pháp này cho phép thực thi mạch lượng tử nhiều hơn hai lần kích thước của phần cứng máy tính hiện tại và gấp nhiều lần giới hạn của giả lập truyền thống.
- **Cải thiện độ tin cậy** khi thực thi mạch trên thiết bị NISQ. CutQC cho thấy mức cải thiện trung bình 21% đến 47% đối với tổn thất χ^2 cho các kiểm chuẩn khác nhau.
- **Tăng tốc** thời gian thực thi mạch lượng tử so với mô phỏng cổ điển. CutQC sử dụng máy tính lượng tử làm bộ đồng xử lý để đạt tốc độ chạy nhanh hơn từ 60X đến 8600X so với mô phỏng cổ điển cho các kiểm chuẩn khác nhau.

Hạn chế

- **Chi phí hậu xử lý cổ điển.** Mặc dù CutQC cung cấp tốc độ chạy hơn giả lập cổ điển đối với một số mạch nhất định, nhưng hậu xử lý cổ điển vẫn có thể tốn nhiều tài nguyên tính toán, đặc biệt là với mạch lớn nhiều lần cắt. Số lượng tích Kronecker cần dùng để tái cấu trúc mạch tăng theo cấp số nhân với số lần cắt (4^K

với K là số lần cắt). Điều này có nghĩa rằng chi phí hậu xử lý cổ điển có thể nặng hơn lợi ích khi sử dụng các máy tính lượng tử nhỏ.

- **Thiết bị NISQ có giới hạn kết nối qubit và độ trung thực của các cổng.** Hiệu năng của CutQC phụ thuộc vào sự khả dụng của máy tính lượng tử với đủ kết nối qubit và độ trung thực của cổng. Các thiết bị NISQ hiện tại có hạn chế ở cả hai mảng này, dẫn tới có thể giới hạn kích cỡ và độ phức tạp của mạch mà CutCQ có thể đo lường được.
- **Hội tụ đầu ra của mạch con** cũng là một vấn đề đáng lưu tâm. Độ chính xác của quá trình tái cấu trúc của CutQC phụ thuộc vào sự hội tụ của các phân phối xác suất thu được từ việc thực thi các mạch con. Nếu số lần tính toán mạch con không đủ, xác suất tái cấu trúc có thể không được chính xác thậm chí cho ra kết quả âm.
- **Khả năng mở rộng cho các mạch đầu ra dày đặc.** Với các mạch có xác suất đầu ra dày đặc, trong khi phần lớn các trạng thái có xác suất khác 0, truy vấn định nghĩa động (DD) của CutQC có thể cần rất nhiều lần đệ quy để đạt được mức độ chính xác mong muốn. Điều này có thể dẫn tới tăng thời gian chạy và sử dụng bộ nhớ, đặc biệt cho các mạch rất lớn.

3.4 QOS - Quantum Operating System

Gần đây, một nghiên cứu đầy tham vọng với mong muốn hiện thực một hệ điều hành lượng tử có thể quản lý tài nguyên lượng tử hiệu quả và giảm thiểu những hạn chế của các thiết bị NISQ hiện nay [47].

Tính năng nổi bật

- QOS có kiến trúc gồm hai phần như sau:
 - Trình biên dịch tối ưu hóa các ứng dụng lượng tử để thực thi trên phần cứng.
 - Hệ thống thời gian chạy định thời ứng dụng để tối ưu hóa thời gian đợi và tài nguyên sử dụng.

- Hệ thống được đánh giá rộng rãi bằng những thiết bị lượng tử thực và thể hiện được những điểm tốt hơn cả về chất lượng kết quả và độ hiệu quả của hệ thống, đề cập tới thử thách quan trọng như độ trung thực, hiện thực hóa và mất cân bằng tải trong tính toán lượng tử đám mây.
- Điểm đổi mới của nghiên cứu này là cách tiếp cận thống nhất, kết hợp giữa tối ưu ở cấp độ biên dịch và quản lý tài nguyên lúc chạy để đề cập tới hạn chế của công nghệ NISQ một cách toàn diện.

Hạn chế

- **Tính toán đa nhiệm của QOS mặc dù tăng tính hiệu quả sử dụng, nhưng có thể dẫn tới hình phạt độ trung thực.** Mặc dù trình đa nhiệm của QOS phần đầu giảm thiểu mất độ trung thực thông qua các kiểm tra độ tương thích và đóng gói các kernel cẩn thận nhưng vẫn tồn tại các rủi ro cố hữu khiến giảm độ trung thực. Một số tác nhân gây ra việc này như hiện tượng giao thoa giữa các qubit hay các hạn chế khi phân bổ những qubit chất lượng cao. Nghiên cứu chỉ ra rằng độ trung thực mất khoảng 9.6% khi so với việc các mạch chạy riêng lẻ và trong một số trường hợp nhất định có thể mất lên tới 18%.
- **Các đo lường của QOS chủ yếu tập trung trên IBM Falcon r5.11 QPUs.** Hiệu năng và hiệu quả của QOS trên nền tảng hay hệ thống lượng tử khác chưa được khám phá trong nghiên cứu.

3.5 Bộ định thời phân bổ nhận biết độ nhiễu và thời gian (NoTADS)

Đây là một bộ bộ định thời phân bổ nhận biết độ nhiễu và thời gian (NoTADS) cho các mạch lượng tử. NoTADS tận dụng việc cắt mạch để phân chia các mạch thành các mạch con nhỏ hơn, giảm thiểu độ nhiễu [7].

Tính năng nổi bật

- **Giảm độ nhiễu thông qua cắt mạch.** NoTADS sử dụng cắt mạch để chia một mạch lượng tử lớn thành các mạch con nhỏ hơn. Điều này giảm tác động của độ nhiễu, vì các mạch nhỏ hơn ít bị lỗi hơn.
- **Định thời phân tán trên nhiều phần cứng.** NoTADS định thời những mạch con này trên nhiều phần cứng lượng tử, tối đa hóa song song và giảm thiểu thời gian thực thi.
- **Lựa chọn phần cứng nhận biết độ nhiễu.** Hệ thống sử dụng *mapomatic*, một công cụ chấm điểm phần cứng dựa trên độ nhiễu. NoTADS ưu tiên định thời các mạch con dựa trên phần cứng ít độ nhiễu nhất có sẵn.
- **Tối ưu hóa theo thời gian.** NoTADS cho phép người dùng đặt thời gian thực thi tối đa (τ) cho mỗi phần cứng, đảm bảo sử dụng tài nguyên hiệu quả và giải quyết các hạn chế tiềm ẩn.
- **Tối ưu hóa độ trung thực.** Mục tiêu ban đầu của NoTADS là để tối ưu hóa độ trung thực (độ chính xác) của toàn bộ phép tính. Việc này đạt được bằng cách kết hợp giảm độ nhiễu thông qua cắt mạch và lựa chọn phần cứng thông minh.
- **Hậu xử lý cổ điển.** Sau khi thực thi các mạch con, NoTADS sử dụng hậu xử lý cổ điển để tái cấu trúc lại phân bố xác suất hoàn chỉnh của mạch chưa cắt ban đầu.

Hạn chế

- **Chi phí hậu xử lý cổ điển.** NoTADS dựa vào cắt mạch, dẫn tới nhu cầu hậu xử lý cổ điển để tái tạo phân phối xác suất từ kết quả mạch con. Thời gian cần thiết cho quá trình hậu xử lý tăng theo cấp số nhân của số lần cắt, khiến nó kém hiệu quả đối với các mạch cần nhiều phân vùng.
- **Khả năng áp dụng hạn chế cho các mạch sâu hoặc không cân bằng.** Việc triển khai hiện tại của NoTADS sử dụng hàm mục tiêu tuyến tính trong quá trình tối ưu hóa. Cách này hiệu quả khi các mạch con có kích thước và độ sâu bằng nhau. Đối với các mạch tạo ra các mạch con mất cân bằng (thay đổi đáng kể về chiều

rộng hay chiều sâu), hàm mục tiêu tuyến tính có thể không đảm bảo định thời tối ưu. Trong những trường hợp như vậy, có thể cần một hàm mục tiêu phi tuyến tính, dẫn tới tăng độ phức tạp của bài toán tối ưu.

- **Phụ thuộc vào độ chính xác của đặc tính độ nhiễu.** Hiệu quả của NoTADS phụ thuộc rất nhiều vào *mapomatic* để chấm điểm độ nhiễu của phần cứng. Đặc tính nhiễu không chính xác hoặc lỗi thời có thể dẫn tới các quyết định định thời không tối ưu, giảm mức độ trung thực tổng thể.
- **Giá trị độ nhiễu của phần cứng lượng tử có thể thay đổi theo thời gian.** **NoTADS hiện tại không tính tới biến động trong hiệu suất phần cứng**, khả năng dẫn tới độ trung thực nếu độ nhiễu thay đổi đáng kể giữa lúc định thời và thực thi.
- **Chưa cân nhắc tới độ trễ của hàng đợi.** Nghiên cứu thừa nhận độ trễ hàng đợi ở phần cứng không được xem xét trong quá trình triển khai NoTADS. Trong trường hợp với truy cập phần cứng được dùng chung, thời gian hàng đợi có thể ảnh hưởng tới thời gian thực thi và hiệu suất tổng thể, một yếu tố chưa được tính tới trong quá trình tối ưu hiện tại.

3.6 MTMC - Multi-Task Multi-Chip Scheduling

MTMC (Multi-Task Multi-Chip Scheduling) là một framework lập lịch đa nhiệm trên nhiều chip lượng tử, nhằm phân bổ tự động các tác vụ lượng tử lên nhiều phần cứng trong môi trường điện toán lượng tử đám mây [12]. Framework này được triển khai ở giai đoạn backend của quantum compiler, giúp tối ưu hoá hiệu suất sử dụng tài nguyên và tăng throughput toàn hệ thống.

Tính năng nổi bật

- **Quản lý đa nhiệm trên nhiều chip.** MTMC cho phép ánh xạ đồng thời nhiều tác vụ lên nhiều chip lượng tử, khai thác song song tài nguyên phần cứng lượng tử.
- **Cơ chế dynamic window và ưu tiên động.** Khi một tác vụ không thể mapping thành công, nó được đưa trở lại hàng đợi với mức ưu tiên cao hơn trong vòng lặp kế tiếp.

- **Cấu trúc “chip slice”.** Mỗi chip được chia thành các lát tài nguyên (chip slice) để tăng tính linh hoạt khi phân bổ, giúp tận dụng tối đa tài nguyên phần cứng.
- **Giải thuật depth-aware.** Framework kiểm tra độ sâu logic của mạch (critical path depth) và độ trung thực (fidelity) để đảm bảo một tác vụ có thể thực thi trên một chip mà không vượt quá giới hạn T_1 (coherence time).
- **Tự động hoá phân bổ chip.** Người dùng không cần chỉ định chip, hệ thống tự động chọn chip tối ưu.
- **Tăng throughput và giảm queue time.** Thử nghiệm cho thấy throughput của hệ thống tăng đáng kể, đặc biệt khi số tác vụ đồng thời (epsilon) tăng.

Hạn chế

- **Hiệu suất giảm dần khi epsilon hoặc số vòng lặp tăng cao.** Tốc độ cải thiện throughput có xu hướng giảm khi tăng quá nhiều tác vụ đồng thời hoặc số vòng lặp lớn.
- **Chưa thử nghiệm trên phần cứng lượng tử thật.** Framework được đánh giá trên backend mô phỏng (quantum fake chips trong Qiskit), chưa kiểm chứng trên thiết bị lượng tử thực tế.
- **Chưa xét đến yếu tố noise dynamics.** Hệ thống chưa tính đến sự thay đổi độ nhiễu (noise fluctuation) của phần cứng theo thời gian.
- **Giả định về độ sâu và thời gian coherence.** Giải thuật depth-aware dựa vào ước lượng T_1 và độ sâu mạch, có thể chưa phản ánh đầy đủ khi phần cứng có topological constraints phức tạp hơn.

3.7 Định thời nhận biết tài nguyên (Resource-aware scheduling)

Ở nghiên cứu này, nhóm tác giả hướng tới một công cụ định thời nhiều mạch lượng tử trên một thiết bị phần cứng để cải thiện thông lượng và khả năng sử dụng phần cứng mà không làm giảm chất lượng của kết quả tính toán [8].

Tính năng nổi bật

- Sắp xếp các qubit có tương tác với nhau theo **lân cận gần nhất (NN)** để **giảm thiểu độ nhiễu và đảm bảo bố trí chất lượng cao**. Điều này liên quan tới việc giảm thiểu số lượng cổng SWAP và chọn bố trí với cấu hình nhiễu tối thiểu.
- **Một bộ đệm khoảng cách để ngăn cản giao tiếp chéo** giữa các qubit. Điều này khiến cho các mạch không được đặt quá gần nhau, giảm thiểu khả năng bị nhiễu.
- Nghiên cứu đưa ra **công thức lập trình số nguyên tuyến tính (ILP)** để **tìm vị trí tối ưu của nhiều mạch trên phần cứng**. ILP xem xét độ nhiễu của các bố trí, khoảng cách đệm và điểm tổng thể của vị trí, đảm bảo rằng bố cục được chọn nằm trong giới hạn chỉ định trước về độ nhiễu.
- Bài toán định thời thuộc kiểu NP-Hard, nghĩa là tìm một giải pháp tối ưu có thể tốn kém về mặt tính toán. Để giải quyết vấn đề này, nghiên cứu đề ra một **giải thuật tham lam dựa trên lý thuyết đồ thị**. Giải thuật này xây dựng một đồ thị thích hợp với các đỉnh biểu thị các mạch và bố cục của chúng, và các cạnh biểu thị độ tương thích dựa trên khoảng cách đệm và độ nhiễu. Sau đó một cụm tối đa được tìm thấy từ đồ thị xác định tập mạch có thể thực hiện đồng thời.

Hạn chế

- **Có sự đánh đổi giữa số giữa số bố cục cho phép được xử lý và độ trung thực**. Nếu số phần trăm bố cục được cho phép lớn, nhiều mạch có thể được thực thi nhưng có thể độ trung thực sẽ thấp. Ngược lại, nếu số phần trăm bố cục được cho phép nhỏ, ít mạch được thực thi hơn nhưng độ trung thực cũng lớn hơn.
- **Giải thuật tham lam**. Giải thuật thời gian đa thức này cung cấp giải pháp nhanh hơn nhưng có thể không tối ưu.

3.8 Bảng tổng hợp các công trình nghiên cứu

Bảng 3.1: Tổng hợp so sánh các công trình nghiên cứu liên quan

Nghiên cứu	Tính năng nổi bật	Hạn chế	Môi trường áp dụng	Lựa chọn hiện thực
MILQ [9]	Circuit cutting + MILP + multi-threading; reduce makespan 26%; circuit reassembly	MILP cost high; classical postprocessing heavy; chưa kiểm chứng multi-backend thật	HPC cluster with heterogeneous QPUs	Đã hiện thực và thêm giải thuật baseline của bài báo
SCIM MILQ [30]	Kết hợp Scatter Search, RL, backfilling; cắt mạch; proxy estimation	Mới kiểm thử độc lập; chưa tích hợp HPC; cần tối ưu thêm	HPC, cloud, multi-backend	Có tiềm năng cao, chưa hiện thực vì cần kiến thức AI sâu
CutQC [37]	Mở rộng kích thước mạch; tăng độ tin cậy; tăng tốc 60x–8600x	Hậu xử lý nặng; phụ thuộc noise, hội tụ xác suất	NISQ, hybrid classical–quantum	Không hiện thực vì không có giải thuật scheduling rõ ràng
QOS [47]	Compiler + runtime tích hợp; tối ưu tài nguyên; quản lý đa nhiệm	Giảm fidelity khi đa nhiệm (9.6–18%); chỉ thử nghiệm IBM Falcon	Quantum cloud, quantum OS	chưa hiện thực
Xem tiếp ở trang sau				

Bảng 3.1 – Nối tiếp từ trang trước

Nghiên cứu	Tính năng nổi bật	Hạn chế	Môi trường áp dụng	Lựa chọn hiện thực
NoTADS [7]	Noise-aware scheduling; cut-based; hardware selection mapomatic; limit runtime	Hậu xử lý nặng; không xét biến động noise, queue latency	Multi-hardware, distributed quantum	Đã triển khai trong môi trường giả lập
Resource-aware scheduling [8]	ILP placement; buffer spacing; heuristic graph clustering	Trade-off fidelity vs throughput; heuristic không tối ưu toàn cục	Single hardware, multi-circuit placement	Chưa hiện thực
MTMC [12]	Multi-tasking multi-chip mapping; dynamic window; chip slice; depth-aware mapping	Chưa kiểm chứng trên hardware thật; chưa xét noise dynamics; throughput giảm khi loops lớn	Quantum cloud, multi-chip backend	Đã triển khai trong môi trường giả lập

3.9 Tóm tắt chương

Chương này đã trình bày và phân tích các công trình nghiên cứu tiêu biểu liên quan đến định thời tài nguyên lượng tử, bao gồm các phương pháp và hệ thống như SCIM MILQ [30], CutQC [37], QOS [47], NoTADS [7] và các mô hình định thời nhận biết tài nguyên. Mỗi nghiên cứu được phân tích về tính năng nổi bật và những hạn chế tồn tại, nhằm làm rõ các hướng tiếp cận hiện nay trong lĩnh vực lập lịch lượng tử.

Từ tổng quan các nghiên cứu, có thể nhận thấy lĩnh vực định thời tài nguyên lượng tử đang phát triển đa dạng cả về phương pháp luận và phạm vi ứng dụng: từ mô hình lập lịch độc lập, kết hợp phương pháp lai lượng tử-cổ điển, đến việc xây dựng hệ điều hành

lượng tử tích hợp. Tuy nhiên, một khoảng trống nghiên cứu vẫn tồn tại khi hiện chưa có công trình nào thực hiện phân tích tổng thể, so sánh có hệ thống giữa các phương pháp hiện có, cũng như đánh giá khả năng ứng dụng trong các môi trường đa backend hoặc môi trường có tải động.

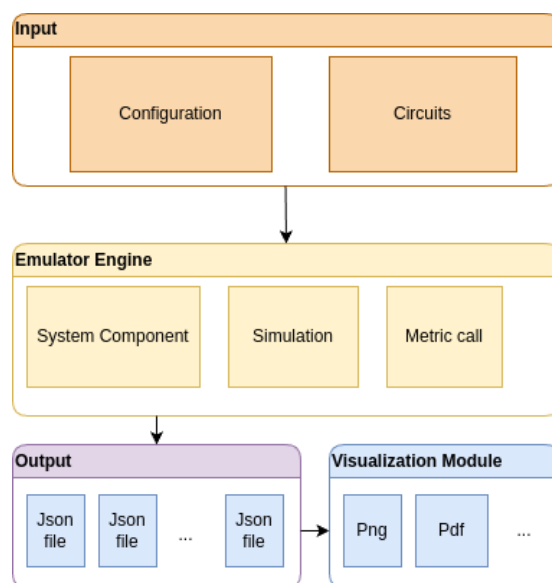
Chương 4

Kiến trúc hệ thống

Chương 4 mô tả một công cụ linh hoạt hỗ trợ việc so sánh và đánh giá các giải thuật lập lịch lượng tử. Để đạt được mục tiêu này, chúng tôi đã phát triển một hệ mô phỏng cho phép nghiên cứu hiệu suất của nhiều phương pháp lập lịch khác nhau.

4.1 Kiến trúc tổng quan

Kiến trúc tổng quan của hệ thống mô phỏng, như minh họa trong Hình 4.1, bao gồm bốn thành phần chính: **Input**, **Emulator Engine**, **Output**, và **Visualization Module**. Thiết kế này hướng tới tiêu chí linh hoạt, mở rộng, cho phép dễ dàng tích hợp và thử nghiệm các phương pháp lập lịch mới trong tương lai.



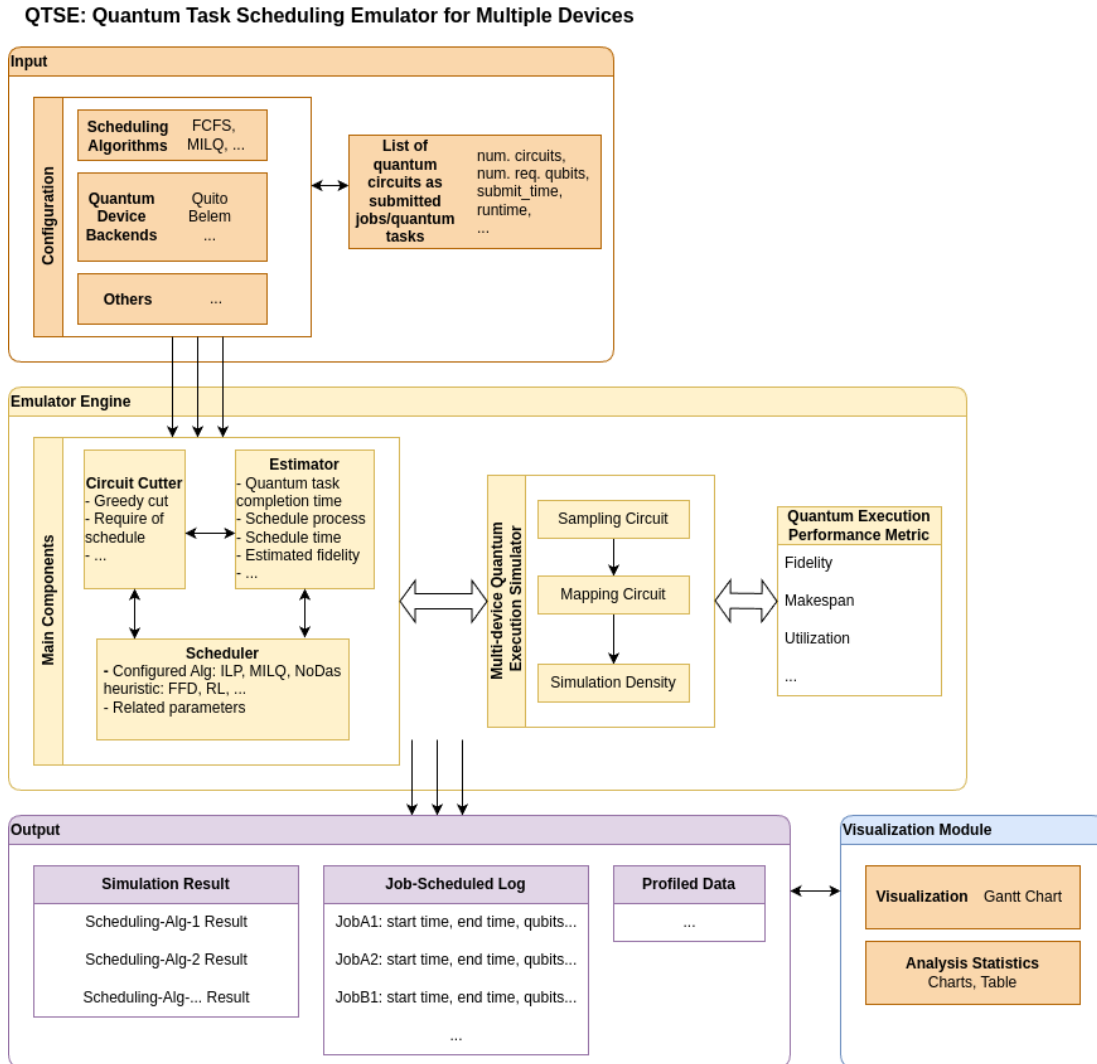
Hình 4.1: Kiến trúc tổng quan hệ thống.

- **Input** là thành phần đầu vào, cung cấp dữ liệu cần thiết cho quá trình mô phỏng. Input bao gồm hai phần: *Configuration* chứa các tham số cấu hình như số lượng qubit, loại thuật toán, môi trường thực thi; và *Circuits* lưu trữ thông tin về mạch lượng tử cần mô phỏng. Cách tổ chức này đảm bảo tính mô-đun, dễ mở rộng khi thử nghiệm các mô hình hoặc cấu hình khác nhau.
- **Emulator Engine** là lõi xử lý của hệ thống, chia thành ba module: *System Component*, *Simulation*, và *Metric calculation*. Module *System Component* chịu trách nhiệm khởi tạo và quản lý các thành phần hệ thống; *Simulation* thực hiện quá trình mô phỏng dựa trên đầu vào; và *Metric calculation* thu thập, tính toán các chỉ số đánh giá hiệu suất như thời gian xử lý, độ trễ, độ chính xác. Việc tách biệt chức năng này giúp nâng cao khả năng bảo trì, phát triển và tích hợp các cải tiến trong tương lai.
- **Output** lưu trữ kết quả mô phỏng dưới dạng các tệp .json, mỗi tệp có thể đại diện cho một lần chạy mô phỏng, một cấu hình hoặc một thuật toán cụ thể. Việc sử dụng định dạng JSON đảm bảo tính có cấu trúc, dễ phân tích, dễ tích hợp với các công cụ xử lý và báo cáo.
- **Visualization Module** thực hiện trực quan hóa kết quả mô phỏng dựa trên dữ liệu đầu ra. Module này chuyển đổi các tệp JSON thành biểu đồ (.png), báo cáo (.pdf) hoặc các định dạng trực quan khác, hỗ trợ người dùng phân tích, so sánh và trình bày kết quả mà không cần thao tác trực tiếp với dữ liệu thô.

Luồng xử lý của hệ thống tuân theo pipeline rõ ràng: dữ liệu từ Input được truyền vào Emulator Engine để xử lý; kết quả thu được lưu tại Output; và cuối cùng được Visualization Module trực quan hóa. Thiết kế này đảm bảo tính tuần tự, mạch lạc, đồng thời hỗ trợ tốt khả năng mở rộng, tái sử dụng và tích hợp trong các nghiên cứu lập lịch lượng tử.

4.2 Mô hình chi tiết

Hình 4.2 minh họa chi tiết kiến trúc tổng thể của hệ thống **QTSE (Quantum Task Scheduling Emulator for Multiple Devices)** — một công cụ mô phỏng lập lịch tác



Hình 4.2: Mô hình chi tiết của hệ thống.

vụ lượng tử trên nhiều thiết bị. Hệ thống được thiết kế gồm ba thành phần chính: **Input (Đầu vào)**, **Emulator Engine (Bộ máy mô phỏng)** và **Output (Kết quả đầu ra)**, cùng với một mô-đun phụ trợ **Visualization Module (Mô-đun trực quan hóa)**.

4.2.1 Input (Đầu vào)

Phần Input cung cấp các thông tin cấu hình cần thiết cho quá trình mô phỏng:

- **Scheduling Algorithms:** Chọn loại giải thuật lập lịch (ILP, AI, heuristic) nhằm định hướng cấu hình hệ thống:
 - *ILP*: Sau bước lập lịch, hệ thống sẽ thêm mô-đun xử lý kết quả từ solver ILP.

- *AI-based*: Hệ thống bổ sung chức năng đọc kết quả từ mô-đun huấn luyện AI.
- *Heuristic*: Hệ thống thực hiện lập lịch dựa trên quy tắc heuristic cài sẵn.

- **Quantum Device Backends**: Lựa chọn danh sách các backend phần cứng lượng tử sẽ sử dụng trong mô phỏng. Cụ thể, hệ thống hỗ trợ 58 backend phổ biến, bao gồm:

Bảng 4.1: Danh sách Quantum Backend theo số qubit

1 Qubit	5 Qubit	7 Qubit	15 Qubit	16 Qubit	20 Qubit	27 Qubit	28 Qubit
FakeArmonkV2	FakeAthensV2 FakeBelemV2 FakeBogotaV2 FakeBurlingtonV2 FakeEssexV2 FakeFractionalBackend FakeLimaV2 FakeLondonV2 FakeManilaV2 FakeOurenseV2 FakeQuitoV2 FakeRomeV2 FakeSantiagoV2 FakeValenciaV2 FakeVigoV2 FakeYorktownV2	FakeCasablancaV2 FakeJakartaV2 FakeLagosV2 FakeNairobiV2 FakeOslo FakePerth	FakeMelbourneV2	FakeGuadalupeV2	FakeAlmadenV2 FakeBoeblingenV2 FakeJohannesburgV2 FakePoughkeepsieV2 FakeSingaporeV2	FakeAlgiers FakeAuckland FakeCairoV2 FakeGeneva FakeHanoiV2 FakeKolkataV2 FakeMontrealV2 FakeMumbaiV2 FakeParisV2 FakePeekskill FakeSydneyV2 FakeTorontoV2	FakeCambridgeV2

33 Qubit	53 Qubit	65 Qubit	127 Qubit	133 Qubit	156 Qubit
FakePrague	FakeRochesterV2	FakeBrooklynV2 FakeManhattanV2	FakeBrisbane FakeCusco FakeKawasaki FakeKyiv FakeKyoto FakeOsaka FakeQuebec FakeSherbrooke FakeWashingtonV2	FakeTorino	FakeMarrakesh

- **Batch of Quantum Circuits as Submitted Jobs**: Danh sách các mạch lượng tử được nạp lên hệ thống, chứa thông tin: số lượng mạch, số qubit yêu cầu, thời gian nộp, runtime ước tính, v.v. Các mạch lượng tử này được lấy từ **MQT Bench** [48], một bộ công cụ benchmark phổ biến trong lĩnh vực lượng tử.

MQT Bench hỗ trợ nhiều loại bài toán benchmark khác nhau, bao gồm: Bảng 4.2 mô tả các benchmark.

Bảng 4.2: Ánh xạ benchmark_name và tên đầy đủ các benchmark

benchmark_name	Benchmark	benchmark_name	Benchmark
ae	Amplitude Estimation	qwalk-noancilla	Quantum Walk (no ancilla)
dj	Deutsch-Jozsa	qwalk-v-chain	Quantum Walk (v-chain)
ghz	GHZ State	random	Random Circuit
graphstate	Graph State	realamprandom	Real Amplitudes (Random)
groundstate	Ground State	su2random	Efficient SU2 (Random)
grover-noancilla	Grover's (no ancilla)	twolocalrandom	Two Local (Random)
grover-v-chain	Grover's (v-chain)	vqe	Variational Quantum Eigensolver
portfolioqaoa	Portfolio Optimization (QAOA)	wstate	W-State
portfoliovqe	Portfolio Optimization (VQE)	shor	Shor's Algorithm
pricingcall	Pricing Call Option	qft	Quantum Fourier Transformation
pricingput	Pricing Put Option	qftentangled	QFT Entangled
qnn	Quantum Neural Network	qpeexact	Quantum Phase Estimation (Exact)
qpeinexact	Quantum Phase Estimation (Inexact)	routing	Routing
tsp	Travelling Salesman Problem		

Mỗi benchmark đại diện cho một loại mạch lượng tử đặc thù, phục vụ đánh giá hiệu suất hệ thống trên nhiều loại bài toán lượng tử khác nhau. Việc hỗ trợ đa dạng benchmark giúp đảm bảo tính tổng quát và khả năng mở rộng của hệ thống mô phỏng.

4.2.2 Emulator Engine (Bộ máy mô phỏng)

Đây là thành phần cốt lõi, thực hiện toàn bộ quy trình lập lịch và mô phỏng:

Main Components:

- **Circuit Cutter:** Thực hiện việc chia nhỏ mạch lượng tử thành các *subcircuit* sao cho mỗi subcircuit không vượt quá giới hạn qubit của từng thiết bị. Thành phần này hỗ trợ nhiều giải thuật cắt, bao gồm:
 - **Greedy Cut:** Giải thuật tham lam cắt các mạch lượng tử lớn theo kích thước lớn nhất có thể chạy trên các máy lượng tử có trong hệ thống

Algorithm 1: Greedy Cut Algorithm

Input: Quantum circuit C , qubit limit max_qubits **Output:** List of subcircuits

```
1 subcircuits  $\leftarrow$  empty list;
2 while  $size(C) > max\_qubits$  do
3    $S \leftarrow$  find largest independent subgraph of  $C$  such that  $size(S) \leq$ 
      $max\_qubits$ ;
4   append  $S$  to subcircuits;
5   remove  $S$  from  $C$ ;
6 if  $size(C) > 0$  then
7   append  $C$  to subcircuits;
8 return subcircuits;
```

- **Half Cut:** Giải thuật chia đôi mạch lượng tử tại điểm giữa, đảm bảo kích thước mạch con nhỏ hơn hoặc bằng một nửa mạch gốc.

Algorithm 2: Half Cut Algorithm

Input: Quantum circuit C **Output:** Two subcircuits C_1, C_2

```
1 if  $size(C) > 1$  then
2    $cut\_point \leftarrow \text{floor}(size(C) / 2)$ ;
3    $C_1 \leftarrow$  gates from index 0 to  $cut\_point - 1$ ;
4    $C_2 \leftarrow$  gates from index  $cut\_point$  to end of  $C$ ;
5 else
6    $C_1 \leftarrow C$ ;
7    $C_2 \leftarrow$  empty circuit;
8 return  $C_1, C_2$ ;
```

Các giải thuật này cung cấp cách tiếp cận linh hoạt, cho phép lựa chọn chiến lược cắt phù hợp theo từng tình huống hoặc yêu cầu tối ưu hoá.

- **Scheduler:** Thành phần chịu trách nhiệm thực hiện lập lịch các tác vụ lượng tử lên các thiết bị, dựa trên giải thuật lập lịch được cấu hình. Scheduler hỗ trợ nhiều loại giải thuật khác nhau, bao gồm:

- **ILP**: Lập lịch dựa trên phương pháp tối ưu số nguyên tuyến tính.
- **AI**: Sử dụng các phương pháp liên quan đến AI như Reinforcement Learning (RL), v.v.
- **Heuristic**: Dùng các nguyên tắc để phân bổ tài nguyên để thực hiện phân bổ các mạch vào các máy trong hệ thống.

Mỗi giải thuật có thể yêu cầu các thông tin đầu vào (*input*) khác nhau để hoạt động, như: ma trận nhiễu, độ sâu mạch, bản đồ kiến trúc qubit, thông tin hàng đợi thiết bị, v.v.

Tuy nhiên, output của Scheduler luôn tuân theo một cấu trúc thống nhất dưới dạng frame JSON, đảm bảo dễ dàng phân tích và so sánh các giải thuật.

Mỗi job được lập lịch sẽ có thông tin:

- **job**: tên định danh tác vụ
- **qubits**: số qubit yêu cầu
- **machine**: tên thiết bị được lập lịch
- **capacity**: số qubit khả dụng trên thiết bị
- **start**: thời điểm bắt đầu thực thi
- **end**: thời điểm kết thúc
- **duration**: thời lượng thực thi

Định dạng này hỗ trợ đảm bảo tính thống nhất, thuận tiện cho các bước xử lý tiếp theo như: mô phỏng thực thi, đánh giá hiệu suất, trực quan hóa biểu đồ Gantt, và lưu trữ hồ sơ lịch sử lập lịch.

- **Estimator**: Thành phần chịu trách nhiệm ước lượng các thông số hiệu suất quan trọng, hỗ trợ quá trình thu thập thông tin phục vụ lập lịch.

Estimator thu thập các thông số đầu vào tùy theo yêu cầu của giải thuật lập lịch. Hệ thống hiện hỗ trợ hai phương thức chính:

- **Width, Depth**: Được trích xuất trực tiếp từ thông tin đặc tả của các mạch lượng tử (ví dụ: thông qua metadata của benchmark hoặc từ thư viện mô tả mạch lượng tử).

- **Fidelity:** Được truy xuất từ thư viện `mapomatic`[49], một công cụ hỗ trợ đánh giá và chấm điểm độ nhiễu của phần cứng lượng tử, cung cấp chỉ số độ trung thực dựa trên đặc tính thiết bị và ánh xạ qubit.

Estimator đóng vai trò quan trọng trong việc cung cấp dữ liệu dự báo, giúp Scheduler lựa chọn thời điểm và thiết bị tối ưu cho từng tác vụ, đặc biệt trong môi trường nhiều thiết bị lượng tử với mức độ nhiễu không đồng nhất.

Multi-device Quantum Execution Simulator: Đây là thành phần chịu trách nhiệm mô phỏng việc thực thi các tác vụ lượng tử trên nhiều thiết bị lượng tử (quantum backends) đồng thời, phản ánh quá trình triển khai thực tế trong môi trường có nhiều thiết bị khác nhau với các giới hạn về tài nguyên, độ nhiễu và kết nối qubit. Thành phần này bao gồm ba bước xử lý chính:

- **Transpile Circuit:** Chuyển đổi mạch lượng tử từ dạng logic trừu tượng ban đầu thành một mạch tương thích với tập lệnh (ISA) của phần cứng cụ thể. Quá trình này bao gồm:
 - Thay thế các cổng logic trừu tượng bằng các cổng vật lý khả thi trên backend.
 - Chèn các cổng SWAP cần thiết để đảm bảo ràng buộc kết nối qubit.
 - Giảm thiểu độ sâu (depth) của mạch thông qua tối ưu hóa.

Mục tiêu của bước này là đảm bảo mạch có thể thực thi được trên backend lựa chọn mà vẫn giữ được độ chính xác cao nhất có thể.

- **Mapping Circuit:** Ánh xạ các qubit logic trong mạch lên các qubit vật lý trên thiết bị lượng tử cụ thể, dựa trên kết quả transpile circuit.
- **Simulation Density:** Sau khi transpile và mapping, mạch được mô phỏng để đánh giá hiệu năng thực thi. Bước này bao gồm:
 - Tính toán độ chính xác (fidelity) thực tế đạt được sau khi thực thi mạch trên backend.
 - Đo lường thời gian thực thi dự kiến (estimated execution time).
 - Kiểm tra mức độ sử dụng tài nguyên (device utilization).

Kết quả từ Simulation Density sẽ được tổng hợp vào **Quantum Execution Performance Metric** để phục vụ phân tích, so sánh giữa các giải thuật lập lịch và backend.

Hệ thống mô phỏng nhiều thiết bị lượng tử trong **Multi-device Quantum Execution Simulator** cho phép kiểm tra các chiến lược lập lịch trong bối cảnh thực tế, nơi các backend có đặc điểm và hạn chế khác nhau, từ đó hỗ trợ các nghiên cứu tối ưu hóa hiệu quả sử dụng hạ tầng lượng tử đa thiết bị.

Quantum Execution Performance Metric: Tính toán các chỉ số hiệu suất: fidelity, makespan, utilization, phục vụ so sánh giải thuật lập lịch (tham chiếu chi tiết trong chương 6).

4.2.3 Output (Kết quả đầu ra)

Kết quả mô phỏng được lưu trữ dưới dạng tệp JSON, đảm bảo tính dễ đọc, dễ xử lý bằng các công cụ lập trình và trực quan hóa. Các kết quả chính bao gồm:

- **Scheduler Result:** Lưu trữ kết quả lập lịch các tác vụ lượng tử theo định dạng khung thời gian chuẩn. Mỗi tác vụ (job) được biểu diễn với các thuộc tính: tên job, số qubit sử dụng, máy được gán, dung lượng qubit máy, thời gian bắt đầu, thời gian kết thúc và thời lượng. Định dạng này đảm bảo tính thống nhất cho mọi giải thuật lập lịch, hỗ trợ so sánh trực tiếp.
- **Simulation Result:** Kết quả mô phỏng của hệ thống được lưu trữ dưới dạng JSON, ghi lại thông tin thời gian bắt đầu, kết thúc, duration, số qubit, máy thực thi... cho từng tác vụ.

Các kết quả này được sinh ra từ hai cơ chế mô phỏng: cho phép multithreading và không cho phép multithreading.

Giải thuật mô phỏng luồng song song được diễn tả như sau:

Algorithm 3: Simulate Scheduling with Multithreading

Input: List of jobs $jobs$

Output: List of jobs with assigned start and end times

```
1 machine_schedules  $\leftarrow$  {each machine: []};
2 jobs  $\leftarrow$  sort in ascending order by  $start$ ;
3 foreach  $job$  in  $jobs$  do
4     machine  $\leftarrow$  job.machine;
5     current_schedule  $\leftarrow$  machine_schedules[machine];
6     start_time  $\leftarrow$  job.start;
7     while not enough qubit capacity do
8         active_jobs  $\leftarrow$  jobs in  $current\_schedule$  with  $end > start\_time$ ;
9         total_qubits  $\leftarrow$  sum of  $qubits$  in  $active\_jobs$ ;
10        if  $total\_qubits + job.qubits \leq job.capacity$  then
11            break;
12        start_time  $\leftarrow$  min( $j.end$  for  $j$  in  $active\_jobs$ );
13    job.start  $\leftarrow$  start_time;
14    job.end  $\leftarrow$  start_time + duration(job);
15    append  $job$  to  $current\_schedule$ ;
16 return  $jobs$ ;
```

Giải thuật mô phỏng không cho phép multithreading (thực hiện từng job một trên mỗi máy) được mô tả như sau:

Algorithm 4: Simulate Scheduling without Multithreading

Input: List of jobs $jobs$

Output: Scheduled jobs per machine

```
1 Initialize  $machine\_current$ ,  $ready\_queue$  for each machine;
2 foreach  $job \in jobs$  do
3    $job.duration \leftarrow get\_duration\_from\_transpiled(job)$ ;
4    $job.end \leftarrow job.start + job.duration$ ;
5  $jobs\_per\_machine \leftarrow group\ jobs\ by\ machine$ ;
6 foreach  $machine \in jobs\_per\_machine$  do
7    $current\_time \leftarrow 0$ ,  $capacity \leftarrow machine.capacity$ ;
8    $job\_list \leftarrow jobs\_per\_machine[machine]$ ;
9   while  $job\_list \neq \emptyset$  do
10    foreach  $job \in job\_list$  do
11      if  $job.start \leq current\_time$  and  $job.qubits \leq capacity$  then
12        move  $job$  to  $ready\_queue[machine]$ ,  $capacity - = job.qubits$ ;
13      if  $ready\_queue[machine] \neq \emptyset$  then
14         $max\_end \leftarrow current\_time$ ;
15        foreach  $job \in ready\_queue[machine]$  do
16           $job.start \leftarrow current\_time$ ,  $job.end \leftarrow job.start + job.duration$ ;
17           $max\_end \leftarrow \max(max\_end, job.end)$ ;
18           $capacity + = job.qubits$ , move  $job$  to
19             $machine\_current[machine]$ ;
20          remove  $job$  from  $job\_list$ ;
21         $ready\_queue[machine] \leftarrow \emptyset$ ,  $current\_time \leftarrow max\_end$ ;
22      else
23         $current\_time + = 1$ ;
24 return  $machine\_current$ ;
```

- **Job-Scheduled Log:** Thông tin log được khôi phục bằng cách tính toán lại thời gian bắt đầu, kết thúc, độ dài và độ trung thực tổng hợp của job gốc dựa trên các subcircuit con.

Mỗi job gốc có thể chứa nhiều subcircuit con. Quá trình tính toán log thực hiện như sau:

1. Xác định `start_time` là thời gian bắt đầu nhỏ nhất của các subcircuit con.
2. Xác định `end_time` là thời gian kết thúc lớn nhất của các subcircuit con.
3. Tính `duration = end_time - start_time`.
4. Tính fidelity tổng hợp dựa trên trung bình có trọng số theo số qubit của từng subcircuit.

Algorithm 5: Recover Parent Job Log from Subcircuit Logs

Input: `origin_job_info`: list of original jobs, each job may contain `childrenJobs`

Output: Updated `start_time`, `end_time`, `duration`, `fidelity` for each original job

```

1 foreach job_name, job_info in origin_job_info do
2   if job_info.childrenJobs  $\neq$  None then
3     count_fidelity  $\leftarrow$  0;
4     foreach child_job in job_info.childrenJobs do
5       job_info.start_time  $\leftarrow$ 
6         min(job_info.start_time, child_job.start_time);
7       job_info.end_time  $\leftarrow$ 
8         max(job_info.end_time, child_job.end_time);
9       job_info.duration  $\leftarrow$  job_info.end_time - job_info.start_time;
10      count_fidelity  $\leftarrow$  count_fidelity + (child_job.fidelity  $\times$ 
11        child_job.qubits);
12    job_info.fidelity  $\leftarrow$  count_fidelity / job_info.qubits;
13  else
14    Print: "Job job_name has no children jobs.";

```

- **Profiled Data:** Dữ liệu chi tiết được thu thập trong quá trình mô phỏng, chứa các thông tin profiling của từng bước xử lý. Dữ liệu này được lưu trữ dưới dạng file JSON có cấu trúc như sau:

- `numcircuit`: Tổng số mạch lượng tử được mô phỏng.
- `nameAlgorithm`: Tên giải thuật lượng tử được sử dụng (ví dụ: `ghz`).

- `averageQubits`: Số lượng qubit trung bình của các mạch lượng tử.
- `nameSchedule`: Tên giải thuật lập lịch được sử dụng (ví dụ: MTMC).
- `typeMachine`: Thông tin số qubit tối đa của từng thiết bị lượng tử tham gia mô phỏng. Đây là một dictionary ánh xạ tên thiết bị với số qubit khả dụng (ví dụ: `fake_belem`: 5).
- `average_turnaroundTime`: Thời gian hoàn tất trung bình của các job (tính từ lúc nộp đến khi hoàn tất).
- `average_responseTime`: Thời gian phản hồi trung bình (tính từ lúc nộp đến khi bắt đầu thực thi).
- `average_fidelity`: Độ trung thực trung bình của các job sau khi thực thi.
- `sampling_overhead`: Chi phí (tính theo thời gian) của bước sampling circuit.
- `average_throughput`: Số lượng job hoàn thành trung bình trên một đơn vị thời gian.
- `average_utilization`: Mức độ sử dụng tài nguyên trung bình trên toàn bộ hệ thống.
- `scheduling_latency`: Thời gian cần thiết để giải thuật trả về kết quả (tính bằng giây).
- `makespan`: Tổng thời gian thực hiện toàn bộ tập job.

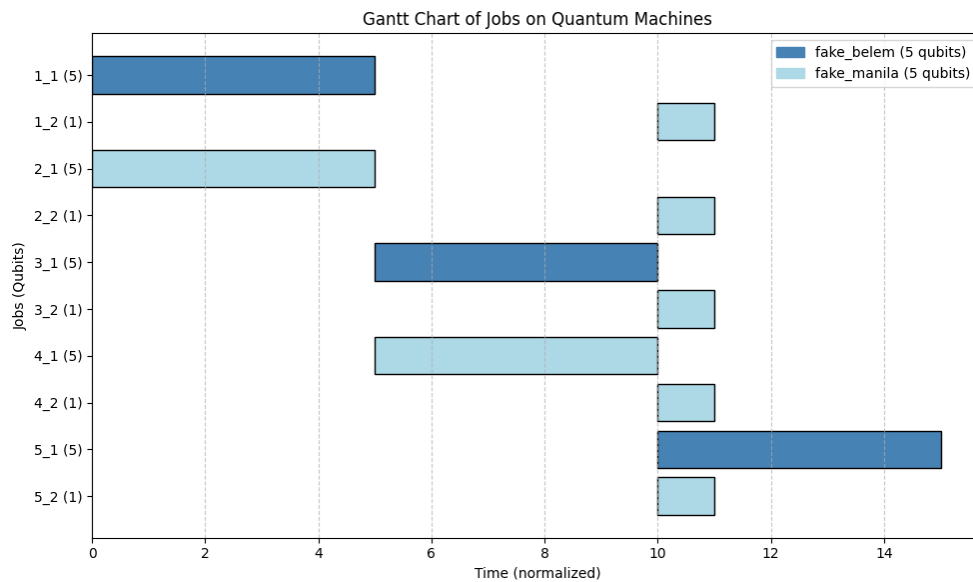
Dữ liệu `Profiled Data` này đóng vai trò quan trọng trong việc phân tích chuyên sâu hiệu quả của các giải thuật lập lịch, giúp đánh giá đa chiều về thời gian, tài nguyên, độ tin cậy và khả năng mở rộng của hệ thống. Ngoài ra, dữ liệu này còn hỗ trợ trực quan hóa nâng cao và phục vụ mục tiêu tối ưu hóa cho các nghiên cứu tương lai.

Việc xuất dữ liệu theo định dạng JSON cũng cho phép dễ dàng sử dụng trong các workflow phân tích tự động.

4.2.4 Visualization Module (Mô-đun trực quan hóa)

Mô-đun này chịu trách nhiệm trực quan hóa kết quả mô phỏng, giúp người dùng dễ dàng phân tích, đánh giá và so sánh hiệu suất của các giải thuật lập lịch. Các chức năng chính của mô-đun bao gồm:

- **Gantt Chart:** Biểu diễn trực quan lịch sử thực thi các tác vụ trên từng thiết bị lượng tử. Biểu đồ Gantt thể hiện mối quan hệ thời gian giữa các job, thời gian bắt đầu, thời gian kết thúc và mức độ song song (parallelism) trong quá trình thực thi, giúp người dùng quan sát rõ ràng các giai đoạn thực hiện.



Hình 4.3: Ví dụ minh họa về Gantt Chart.

- **Analysis Statistics:** Hiển thị các thống kê phân tích dưới dạng bảng dữ liệu hoặc các biểu đồ (ví dụ: histogram, bar chart, line chart). Các thống kê bao gồm: trung bình thời gian hoàn thành (average turnaround time), độ trung thực trung bình (average fidelity), throughput, utilization, makespan, v.v. Các biểu đồ này được sinh tự động từ dữ liệu Profiled Data.

Mô-đun trực quan hóa được xây dựng dựa trên thư viện **Matplotlib**, tạo ra các biểu đồ dưới dạng ảnh tĩnh, đồng thời hỗ trợ xuất kết quả ra các định dạng phổ biến như .png, .pdf. Việc lựa chọn định dạng đầu ra tùy thuộc vào nhu cầu của người dùng, giúp thuận tiện cho quá trình báo cáo, lưu trữ hoặc trình bày kết quả.

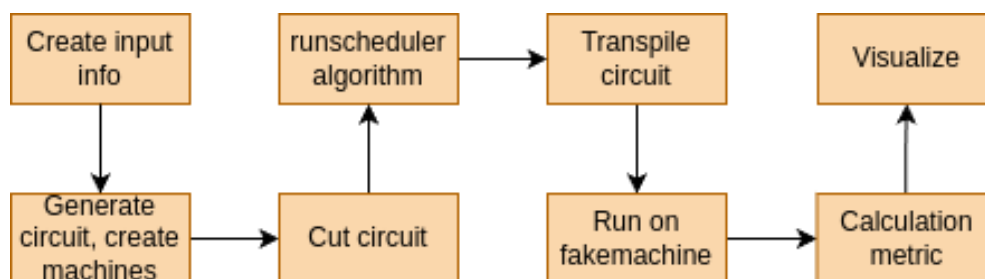
Hệ thống **QTSE (Quantum Task Scheduling Emulator)** cung cấp một môi trường mô phỏng toàn diện cho bài toán lập lịch tác vụ lượng tử trên nhiều thiết bị. Nhờ sự tích hợp giữa các module: *Input*, *Emulator Engine*, *Output* và *Visualization Module*, hệ thống cho phép đánh giá hiệu quả của nhiều giải thuật lập lịch khác nhau thông qua nhiều tiêu chí đánh giá: thời gian phản hồi, độ trung thực, makespan, throughput, utilization, v.v. Thiết kế linh hoạt và khả năng mở rộng của hệ thống tạo tiền đề thuận lợi cho việc thử nghiệm các giải thuật mới hoặc áp dụng trên các kiến trúc phần cứng lượng tử khác nhau trong tương lai.

4.3 Workflow hệ thống

Việc xây dựng workflow hệ thống không chỉ giúp đảm bảo tính tái lập và kiểm thử được cho các thực nghiệm, mà còn đóng vai trò như một khung triển khai thống nhất để tích hợp và đánh giá các giải thuật lập lịch khác nhau. Hệ thống này bao gồm các bước từ thiết lập đầu vào, xử lý mạch lượng tử, thực thi giải thuật lập lịch, cho đến mô phỏng kết quả, tính toán các chỉ số đánh giá và trực quan hoá đầu ra. Mỗi bước được xây dựng thành một khối chức năng độc lập, cho phép dễ dàng thay đổi thuật toán, cấu hình phần cứng, hoặc chiến lược xử lý mạch mà không ảnh hưởng đến các bước khác.

Mô tả sơ bộ

Hệ thống mô phỏng được triển khai theo quy trình xử lý tuần tự như minh họa ở Hình 4.4. Quy trình bao gồm nhiều bước từ thiết lập dữ liệu đầu vào, thực thi giải thuật lập lịch, đến tính toán và hiển thị kết quả. Mỗi bước là một khối xử lý độc lập giúp cho việc linh hoạt thay đổi và phát triển hệ thống sau này.



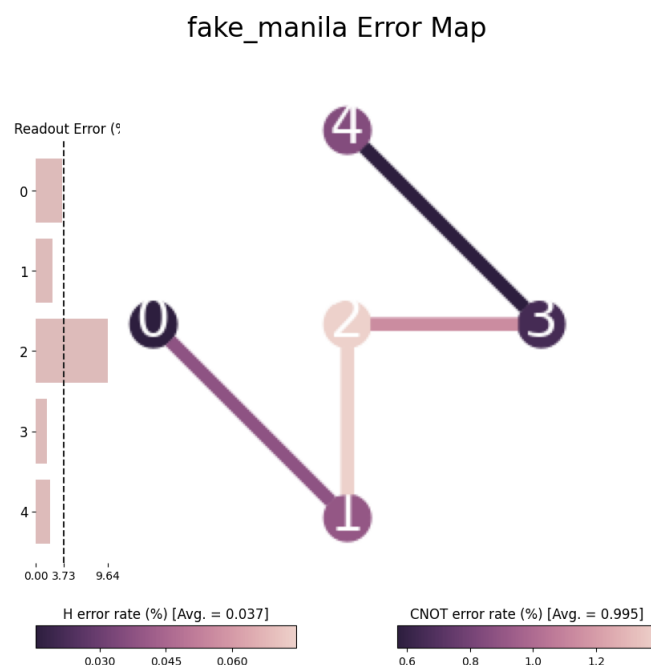
Hình 4.4: Workflow của hệ thống.

Quy trình xử lý này không chỉ được mô tả dưới dạng sơ đồ mà còn được triển khai

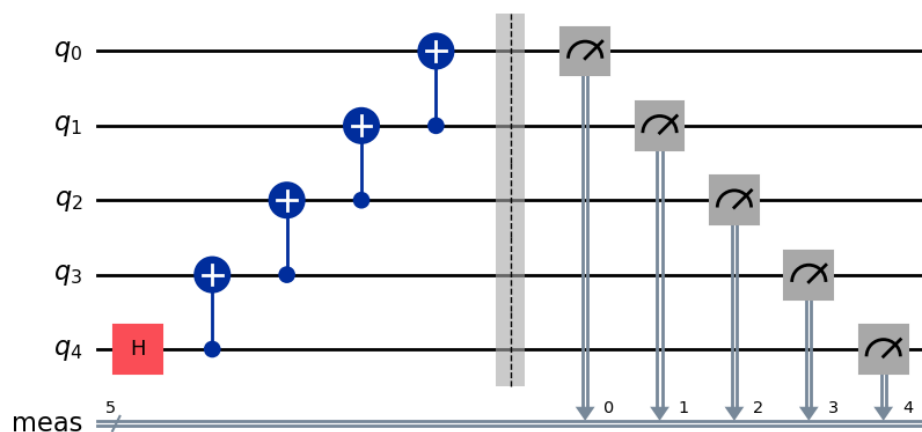
thành một luồng xử lý đầy đủ trong hệ thống, đảm bảo tính khả thi, khả năng tái lập và dễ dàng mở rộng.

Giải thích các thành phần

- **Create input info:**
 - **Input:** Tập cấu hình JSON chứa thông tin giải thuật lập lịch, danh sách mạch benchmark (dạng tên benchmark MQT-Bench), danh sách thiết bị lượng tử (tên backend, số qubit, topology, gate error,...).
 - **Output:** Một đối tượng cấu hình trong chương trình, sẵn sàng truyền sang bước tiếp theo.
- **Generate circuit, create machines:**
 - **Input:** Danh sách benchmark (ví dụ: "ghz", "qft",...), thông tin backend.
 - **Output:** Một danh sách các đối tượng circuit (mạch lượng tử) và danh sách các fake machine (mô tả khả năng thiết bị lượng tử: số qubit, error rate...).



Hình 4.5: Kiến trúc backend (manila).

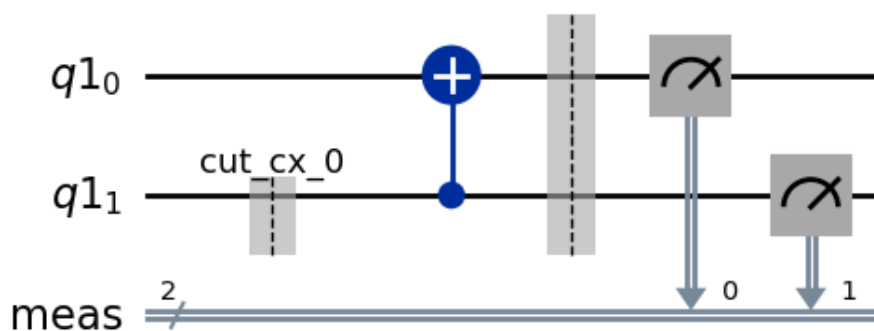


Hình 4.6: Circuit ghz.

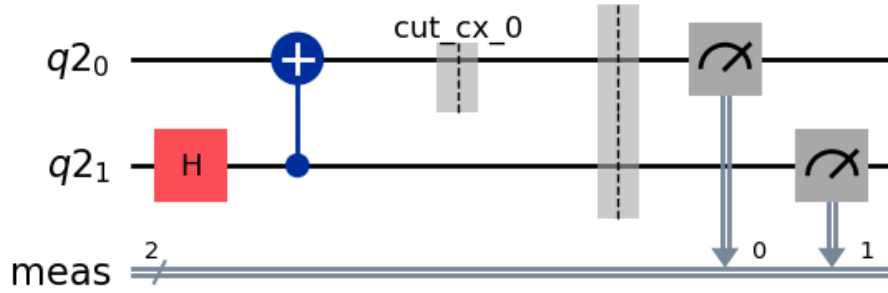
• **Cut circuit:**

- **Input:** Mạch lượng tử ban đầu, thông tin về số qubit tối đa mỗi backend.
- **Output:** Một tập các subcircuit thỏa mãn điều kiện kích thước qubit, lưu kèm metadata (thuộc về circuit nào, vị trí cắt,...).

Áp dụng giải thuật chia mạch như *Greedy Cut*, *Half Cut* để cắt mạch lớn thành các subcircuit phù hợp.



Hình 4.7: Mạch con 2 của ghz theo half cut.



Hình 4.8: Mạch con 2 của ghz theo half cut.

• **Run scheduler algorithm:**

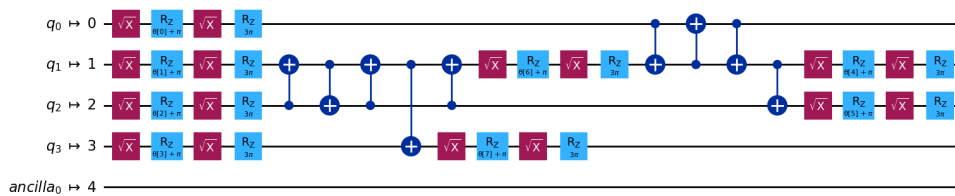
- **Input:** Danh sách subcircuit, danh sách thiết bị, giải thuật lập lịch đã chọn (ví dụ: FCFS, MILP, ILP, RL...).
- **Output:** Danh sách job đã lập lịch, mỗi job chứa thông tin: tên job, qubit, thiết bị gán, thời gian bắt đầu, thời gian kết thúc, duration.

Output được chuẩn hoá ở dạng JSON như sau:

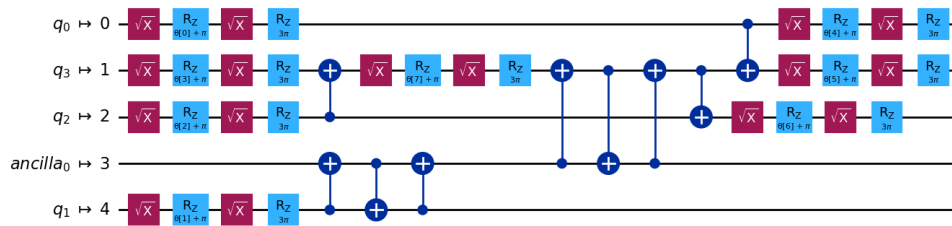
```
1  {"job": "1_1", "qubits": 5, "machine": "fake_belem",
    "start": 0.0, "end": 5.0, "duration": 5}
```

• **Transpile circuit:**

- **Input:** Danh sách job đã lập lịch (tên job, circuit tương ứng, backend tương ứng).
- **Output:** Mạch đã transpile tương thích hardware. 4.9, 4.10 trên máy 5 qubits.



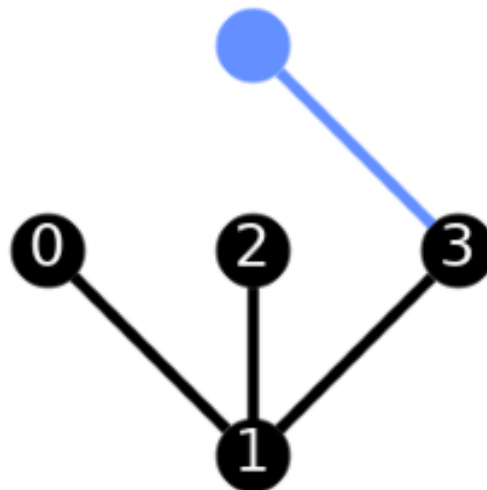
Hình 4.9: Transpile theo vị trí [0,1,2,3].



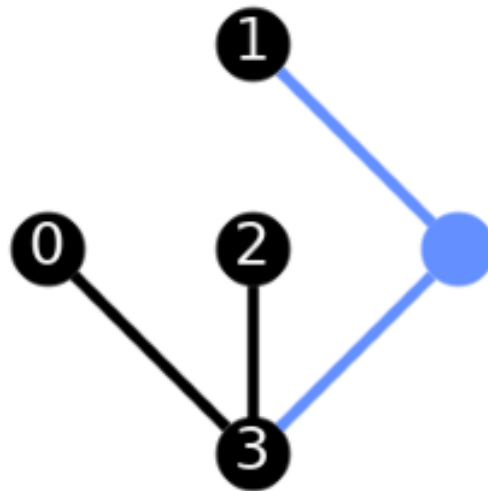
Hình 4.10: Transpile theo vị trí [0,1,2,4].

- **Run on fakemachine:**

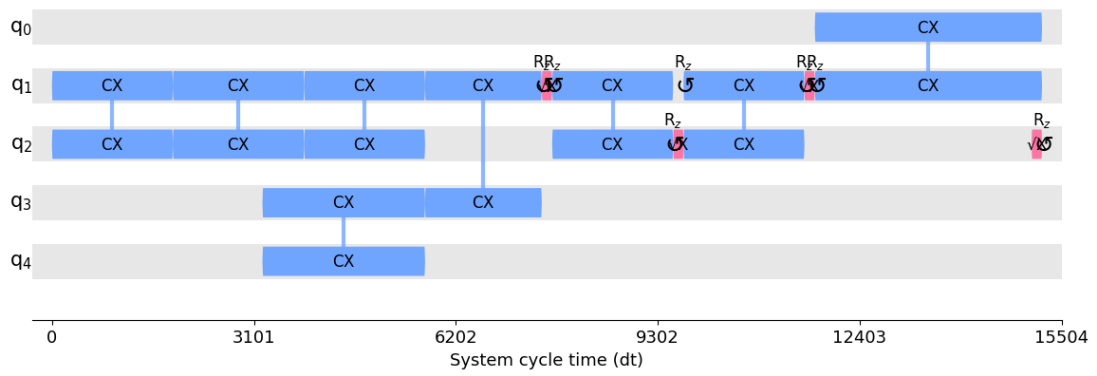
- **Input:** Danh sách circuit đã transpile, thông tin job (thời gian bắt đầu, backend, số qubit,...).
- **Output:** Kết quả mô phỏng từng job: thời gian thực thi thực tế, fidelity, lỗi đo, kết quả measurement.



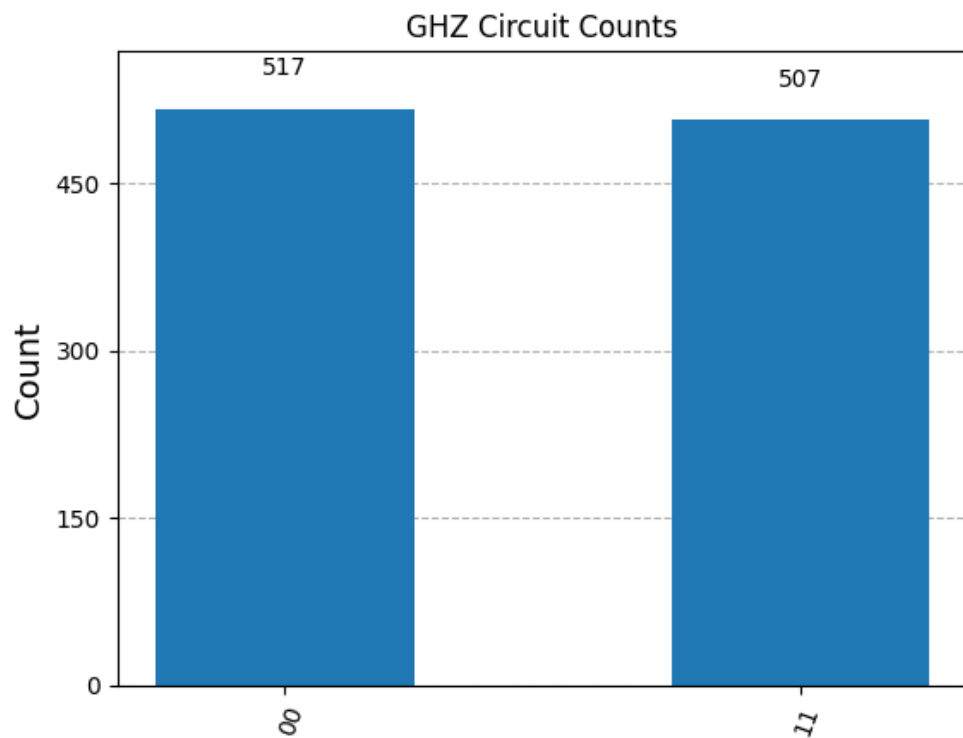
Hình 4.11: Map mạch lên máy theo vị trí [0,1,2,3].



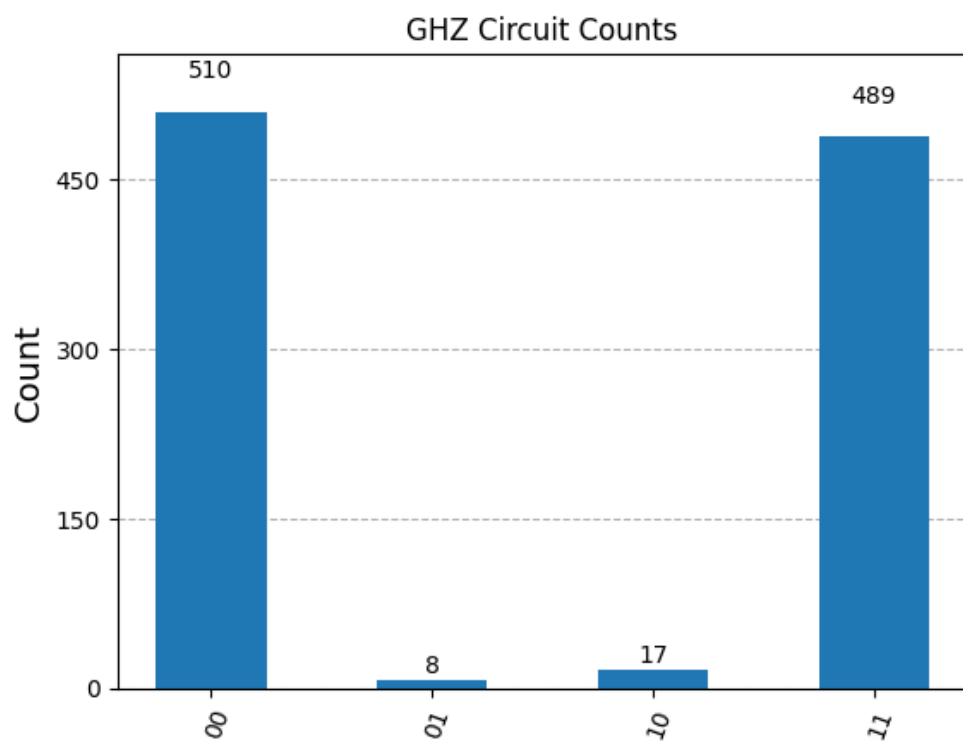
Hình 4.12: Map mạch lên máy theo vị trí [0,1,2,4].



Hình 4.13: Thời gian chạy trên máy.



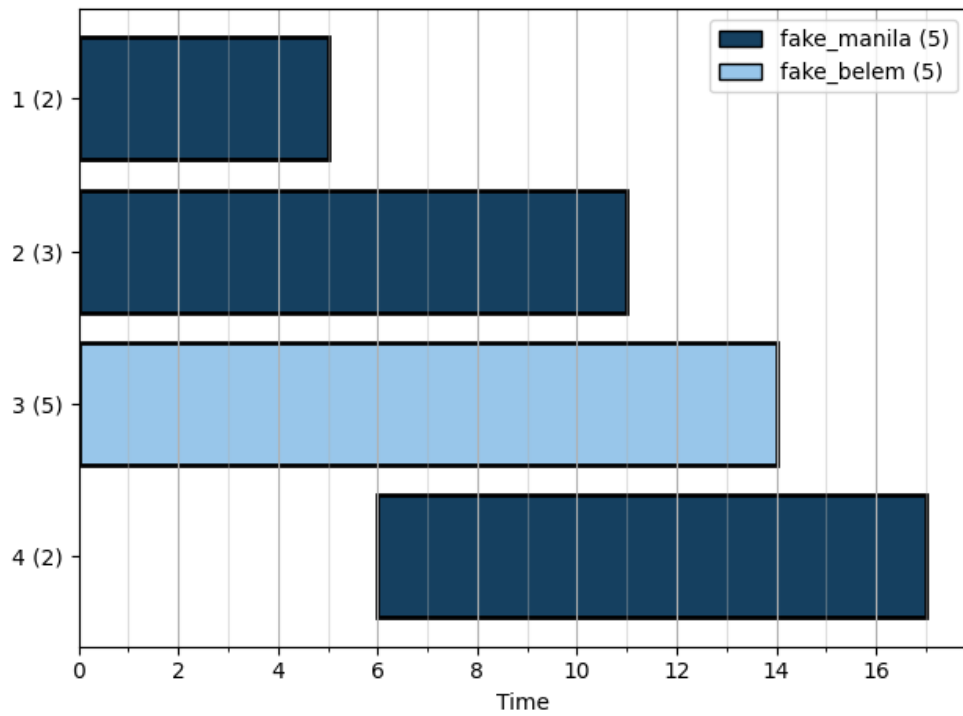
Hình 4.14: Kết quả chạy measurement không có nhiễu.



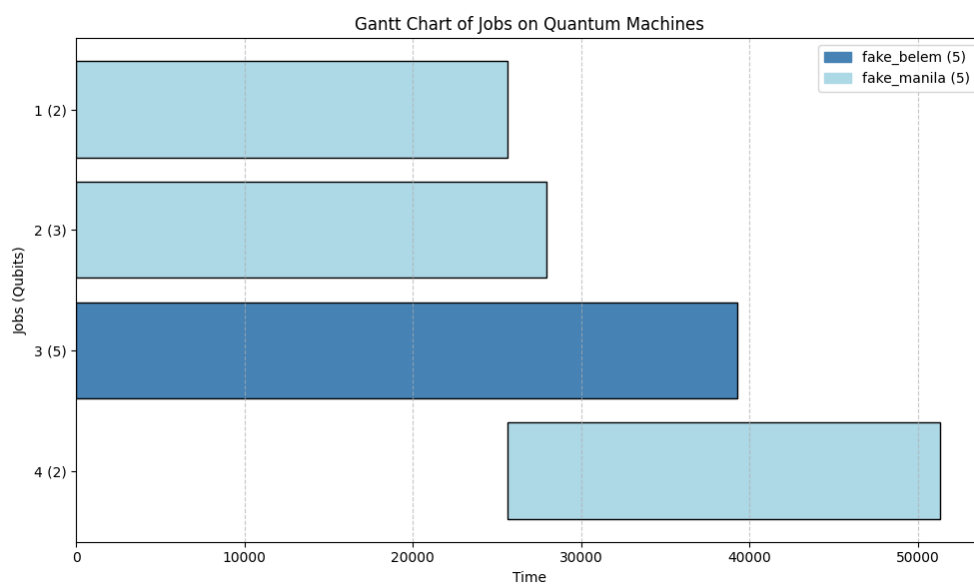
Hình 4.15: Kết quả chạy measurement có nhiễu.

Hai chế độ mô phỏng được hỗ trợ:

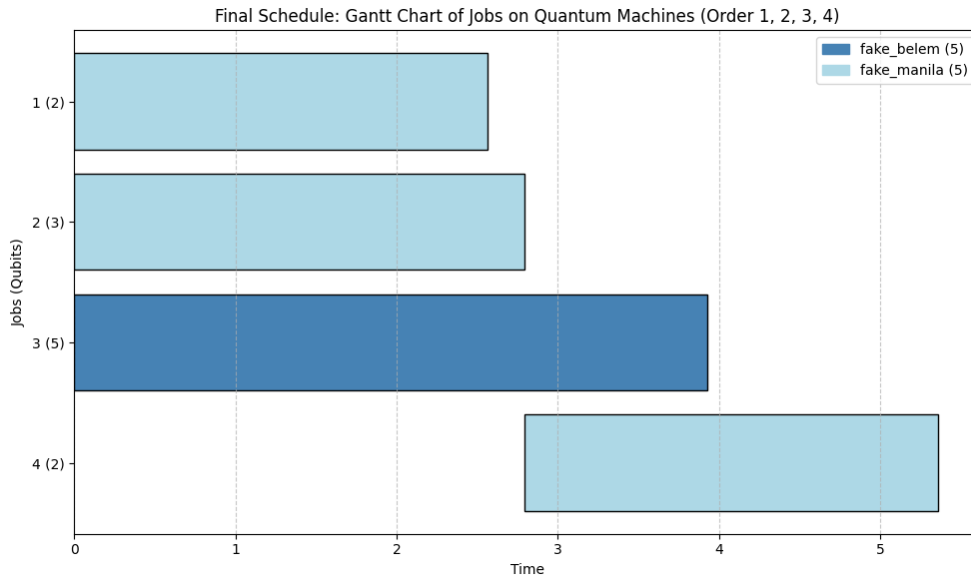
- **Cho phép multithreading:** job có thể chạy song song nếu tổng qubit chưa vượt khả năng backend. 4.17
- **Không cho phép multithreading:** job phải đợi tất cả job trên máy hoàn thành mới được nạp tiếp. 4.18



Hình 4.16: Kết quả scheduler ban đầu.



Hình 4.17: Hiệu chỉnh thực tế của scheduler cho phép multithreading.



Hình 4.18: Hiệu chỉnh thực tế của scheduler không cho phép multithreading.

Calculation metric:

- **Input:** Kết quả thực thi của tất cả job.
- **Output:** Các chỉ số tổng hợp: fidelity trung bình, makespan, utilization, latency, turnaround time, throughput các tiêu chí này sẽ được mô tả rõ ở chương 5 enditemize

Dữ liệu được lưu ở dạng JSON như sau:

```

1  {
2      "numcircuit": 9,
3      "nameAlgorithm": "ghz",
4      "averageQubits": 9.0,
5      "nameSchedule": "MTMC",
6      "average_fidelity": 0.86,
7      "average_turnaroundTime": 61066.67,
8      "average_utilization": 0.91,
9      "makespan": 157728.0
10 }
```

- **Visualize:**

- **Input:** Kết quả job-scheduled log và metric JSON hỗ trợ trực quan hoá và phân tích sau này.
- **Output:** Hình ảnh biểu đồ Gantt, biểu đồ thống kê (PNG/PDF), bảng số liệu. Và với mỗi giải thuật nhóm sẽ có biểu đồ chuẩn hóa của toàn tập dữ liệu. Biểu đồ có dạng radar với dữ liệu được chuẩn hóa nhằm trực quan hóa mức độ tốt – xấu của từng giải thuật trên các tiêu chí đánh giá. Biểu đồ này đồng nhất hướng tối ưu của các tiêu chí bằng cách đảo ngược các tiêu chí mà giá trị nhỏ hơn là tốt hơn, đảm bảo rằng tất cả các tiêu chí đều theo hướng càng lớn càng tốt. Nhờ đó, biểu đồ radar thể hiện rõ ràng độ bao phủ của từng giải thuật trên toàn bộ tập tiêu chí: vùng bao phủ càng rộng biểu thị hiệu quả tổng thể càng cao.

Cách tiếp cận này cho phép so sánh trực quan và toàn diện giữa các giải thuật, giúp dễ dàng nhận diện ưu, nhược điểm của từng phương pháp thông qua hình dạng và diện tích của đa giác trên biểu đồ. Ngoài ra, biểu đồ còn hỗ trợ phát hiện các tiêu chí mà một giải thuật có thể cần cải thiện, từ đó cung cấp định hướng cho việc tối ưu hóa trong các nghiên cứu tiếp theo.

4.4 Tóm tắt chương

Chương này đã trình bày kiến trúc tổng quan và mô hình chi tiết của hệ thống, bao gồm các thành phần chính: input, emulator engine, output và visualization module. Quy trình hoạt động của hệ thống cũng được mô tả, làm rõ luồng xử lý và mối quan hệ giữa các thành phần.

Những nội dung hiện thực này là cơ sở để áp dụng các giải thuật lập lịch và xác định tiêu chí đánh giá trong chương tiếp theo.

Chương 5

Tiêu chí đánh giá

Trong chương 5, các tiêu chí đánh giá sẽ được trình bày cũng như kết quả so sánh từ chạy thực nghiệm các giải thuật trên môi trường đánh giá đã được trình bày ở chương trước.

5.1 Các tiêu chí đánh giá

Để đánh giá hiệu quả và chất lượng của các giải thuật lập lịch trong môi trường tính toán lượng tử đa thiết bị, hệ thống sử dụng một tập hợp các tiêu chí phản ánh toàn diện cả về độ chính xác, hiệu suất xử lý, chi phí tính toán và mức độ tận dụng tài nguyên. Các tiêu chí này được lựa chọn nhằm đảm bảo sự cân bằng giữa tính đúng đắn của kết quả lượng tử và khả năng ứng dụng thực tế của giải thuật. Cụ thể:

- **Fidelity (Độ trung thực):** Đo lường mức độ gần đúng giữa trạng thái lượng tử thu được và trạng thái lý tưởng mong muốn. Đây là tiêu chí quan trọng hàng đầu nhằm đảm bảo độ tin cậy của kết quả lượng tử.
- **Throughput:** Phản ánh số lượng công việc (job) được xử lý trong một đơn vị thời gian, biểu thị hiệu suất tổng thể của hệ thống.
- **Response Time:** Đo khoảng thời gian từ khi một job được gửi vào hệ thống đến khi bắt đầu thực thi, phản ánh khả năng đáp ứng nhanh của giải thuật lập lịch.
- **Turnaround Time:** Tính tổng thời gian từ lúc nộp job đến khi hoàn thành, thể hiện trải nghiệm người dùng và mức độ hiệu quả xử lý.

- **Makespan:** Tổng thời gian cần thiết để hoàn thành toàn bộ các job, phản ánh khả năng tối ưu hóa phân bổ tài nguyên và lập lịch.
- **Scheduling latency:** Đo thời gian giải thuật cần để sinh ra lịch trình, đại diện cho chi phí tính toán của quá trình lập lịch.
- **Utilization:** Đánh giá mức độ sử dụng tài nguyên (qubit) trong suốt quá trình thực thi, phản ánh hiệu quả khai thác phần cứng lượng tử.
- **Overhead Sampling:** Đo lường chi phí phụ phát sinh từ quá trình lấy mẫu, đo đạc hoặc hiệu chỉnh trạng thái lượng tử, tác động đến hiệu suất chung của hệ thống.

Tổng hợp các tiêu chí trên cho phép đánh giá giải thuật không chỉ ở khía cạnh chính xác mà còn ở khả năng mở rộng, hiệu quả thực thi, và tính khả thi trong thực tế.

5.1.1 Độ trung thực

Fidelity trong tính toán lượng tử

Fidelity (độ trung thực) trong tính toán lượng tử là một chỉ số đánh giá mức độ giống nhau giữa hai trạng thái lượng tử, thường được sử dụng để đo lường chất lượng của phép tính hoặc phép đo lượng tử trong môi trường có nhiễu (noisy quantum system). Fidelity có giá trị từ 0 đến 1, với 1 biểu thị hai trạng thái hoàn toàn giống nhau, và 0 biểu thị hoàn toàn khác nhau [50].

Trong bối cảnh lập lịch và thực thi trên phần cứng lượng tử, fidelity được sử dụng để ước lượng xác suất thành công của một phép đo, hoặc khả năng bảo toàn thông tin sau khi mạch lượng tử được thực thi trên phần cứng thực.

Cách tính Fidelity [51] Giả sử hai trạng thái lượng tử thuần (pure states) $|\psi\rangle$ và $|\phi\rangle$, fidelity được định nghĩa bằng bình phương của giá trị tuyệt đối của tích vô hướng giữa hai vector trạng thái:

$$F(|\psi\rangle, |\phi\rangle) = |\langle\psi|\phi\rangle|^2 \quad (5.1)$$

Trong trường hợp tổng quát hơn, giữa một trạng thái hỗn hợp (mixed state) ρ và một trạng thái thuần $|\psi\rangle$, fidelity được tính theo:

$$F(\rho, |\psi\rangle) = \langle \psi | \rho | \psi \rangle \quad (5.2)$$

Nếu cả hai là trạng thái hỗn hợp ρ và σ :

$$F(\rho, \sigma) = \left(\text{Tr} \sqrt{\sqrt{\rho} \sigma \sqrt{\rho}} \right)^2 \quad (5.3)$$

Trong ứng dụng lập lịch lượng tử, fidelity của một job có thể được tính dựa trên trung bình có trọng số theo số qubit của các subcircuit, theo công thức:

$$\text{Average Fidelity} = \frac{\sum(\text{numQbit} \times \text{Fidelity})}{\text{Total num of qubit}} \quad (5.4)$$

Trong đó:

- **numQubit:** Số lượng qubit được sử dụng cho từng công việc.
- **Fidelity:** Độ chính xác của phép tính lượng tử với qubit tương ứng.
- **Total num of qubit:** Tổng số qubit được sử dụng trong toàn bộ công việc.

Công thức này đảm bảo mỗi subcircuit đóng góp fidelity tỷ lệ với số qubit nó chiếm, phản ánh mức ảnh hưởng của từng phần trong toàn bộ job.

Trong hệ thống lượng tử, các phép tính rất nhạy cảm với lỗi và nhiễu. Đảm bảo độ trung thực cao là cần thiết để kết quả tính toán chính xác và đáng tin cậy. Tiêu chí này giúp giải thuật lập lịch tránh sử dụng các tài nguyên kém chất lượng và nâng cao độ tin cậy của hệ thống.

Khi độ trung thực là một tiêu chí đánh giá, giải thuật lập lịch phải chú trọng đến việc phân bổ tài nguyên lượng tử sao cho đảm bảo chất lượng tính toán. Điều này có thể làm giảm thông lượng nhưng bù lại đảm bảo độ chính xác, từ đó tối ưu hóa toàn cục hiệu quả thực thi.

Giả sử có 3 công việc như sau:

Công việc	numQubit	Fidelity
A	4	0.92
B	2	0.85
C	3	0.88

Tính độ trung thực trung bình:

$$\begin{aligned}\text{Average Fidelity} &= \frac{(4 \times 0.92) + (2 \times 0.85) + (3 \times 0.88)}{4 + 2 + 3} \\ &= \frac{3.68 + 1.70 + 2.64}{9} = \frac{8.02}{9} \approx 0.891\end{aligned}$$

Algorithm 6: Compute Average Fidelity

Input: A list of jobs, each element contains numQbit and fidelity

Output: The average fidelity value

```

1 Function ComputeAverageFidelity(jobs):
2   total_weighted_fidelity  $\leftarrow$  0;
3   total_qubit  $\leftarrow$  0;
4   foreach job  $\in$  jobs do
5     total_weighted_fidelity  $\leftarrow$  total_weighted_fidelity + (job.numQbit  $\times$ 
6       job.fidelity);
7     total_qubit  $\leftarrow$  total_qubit + job.numQbit;
8   if total_qubit == 0 then
9     return 0 ;                                // Avoid division by zero
10  else
11    average_fidelity  $\leftarrow$  total_weighted_fidelity / total_qubit;
12    return average_fidelity;

```

5.1.2 Throughput

Thông lượng là chỉ số cho biết hệ thống có thể xử lý bao nhiêu công việc trong một đơn vị thời gian. Nó được tính bằng công thức:

$$\text{Throughput} = \frac{\text{NumJob}}{\text{Makespan}} \quad (5.5)$$

Trong đó:

- **NumJob:** Tổng số công việc hoàn thành.
- **Makespan:** Tổng thời gian từ khi bắt đầu công việc đầu tiên đến khi kết thúc công

việc cuối cùng.

Throughput là một thước đo hiệu quả cốt lõi của bất kỳ giải thuật lập lịch nào. Nó cho thấy hệ thống có thể xử lý khối lượng công việc lớn đến đâu trong thời gian ngắn, từ đó ảnh hưởng trực tiếp đến hiệu năng và khả năng mở rộng.

Một giải thuật tối ưu theo thông lượng sẽ tìm cách giảm thời gian chết và tăng số lượng công việc hoàn thành. Tuy nhiên, nếu chỉ tối ưu hóa thông lượng mà bỏ qua độ trung thực, có thể dẫn đến sai sót nghiêm trọng trong tính toán lượng tử. Do đó cần cân bằng giữa hiệu quả và độ tin cậy.

Giả sử có 10 công việc được hoàn thành trong 50 đơn vị thời gian:

$$\text{Throughput} = \frac{10}{50} = 0.2 \text{ jobs/unit time} \quad (5.6)$$

Input

- jobs: Danh sách các công việc đã hoàn thành.
- end_times: Danh sách thời điểm kết thúc của từng công việc.

Output

- throughput: Giá trị thông lượng tính được.

Algorithm 7: Compute Throughput

Input: A list of jobs and a corresponding list of end_times

Output: The throughput value

```
1 num_jobs ← length of jobs;
2 makespan ← maximum value in end_times;
3 if makespan == 0 then
4   | return 0; // Avoid division by zero
5 else
6   | throughput ← num_jobs / makespan;
7   | return throughput;
```

Giả sử có 10 công việc hoàn thành với thời điểm kết thúc lần lượt như sau:

$$\text{end_times} = \{5, 10, 12, 8, 11, 7, 13, 15, 9, 10\}$$

$$\text{NumJob} = 10, \quad \text{Makespan} = 15 \Rightarrow \text{Throughput} = \frac{10}{15} \approx 0.667$$

5.1.3 Response time

Thời gian phản hồi (Response Time) [52] là khoảng thời gian từ khi một công việc được gửi vào hệ thống đến khi nó bắt đầu được thực hiện. Đây là một chỉ số đánh giá khả năng phản ứng nhanh của hệ thống lập lịch. Trung bình thời gian phản hồi được tính theo công thức:

$$\text{Average Response Time} = \frac{\sum \text{StartTime}}{\text{NumJob}} \quad (5.7)$$

Trong nhiều hệ thống, đặc biệt là các hệ thống thời gian thực hoặc tương tác, việc phản hồi nhanh chóng là vô cùng quan trọng. Người dùng thường quan tâm đến tốc độ phản hồi hơn là thời gian hoàn thành toàn bộ. Do đó, tiêu chí này giúp cải thiện trải nghiệm người dùng và tính linh hoạt của hệ thống.

Giải thuật lập lịch khi tối ưu hóa theo thời gian phản hồi sẽ có xu hướng ưu tiên các công việc ngắn hoặc công việc đến sớm. Điều này giúp cải thiện cảm nhận về hiệu suất hệ thống nhưng có thể dẫn đến giảm hiệu quả tổng thể nếu không kết hợp các tiêu chí khác.

Giả sử có 3 công việc có thời điểm bắt đầu thực thi lần lượt là: 2, 4, 6. Khi đó:

$$\text{Average Response Time} = \frac{2 + 4 + 6}{3} = \frac{12}{3} = 4 \quad (5.8)$$

Thời gian phản hồi trung bình là khoảng thời gian trung bình từ khi công việc đến cho đến khi nó bắt đầu được thực thi.

$$\text{Average Response Time} = \frac{\sum \text{StartTime}}{\text{NumJob}}$$

Input

- `start_times`: Danh sách thời điểm bắt đầu thực thi của từng công việc.

Output

- `avg_response_time`: Giá trị thời gian phản hồi trung bình.

Algorithm 8: Compute Average Response Time

Input: A list of `start_times`

Output: The average response time value

```
1 num_jobs ← length of start_times;
2 sum_start ← sum of all values in start_times;
3 if num_jobs == 0 then
4     return 0;                                // No jobs to process
5 else
6     avg_response_time ← sum_start / num_jobs;
7     return avg_response_time;
```

5.1.4 Turnaround time

Thời gian hoàn tất (Turnaround Time) là khoảng thời gian từ lúc một công việc được gửi vào hệ thống cho đến khi nó được hoàn thành. Thời gian hoàn tất trung bình được tính bằng:

$$\text{Average Turnaround Time} = \frac{\sum \text{EndTime}}{\text{NumJob}} \quad (5.9)$$

Turnaround Time là một tiêu chí tổng quát giúp đánh giá hiệu quả tổng thể của giải thuật lập lịch. Nó phản ánh khoảng thời gian người dùng cần chờ để nhận kết quả từ hệ thống. Giá trị càng thấp chứng tỏ giải thuật càng hiệu quả trong việc tối ưu luồng xử lý công việc.

Một giải thuật được tối ưu theo thời gian hoàn tất sẽ có xu hướng giảm độ trễ của toàn bộ hệ thống, bằng cách phân phối công việc hợp lý hơn. Điều này đặc biệt quan trọng với các hệ thống yêu cầu xử lý khối lượng công việc lớn trong thời gian ngắn.

Giả sử có 3 công việc với thời điểm kết thúc lần lượt là: 8, 10, 12. Khi đó:

$$\text{Average Turnaround Time} = \frac{8 + 10 + 12}{3} = \frac{30}{3} = 10 \quad (5.10)$$

Turnaround Time là thời gian trung bình từ khi công việc đến hệ thống cho đến khi nó hoàn tất.

$$\text{Average Turnaround Time} = \frac{\sum \text{EndTime}}{\text{NumJob}}$$

Input

- `end_times`: Danh sách thời điểm kết thúc công việc.

Output

- `avg_turnaround_time`: Thời gian hoàn tất trung bình.

Algorithm 9: Compute Average Turnaround Time

Input: A list of `end_times`

Output: The average turnaround time value

```

1 num_jobs ← length of end_times;
2 sum_end ← sum of all values in end_times;
3 if num_jobs == 0 then
4     return 0;                                // No jobs to process
5 else
6     avg_turnaround_time ← sum_end / num_jobs;
7     return avg_turnaround_time;
```

5.1.5 Makespan

Makespan là khoảng thời gian dài nhất từ khi bắt đầu công việc đầu tiên đến khi hoàn thành công việc cuối cùng. Đây là thước đo phản ánh hiệu quả khai thác tài nguyên hệ thống và thường được dùng trong tối ưu hóa đa công việc hoặc hệ thống phân tán. Công thức:

$$\text{Makespan} = \max(\text{EndTime}) \quad (5.11)$$

Trong môi trường đa nhiệm hoặc phân chia tài nguyên, Makespan càng ngắn chứng tỏ hệ thống hoạt động càng hiệu quả. Giảm Makespan giúp tăng khả năng xử lý luồng công việc mới và giảm chi phí vận hành tài nguyên.

Giải thuật lập lịch cần phân bố công việc một cách thông minh để tránh việc các công việc chồng chéo hoặc dồn về cuối, dẫn đến Makespan cao. Tối ưu Makespan đồng nghĩa với việc giảm tổng thời gian hoạt động của hệ thống.

Giả sử 4 công việc có thời gian kết thúc là: 12, 15, 11, 13. Khi đó:

$$\text{Makespan} = \max(12, 15, 11, 13) = 15 \quad (5.12)$$

Makespan là thời gian kết thúc dài nhất trong số tất cả các công việc, dùng để đo tổng thời gian cần thiết để hoàn thành tất cả công việc.

$$\text{Makespan} = \max(\text{EndTime})$$

Input

- `end_times`: Danh sách thời điểm kết thúc các công việc.

Output

- `makespan`: Giá trị Makespan tính được.

Algorithm 10: Compute Makespan

Input: A list of `end_times`

Output: The makespan value

```

1 if end_times is empty then
2   |   return 0 ;                               // No jobs to process
3 else
4   |   makespan ← maximum value in end_times;
5   |   return makespan;

```

5.1.6 Scheduling latency

Scheduling latency là khoảng thời gian mà giải thuật cần để tạo ra một lịch trình hoàn chỉnh cho việc thực thi các công việc lượng tử. Tiêu chí này không liên quan trực tiếp đến quá trình thực thi công việc mà phản ánh chi phí tính toán cần thiết để sinh ra lịch. Một giải thuật lập lịch dù cho tối ưu đến đâu nhưng mất quá nhiều thời gian để tính

toán sẽ không phù hợp với các hệ thống yêu cầu phản hồi nhanh hoặc có quy mô lớn. Do đó, thời gian sinh lịch là một tiêu chí quan trọng để đánh giá khả năng ứng dụng thực tế của giải thuật. Nếu giải thuật mất nhiều thời gian để sinh lịch, khả năng áp dụng của nó vào thực tế sẽ bị hạn chế. Một số chiến lược lập lịch có thể hy sinh một phần chất lượng lịch để đổi lấy thời gian tính toán ngắn hơn. Giả sử một giải thuật cần 1.25 giây để sinh ra lịch cho 20 công việc. Khi so sánh với một giải thuật khác chỉ mất 0.3 giây, dù kết quả lịch tương đương, giải thuật thứ hai sẽ được ưu tiên trong môi trường yêu cầu phản hồi nhanh. Scheduling latency là tổng thời gian cần để giải thuật lập lịch tạo ra lịch trình hoàn chỉnh cho một nhóm công việc.

5.1.7 Machine Utilization

Hiệu suất sử dụng (Utilization) đo lường mức độ mà tài nguyên lượng tử được tận dụng trong suốt thời gian thực thi công việc. Đây là tỷ lệ giữa tổng thời gian qubit được sử dụng và tổng khả năng tài nguyên mà hệ thống có thể cung cấp.

$$\text{Average Utilization} = \frac{\sum \text{Utilization của các qubit}}{\text{Total num qubit of machine}} \quad (5.13)$$

Việc sử dụng tài nguyên hiệu quả giúp giảm chi phí và tăng hiệu suất hệ thống. Một lịch trình tối ưu không nên để nhiều qubit rảnh rỗi trong thời gian dài. Do đó, Utilization là thước đo giúp xác định chất lượng phân bổ tài nguyên.

Khi tiêu chí Utilization được đưa vào tối ưu hóa, giải thuật sẽ tìm cách phân bổ công việc sao cho tối đa hóa việc sử dụng đồng thời các qubit. Điều này có thể giúp giảm Makespan và tăng Throughput.

Giả sử hệ thống có 10 qubit, trong quá trình lập lịch, tổng thời gian các qubit được sử dụng là 80 đơn vị thời gian. Khi đó:

$$\text{Average Utilization} = \frac{80}{10} = 8 \quad (5.14)$$

Hiệu suất sử dụng đo lường mức độ tận dụng tài nguyên qubit so với tổng tài nguyên có thể cung cấp trong hệ thống.

$$\text{Average Utilization} = \frac{\sum \text{Thời gian sử dụng của các qubit}}{\text{Tổng số qubit của hệ thống}}$$

Input

- `total_used_time`: Tổng thời gian các qubit được sử dụng.
- `total_qubit`: Tổng số qubit mà hệ thống có thể cung cấp.

Output

- `average_utilization`: Giá trị hiệu suất sử dụng trung bình.

Algorithm 11: Compute Average Utilization

Input: `total_used_time`, `total_qubit`

Output: The average utilization

```
1 if total_qubit == 0 then
2   |   return 0 ;                               // Avoid division by zero
3 else
4   |   average_utilization ← total_used_time / total_qubit;
5   |   return average_utilization;
```

5.1.8 Cut Sampling Overhead

Cut Sampling Overhead là chi phí phụ phát sinh trong quá trình kiểm tra, hiệu chỉnh hoặc xác minh kết quả thông qua việc lấy mẫu. Đây là phần chi phí không trực tiếp tham gia vào tính toán nhưng lại ảnh hưởng đến tài nguyên hệ thống.

$$\text{Cut Sampling Overhead} = \sum \text{Sum_CutSampling Overhead} \quad (5.15)$$

Trong máy tính lượng tử, quá trình lấy mẫu để đo hoặc kiểm tra trạng thái lượng tử tiêu tốn tài nguyên đáng kể. Một giải thuật có quá nhiều thao tác lấy mẫu sẽ làm giảm hiệu suất chung. Do đó, đây là một tiêu chí cần được tối ưu.

Khi tiêu chí này được xem xét, giải thuật sẽ hạn chế số lần yêu cầu lấy mẫu, từ đó giảm chi phí gián tiếp và tăng hiệu suất. Tuy nhiên, điều này cần cân bằng với yêu cầu kiểm tra độ chính xác của hệ thống.

Giả sử có 5 công việc với chi phí lấy mẫu lần lượt là 2, 3, 1, 4, và 2 đơn vị. Khi đó:

$$\text{Cut Sampling Overhead} = 2 + 3 + 1 + 4 + 2 = 12 \quad (5.16)$$

Overhead Sampling là tổng chi phí lấy mẫu để kiểm tra, hiệu chỉnh hoặc đo trạng thái lượng tử trong quá trình thực hiện công việc.

$$\text{Cut Sampling Overhead} = \sum \text{Chi phí lấy mẫu của các công việc}$$

Input

- `overhead_list`: Danh sách chi phí lấy mẫu cho từng công việc.

Output

- `overhead_sum`: Tổng chi phí lấy mẫu.

Algorithm 12: Compute Overhead Sampling

Input: A list of `overhead_list`

Output: The total overhead `overhead_sum`

```
1 overhead_sum  $\leftarrow$  0;  
2 foreach overhead in overhead_list do  
3   | overhead_sum  $\leftarrow$  overhead_sum + overhead;  
4 return overhead_sum
```

5.2 Tóm tắt chương

Chương này đã trình bày các tiêu chí đánh giá được sử dụng trong nghiên cứu, bao gồm: độ trung thực, throughput, response time, turnaround time, makespan, scheduling latency, utilization và sampling overhead. Mỗi tiêu chí được giải thích về ý nghĩa, cách đo lường và vai trò trong đánh giá giải thuật lập lịch lượng tử.

Những tiêu chí này sẽ làm cơ sở để đánh giá và so sánh các giải thuật được triển khai trong chương tiếp theo.

Chương 6

Giải thuật định thời lượng tử

Phần này mô tả chi tiết các giải thuật được áp dụng trong nghiên cứu để kiểm tra tính ứng dụng của môi trường lập lịch lượng tử. Các giải thuật được phân thành 3 nhóm chính dựa trên đặc điểm và phương pháp tiếp cận: heuristic, lập trình nguyên tuyến tính (ILP), và trí tuệ nhân tạo (AI-based).

6.1 Tổng quan

Mỗi nhóm giải thuật đại diện cho một hướng tiếp cận khác nhau:

- **Nhóm Heuristic:** Các giải thuật dựa trên nguyên tắc heuristic, đơn giản, dễ cài đặt, và có tốc độ xử lý nhanh. Mục tiêu chính là tìm lời giải “tốt đủ” trong thời gian ngắn, thay vì lời giải tối ưu toàn cục.
- **Nhóm ILP (Integer Linear Programming):** Các giải thuật sử dụng mô hình toán học tuyến tính với ràng buộc nguyên, giải bài toán tối ưu hóa lập lịch bằng các solver.
- **Nhóm AI (Trí tuệ nhân tạo):** Các giải thuật dựa trên machine learning hoặc reinforcement learning, học từ dữ liệu lịch sử hoặc qua tương tác với môi trường để dần cải thiện chiến lược lập lịch.

Nội dung chương này sẽ trình bày chi tiết từng nhóm giải thuật, bao gồm: nguyên lý hoạt động, ưu nhược điểm, mã giả, và nhận xét ứng dụng.

6.2 Mô tả giải thuật

Trong phần này, chúng tôi sẽ trình bày chi tiết các giải thuật lập lịch đã được triển khai trong hệ thống mô phỏng.

Các phần sau đây sẽ lần lượt mô tả:

- Nguyên lý hoạt động cốt lõi của từng giải thuật.
- Cách vận hành của các giải thuật
- Đặc điểm nổi bật, ưu và nhược điểm của mỗi giải thuật.

Thông qua việc phân tích riêng từng thuật toán, chúng tôi hướng tới việc so sánh toàn diện hơn ở các chương sau, từ đó lựa chọn giải pháp phù hợp cho từng loại bài toán hoặc cấu hình hệ thống cụ thể.

6.2.1 First-Fit Decreasing Bin Packing

Giải thuật First-Fit Decreasing (FFD) [11] là một phương pháp xấp xỉ cổ điển cho bài toán phân phối tài nguyên hữu hạn, trong đó các tác vụ (jobs) được xếp vào các vùng chứa (bins), sao cho mỗi vùng chứa không vượt quá giới hạn tài nguyên, và số lượng vùng chứa được sử dụng là ít nhất có thể.

Trong bối cảnh lập lịch lượng tử, mỗi công việc lượng tử yêu cầu một số lượng qubit nhất định, và các thiết bị lượng tử (bins) có giới hạn qubit cụ thể. Mục tiêu là phân phối các công việc lên thiết bị sao cho tối ưu việc sử dụng tài nguyên và số thiết bị cần dùng.

Nguyên lý hoạt động

Giải thuật hoạt động theo ba bước chính:

- **Sắp xếp công việc:** Các công việc được sắp xếp giảm dần theo số lượng qubit yêu cầu.
- **Phân phối công việc:** Duyệt lần lượt từng công việc, đặt vào bin đầu tiên có thể chứa nó (first-fit).
- **Mở rộng tài nguyên nếu cần:** Nếu không có bin nào phù hợp, giải thuật mở thêm bin mới từ danh sách thiết bị ban đầu.

Mã giả của giải thuật

Algorithm 13: Scheduling with First-Fit Decreasing Bin Packing

Input: J : set of jobs; B : list of devices (bins)

Output: List of bins with assigned jobs

```
1  $J' \leftarrow$  Sort  $J$  in decreasing order of required qubits;
2  $open \leftarrow B$ ;                                     // Available devices (bins)
3  $closed \leftarrow \emptyset$ ;                         // Full devices (bins)
4 foreach  $job \in J'$  do
5      $bin \leftarrow \text{FindFirstFitting}(job, open)$ ;
6     if  $bin = none$  then
7          $new \leftarrow B$ ;
8          $bin \leftarrow \text{FindFirstFitting}(job, new)$ ;
9         add  $new$  to  $open$ ;
10    add  $job$  to  $bin$ ;
11    if  $bin$  is full then
12        remove  $bin$  from  $open$ ;
13        add  $bin$  to  $closed$ ;
14 add all  $open$  to  $closed$ ;
15 return  $closed$ ;
```

Ưu điểm

- Đơn giản, dễ cài đặt.
- Có tốc độ chạy nhanh, hiệu quả với các tập công việc có yêu cầu qubit không chênh lệch quá lớn.
- Cho kết quả gần tối ưu trong nhiều trường hợp thực tế.

Nhược điểm

- Không đảm bảo tối ưu toàn cục.

- Nếu các công việc có kích thước chênh lệch lớn, dễ dẫn đến phân bổ không đều, gây lãng phí tài nguyên.
- Không xét đến các yếu tố khác như fidelity, latency hoặc thời gian thực thi - cần kết hợp thêm trong mô hình tổng thể.

Nhận xét ứng dụng trong môi trường lượng tử

Giải thuật FFD phù hợp với bước phân phối công việc lượng tử ban đầu lên các thiết bị có giới hạn tài nguyên. Trong môi trường lượng tử, việc sử dụng tài nguyên hiệu quả là thiết yếu, và FFD có thể đóng vai trò như một baseline (chuẩn so sánh) để đối chiếu với các chiến lược lập lịch thông minh hơn như MILQ hoặc RL-based.

Mặc dù vậy, trong các môi trường phức tạp hơn — nơi mà fidelity, thời gian thực thi và độ ưu tiên giữa các job đóng vai trò quan trọng — cần tích hợp thêm các module đánh giá đa mục tiêu hoặc heuristic bổ sung để cải thiện chất lượng lập lịch.

6.2.2 MILP (Mixed Integer Linear Programming)

Giải thuật MILP (Mixed Integer Linear Programming) [9], [53] áp dụng mô hình tối ưu hóa nguyên tuyến tính hỗn hợp cho bài toán lập lịch tác vụ lượng tử trên nhiều thiết bị. Phương pháp này sử dụng một tập các biến quyết định nhị phân và liên tục, với mục tiêu chính là tối thiểu hóa **makespan** — thời gian hoàn thành trễ nhất của tất cả các công việc.

Hàm mục tiêu

Hàm mục tiêu được định nghĩa như sau:

$$\min(c_{max})$$

trong đó c_{max} là thời gian hoàn tất muộn nhất của tất cả các công việc, đại diện cho makespan của hệ thống.

Các biến

Dựa trên Bảng MILP Notation (Table I), các biến trong mô hình bao gồm:

- **Biến nhị phân:**

- x_{jm} : 1 nếu công việc j được gán trên máy m ; 0 nếu không.
- y_{ijm} : 1 nếu công việc j là hậu nhiệm của công việc i trên máy m .
- z_{jmt} : 1 nếu công việc j được thực thi trên máy m tại thời điểm t .

- **Biến thực:**

- b_j : thời gian bắt đầu công việc j .
- c_j : thời gian hoàn thành công việc j .

- **Các biến trợ giúp nhị phân:** $\alpha_{ij}, \beta_{ij}, \gamma_{jm}, \delta_{ijkm}$ phục vụ ràng buộc thứ tự và lập lịch.

Phân tích sơ bộ chính

Mô hình MILP bao gồm một tập các ràng buộc đảm bảo tính khả thi của lịch:

- **Ràng buộc (C1)-(C2):** Đảm bảo $c_j \leq c_{max}$ với mọi $j \in J$, đặt $c_0 = 0$ (dummy job).
- **Ràng buộc (C3)-(C4):** Mỗi công việc chỉ được gán trên duy nhất một máy và chỉ thực thi một lần.
- **Ràng buộc (C5):** Tính thời gian hoàn thành công việc dựa trên thời gian bắt đầu, thời gian xử lý p_{jm} , và thời gian setup s_{ijm} .
- **Ràng buộc (C6):** Đảm bảo tất cả các công việc tiền nhiệm phải hoàn thành trước khi công việc j bắt đầu.
- **Ràng buộc (C7)-(C9):** Liên kết các biến thời gian, đảm bảo tính hợp lệ giữa z_{jmt} (biến theo thời gian), x_{jm} , và c_j, b_j .
- **Ràng buộc (C10)-(C11):** Đảm bảo tài nguyên (số qubit) được phân bổ hợp lý không vượt quá giới hạn của thiết bị.
- **Ràng buộc (C12)-(C14):** Đảm bảo quan hệ hậu nhiệm giữa các công việc trên cùng một máy.

- **Ràng buộc (C15)-(C20):** Ràng buộc phụ trợ và logic nhằm đảm bảo quan hệ thứ tự giữa các công việc trên cùng thiết bị, tránh chồng chéo.

Hàm mục tiêu

Hàm mục tiêu của mô hình:

$$\min(c_{max})$$

trong đó c_{max} là thời gian hoàn thành muộn nhất của tất cả các công việc.

Mô tả chi tiết các ràng buộc

Mô hình MILP được xây dựng với 20 ràng buộc chính, đảm bảo tính khả thi của lịch trình lập lịch. Mỗi ràng buộc đóng một vai trò cụ thể:

- **(C1):** Ràng buộc đảm bảo thời gian hoàn thành c_j của mọi công việc j không vượt quá c_{max} (makespan). Giúp liên kết thời gian hoàn thành của từng công việc với hàm mục tiêu.

$$c_j \leq c_{max}, \quad \forall j \in J$$

- **(C2):** Đặt thời gian hoàn thành của dummy job (công việc giả định 0) là 0. Dummy job được dùng làm mốc khởi đầu.

$$c_0 = 0$$

- **(C3):** Mỗi công việc j chỉ được gán cho đúng một thiết bị m . Không có công việc nào chạy đồng thời trên nhiều thiết bị.

$$\sum_{m \in M} x_{jm} = 1, \quad \forall j \in J$$

- **(C4):** Mỗi công việc chỉ được thực hiện tại một thời điểm duy nhất trên timeline:

$$\sum_{m \in M} z_{jmt} \leq 1, \quad \forall j \in J, \forall t \in T$$

- **(C5):** Tính toán thời gian hoàn thành c_j dựa trên thời gian bắt đầu b_j , thời gian xử lý p_{jm} , thời gian setup từ công việc tiền nhiệm s_{ijm} (nếu có), và biến quan hệ tiền nhiệm y_{ijm} .

$$c_j \geq b_j + \sum_{m \in M} p_{jm} \cdot x_{jm} + \sum_{i \in J \cup \{0\}} \sum_{m \in M} s_{ijm} \cdot y_{ijm}$$

- **(C6):** Đảm bảo rằng mọi công việc tiền nhiệm của j đều kết thúc trước khi j bắt đầu. Ràng buộc này bảo vệ mối quan hệ phụ thuộc giữa các công việc.

$$b_j \geq c_i + M \left(\sum_{m \in M} y_{ijm} - 1 \right), \quad \forall j \in J, \forall i \in J \cup \{0\}$$

- **(C7):** Liên kết thời gian hoàn thành với biến z_{jmt} , đảm bảo c_j tương ứng với thời điểm cuối t mà công việc j được thực hiện trên m .

$$\sum_{m \in M} \sum_{t \in T} z_{jmt} = c_j - b_j + 1, \quad \forall j \in J$$

- **(C8):** Đảm bảo rằng nếu $x_{jm} = 1$ (công việc j gán trên máy m), thì tổng số timestep công việc đó thực hiện trên m phải đủ lớn.

$$\sum_{t \in T} z_{jmt} \leq M \cdot x_{jm}, \quad \forall j \in J, \forall m \in M$$

- **(C9):** Mọi thời điểm bắt đầu b_j của công việc phải lớn hơn hoặc bằng thời điểm đầu tiên t mà $z_{jmt} = 1$:

$$b_j \geq t \cdot \sum_{m \in M} z_{jmt}, \quad \forall j \in J, \forall t \in T$$

- **(C10):** Ngược lại, thời điểm kết thúc s_j phải nhỏ hơn thời điểm t cuối cùng mà $z_{jmt} = 1$:

$$s_j \leq t \cdot \sum_{m \in M} z_{jmt} + M \left(1 - \sum_{m \in M} z_{jmt} \right)$$

- **(C11):** Tổng tài nguyên (số qubit) mà các công việc sử dụng trên một thiết bị m

tại thời điểm t không được vượt quá Q_m :

$$\sum_{j \in J} q_j \cdot z_{jmt} \leq Q_m, \quad \forall m \in M, \forall t \in T$$

- **(C12):** Mỗi công việc (trừ dummy job) phải có ít nhất một tiền nhiệm:

$$1 \leq \sum_{i \in J \cup \{0\}} \sum_{m \in M} y_{ijm}, \quad \forall j \in J$$

- **(C13):** Nếu $x_{jm} = 1$, tổng tiền nhiệm i của j trên m phải bằng 1:

$$M \cdot x_{jm} \geq \sum_{i \in J \cup \{0\}} y_{ijm}, \quad \forall j \in J, \forall m \in M$$

- **(C14):** Tương tự, nếu $x_{jm} = 1$, tổng số hậu nhiệm j của i trên m phải bằng 1:

$$M \cdot x_{jm} \geq \sum_{i \in J \cup \{0\}} y_{ijm}, \quad \forall j \in J, \forall m \in M$$

- **(C15):** Đảm bảo rằng công việc dummy không phải là tiền nhiệm của chính nó:

$$z_{j0m} = y_{j0m}, \quad \forall j \in J, \forall m \in M$$

- **(C16)-(C19):** Ràng buộc phụ trợ, tạo biến logic $\alpha_{ij}, \beta_{ij}, \gamma_{jm}, \delta_{ijkm}$ để hỗ trợ ràng buộc thứ tự giữa các công việc i, j, k (ví dụ: đảm bảo i hoàn thành trước j và j hoàn thành trước k).

- **(C20):** Đảm bảo thứ tự giữa các công việc khi sắp xếp hậu nhiệm, ngăn xung đột thứ tự.

$$y_{ij} \leq \gamma_{jm} + (1 - \sum_{k \in J} \delta_{ijkm}) + \gamma_{im} - 2$$

Ý nghĩa tổng quan

Mục tiêu của mô hình MILP là tìm ra một lịch trình phân bổ các công việc lượng tử lên nhiều thiết bị sao cho:

- Thời gian hoàn thành toàn bộ (makespan) là ngắn nhất.
- Tuân thủ ràng buộc tài nguyên (số qubit trên từng thiết bị).
- Đảm bảo tính liên tục và thứ tự xử lý, không vi phạm quan hệ tiền nhiệm-hậu nhiệm giữa các công việc.

Mô hình cho phép xử lý thêm các yếu tố phức tạp như:

- Thời gian thiết lập lại thiết bị giữa các công việc (s_{ijm}).
- Giới hạn tài nguyên động (số qubit khả dụng Q_m).
- Lập lịch nhiều thiết bị đồng thời với nhiều công việc phụ thuộc lẫn nhau.

Ưu điểm

- Cung cấp lời giải tối ưu toàn cục nếu solver hội tụ.
- Mô hình hóa được nhiều yếu tố phức tạp của hệ thống lượng tử (setup, resource, dependency).
- Có thể mở rộng thêm ràng buộc hoặc hàm mục tiêu cho các yếu tố khác.

Nhược điểm

- Chi phí tính toán cao khi kích thước bài toán tăng (số job, số thiết bị, độ phức tạp ràng buộc).
- Không phù hợp cho hệ thống yêu cầu phản hồi nhanh hoặc môi trường động.
- Phụ thuộc vào khả năng hội tụ và hiệu năng của solver MILP.

Nhận xét ứng dụng trong môi trường lượng tử

Giải thuật MILP phù hợp làm baseline cho bài toán lập lịch lượng tử trong môi trường quy mô vừa và nhỏ, nơi cần độ chính xác cao và không yêu cầu thời gian tính toán thực thi ngắn. Mô hình đặc biệt hữu ích trong các nghiên cứu so sánh hiệu quả giải thuật, hoặc trong giai đoạn thiết kế lập lịch offline (tạo lịch trước khi thực thi).

Trong môi trường quy mô lớn hoặc thời gian thực, MILP có thể cần được thay thế bằng các heuristic hoặc AI-based approach nhằm giảm chi phí tính toán.

6.2.3 Noise and Time Aware Distributed Scheduler (NoTADS)

Giải thuật NoTADS (Noise and Time Aware Distributed Scheduler) [7] là một mô hình lập lịch dựa trên Integer Linear Programming (ILP), nhằm mục tiêu phân bổ các **subcircuit** (mạch con) sau khi cắt từ một mạch lượng tử ban đầu C lên các phần cứng lượng tử khác nhau, sao cho tối ưu hoá độ trung thực (fidelity) trong khi vẫn đảm bảo thời gian thực thi không vượt quá giới hạn quy định trên từng phần cứng.

Mục tiêu tối ưu hóa

Mục tiêu của giải thuật NoTADS là **tối đa hóa độ trung thực tổng thể của toàn bộ mạch**, thông qua việc lựa chọn phần cứng phù hợp cho từng mạch con. Độ trung thực được phản ánh thông qua một chỉ số **mapomatic score** Q_{ij} [49] — đánh giá mức độ phù hợp của việc gán subcircuit i lên phần cứng j . Do đó, bài toán trở thành bài toán tối ưu hoá hàm mục tiêu tuyến tính:

$$\min \sum_{i \in C} \sum_{j \in H} X_{ij} \cdot Q_{ij}$$

trong đó:

- X_{ij} là biến nhị phân: $X_{ij} = 1$ nếu mạch con i được gán cho phần cứng j ; ngược lại $X_{ij} = 0$.
- Q_{ij} là mapomatic score của việc gán i lên j .

Hàm mục tiêu này tìm kiếm phép gán sao cho tổng score là nhỏ nhất, đồng nghĩa với độ trung thực cao nhất.

Các biến

- X_{ij} : biến nhị phân quyết định việc subcircuit i được gán lên phần cứng j .

- η_i : số lượng instance của subcircuit i ; được xác định bởi:

$$\eta_i = \begin{cases} 1 & \text{nếu tất cả instance của } i \text{ lập lịch riêng biệt} \\ v(\rho_i, O_i) & \text{ngược lại} \end{cases}$$

trong đó $v(\rho_i, O_i)$ là hàm tính số lượng instance dựa trên số qubit chuẩn bị ρ_i và số qubit đo lường O_i .

Các ràng buộc

Mô hình ILP bao gồm hai ràng buộc chính:

1. **Ràng buộc phân bổ:** mỗi subcircuit i bắt buộc phải được gán cho đúng một phần cứng j :

$$\sum_{j \in H_i} X_{ij} = 1 \quad \forall i \in C$$

(ràng buộc này đảm bảo mỗi mạch con được lập lịch duy nhất trên một thiết bị).

2. **Ràng buộc thời gian:** tổng thời gian thực thi tất cả subcircuit được gán lên cùng một phần cứng j không vượt quá giới hạn thời gian τ_j của thiết bị đó:

$$\sum_{i \in C} \eta_i \cdot t_i \cdot X_{ij} \leq \tau_j \quad \forall j \in H$$

trong đó:

- t_i : thời gian thực thi subcircuit i .
- τ_j : thời gian tối đa cho phép trên phần cứng j .

Ý nghĩa

Giải thuật NoTADS đồng thời tối ưu:

- Độ trung thực tổng thể (qua minimization of Q_{ij}).
- Phân bổ tài nguyên sao cho không vượt quá thời gian cho phép của từng thiết bị.

Nhận xét

- **Ưu điểm:**

- Có thể xử lý sự khác biệt về nhiễu (noise profile) của từng phần cứng.
- Đảm bảo thời gian thực thi hợp lệ.
- Hàm mục tiêu tuyến tính cho phép giải nhanh hơn so với mô hình phi tuyến.

- **Nhược điểm:**

- Giả định các subcircuit có độ lớn tương đương (số qubit và độ sâu).
- Nếu subcircuit có độ không cân bằng lớn, hàm mục tiêu tuyến tính có thể không đảm bảo tối ưu toàn cục; cần hàm mục tiêu phi tuyến trong trường hợp đó.

Ứng dụng

NoTADS phù hợp trong bối cảnh:

- Có nhiều subcircuit với profile noise khác nhau.
- Cần lập lịch trên nhiều thiết bị quantum song song.
- Muốn cân bằng giữa fidelity và thời gian thực thi.

Theo kết quả nghiên cứu, nếu việc cắt mạch tạo ra các subcircuit tương đối cân bằng, hàm mục tiêu tuyến tính là đủ để đạt hiệu quả tốt mà không cần phức tạp hóa mô hình bằng phi tuyến.

6.2.4 Multi-Task Multi-Chip Scheduling (MTMC)

Giải thuật Multi-Task Multi-Chip Scheduling (MTMC) [12] là một phương pháp heuristic được thiết kế để phân bổ đồng thời nhiều tác vụ lượng tử (multi-task) lên nhiều thiết bị lượng tử (multi-chip). Mục tiêu chính là tìm một lịch trình phân bổ hiệu quả, đảm bảo số lượng tác vụ được thực thi thành công tối đa, đồng thời giảm thiểu số lần lặp (loop) không thành công.

Ý tưởng tổng quan

Giải thuật hoạt động theo nguyên lý:

- Bắt đầu với toàn bộ danh sách tác vụ cần lập lịch.
- Tìm kiếm tập tác vụ khả thi có thể phân bổ lên các chip, tuân thủ ràng buộc tài nguyên và dung lượng.
- Nếu không thể lập lịch toàn bộ tác vụ, giải thuật sẽ cố gắng lập lịch từng phần, loại bỏ tạm thời các tác vụ không khả thi, và tiếp tục lập lịch phần còn lại.
- Lặp lại quá trình cho đến khi không còn tác vụ nào hoặc đạt giới hạn lặp.

Giải thích chi tiết giải thuật

Giải thuật được mô tả qua pseudocode:

Algorithm 14: Algorithm of Multi-Task Multi-Chip Scheduling

Input: $chiplist, tasks$ **Output:** $results_{success}, results_{failed}$

```
1  $task_{remaining} \leftarrow tasks;$ 
2  $count_{loop} \leftarrow 0;$ 
3  $N \leftarrow 0;$ 
4  $tasklist \leftarrow NULL;$ 
5  $task_{no.execute} \leftarrow NULL;$ 
6  $templist \leftarrow NULL;$ 
7  $task_{allocated} \leftarrow NULL;$ 
8  $task_{no.allocated} \leftarrow NULL;$ 
9  $results_{success} \leftarrow NULL;$ 
10  $results_{failed} \leftarrow NULL;$ 
11 while  $task_{remaining} \neq NULL$  or  $task_{no.execute} \neq NULL$  do
12   if  $length(task_{no.execute}) \geq \epsilon$  then
13      $tasklist \leftarrow sort(task_{no.execute});$ 
14   else
15      $N \leftarrow \epsilon - length(task_{no.execute});$ 
16      $task_{remaining} \leftarrow task_N;$ 
17      $tasklist \leftarrow sort(task_{no.execute}) + sort(task_{remaining});$ 
18    $task_{allocated}, task_{no.execute} \leftarrow$ 
19      $MultiTaskMultiChipsPreAlloc(tasklist, chiplist);$ 
20    $results_{success}, task_{no.execute} \leftarrow submit\_entry\_parallel(task_{allocated});$ 
21   return  $Tidy\_Result\_and\_Display(results_{success}, task_{no.execute});$ 
22   if  $count_{loop} == 0$  then
23      $templist \leftarrow task_{no.execute};$ 
24   if  $count_{loop} == loops$  then
25      $task_{allocated} \leftarrow task$  in  $task_{no.execute}$  and  $templist;$ 
26     return  $results_{failed};$ 
27      $task_{no.execute} \leftarrow task$  in  $task_{no.execute}$  but not in  $templist;$ 
28      $count_{loop} \leftarrow count_{loop} - 1;$ 
29    $count_{loop} \leftarrow count_{loop} + 1;$ 
```

1. Khởi tạo:

- Gán $task_{remaining} \leftarrow tasks$: danh sách tác vụ cần lập lịch.
- Các biến trung gian $task_{list}, task_{no.execute}, temp_{list}, task_{allocated}, result_{success}, result_{failed}$ được khởi tạo NULL.
- Đếm số vòng lặp $count_{loop} \leftarrow 0$.

2. Vòng lặp chính: giải thuật tiếp tục lặp cho đến khi $task_{remaining}$ rỗng hoặc đạt ngưỡng lặp.

$$\text{while } task_{remaining} \neq NULL \text{ or } task_{no.execute} \neq NULL$$

3. Kiểm tra số lượng tác vụ khả thi: Nếu số tác vụ khả thi trong $task_{no.execute}$ lớn hơn ϵ (một ngưỡng nhỏ), sắp xếp lại:

$$task_{list} \leftarrow sort(task_{no.execute})$$

Nếu không:

$$N \leftarrow \epsilon - length(task_{no.execute})$$

Thêm N tác vụ từ $task_{remaining}$ vào:

$$task \leftarrow sort(task_{no.execute}) + sort(task_{remaining})$$

4. Tiền xử lý phân bổ: Gọi hàm:

$$task_{allocated}, task_{no.execute} \leftarrow MultiTaskMultiChipsPreAlloc(task_{list}, chip_{list})$$

Hàm này xác định tác vụ nào có thể phân bổ lên chip, tác vụ nào chưa phân bổ được.

5. Phân bổ song song: Nếu có tác vụ khả thi:

$$result_{success}, task_{no.execute} \leftarrow submit_entry_parallel(task_{allocated})$$

Hiển thị kết quả:

returnTidy_Result_and_Display(result_{success}, task_{no.execute})

6. **Xử lý khi không phân bổ được:** Nếu $count_{loop} == 0$, lưu $task_{no.execute}$ vào $templist$.

Nếu $count_{loop}$ đạt $loops$ (số lần lặp tối đa), kết hợp tác vụ chưa phân bổ với $templist$ và trả về $result_{failed}$.

Nếu không, giảm $count_{loop}$ và tiếp tục vòng lặp.

Ưu điểm

- Tốc độ xử lý nhanh do bản chất heuristic.
- Có khả năng xử lý nhiều tác vụ song song trên nhiều thiết bị.
- Có cơ chế loại tác vụ không khả thi, giảm nghẽn.

Nhược điểm

- Không đảm bảo tìm được lời giải tối ưu toàn cục.
- Phụ thuộc vào chiến lược sắp xếp (sort) và hàm tiền xử lý *MultiTaskMultiChipsPreAlloc*.
- Khi $task_{no.execute}$ quá nhiều, dễ bị lặp lại hoặc không hội tụ.

Ứng dụng

Giải thuật MTMC phù hợp cho các bài toán:

- Lập lịch nhanh trên hệ thống quantum multi-chip.
- Môi trường yêu cầu throughput cao, xử lý batch nhiều tác vụ.
- Kết hợp với các pipeline lập lịch phức tạp hơn như circuit cutting hoặc hybrid scheduler.

6.3 Bảng so sánh

Bảng 6.1: So sánh các giải thuật lập lịch lượng tử

Tiêu chí	FFD (Heuristic)	MILP	NoTADS	MTMC
Độ phức tạp	Thấp	Rất cao	Cao	Trung bình
Tốc độ thực thi	Nhanh	Chậm (solver)	Chậm – Trung bình	Nhanh
Độ chính xác (Fidelity)	Thấp – Trung bình	Cao	Cao	Trung bình – Cao
Khả năng mở rộng (scale)	Cao	Thấp – Trung bình	Trung bình	Cao
Xử lý môi trường nhiều backend	Không	Có (qua biến)	Có (đa backend, noise-aware)	Có (đa chip)
Tối ưu độ trung thực	Không xét	Không trực tiếp	Có (tối ưu fidelity)	Không xét trực tiếp
Tối ưu thời gian thực thi	Không xét	Có (makespan)	Có (giới hạn thời gian)	Giảm lặp không thành công
Phù hợp môi trường động	Thấp	Thấp	Trung bình – Cao	Cao
Ứng dụng phù hợp	Baseline nhanh, ít yêu cầu	Quy mô nhỏ/giữa, cần tối ưu chính xác	Nhiều subcircuit, cần balance fidelity và thời gian	Nhiều task, multi-chip, throughput cao

6.4 Tóm tắt chương

Chương này đã giới thiệu bốn giải thuật lập lịch gồm: First-Fit Decreasing (FFD) [11], MILP [9], NoTADS [7] và MTMC [12], cùng nguyên lý, ưu nhược điểm và ứng dụng của từng giải thuật. Bảng so sánh tổng quan cũng được trình bày nhằm đối chiếu các phương pháp.

Các giải thuật này sẽ được triển khai và đánh giá thực nghiệm trong Chương 7, nhằm phân tích hiệu quả và khả năng ứng dụng trên dữ liệu mô phỏng.

Chương 7

Đánh giá giải thuật

Trong chương 7, các tiêu chí đánh giá, quá trình thực nghiệm và kết quả so sánh các giải thuật lập lịch trên môi trường mô phỏng được trình bày chi tiết.

7.1 Mô tả hệ thống thực nghiệm

Thực nghiệm được triển khai trên hệ thống phần cứng cá nhân với cấu hình:

- CPU: Intel Core i7-13650HX (1.9 GHz - 4.9 GHz, 14 nhân, 20 luồng)
- RAM: 16 GB.
- Hệ điều hành: Zorin 22.04 LTS.
- Phần mềm: Python 3.10, Qiskit 1.4, NumPy, Matplotlib.

Môi trường thực nghiệm được xây dựng hoàn toàn **offline**, sử dụng các máy mô phỏng (Fake machines) để đảm bảo tính tái lập mang kết quả sát với phần cứng lượng tử thực tế.

7.2 Mô tả benchmark thực nghiệm

Để đánh giá các giải thuật, một bộ *benchmark* được thiết lập dựa trên các biến đầu vào chính:

- **Danh sách máy lượng tử (QPU):** tập hợp các backend mô phỏng các hệ thống lượng tử thật của IBM Quantum.

- **Danh sách mạch lượng tử (batch):** mỗi batch gồm nhiều mạch lượng tử, mỗi mạch được mô tả bởi:
 - **type of circuit:** loại mạch (GHZ, QAOA, Random Circuit).
 - **size of circuit:** số qubit của mỗi mạch, dao động từ 2–130 qubit.
 - **num_job:** số lượng mạch trong mỗi batch (không giới hạn).

Mỗi tổ hợp giá trị của QPU, num_job, size_circuit, type_circuit được coi là một **scenario** (tình huống thử nghiệm):

$$Scenario = \{QPU_s, num_jobs, size_circuit, type_circuit\}$$

Kết quả thực nghiệm của mỗi scenario được lưu vào file .json chứa toàn bộ các chỉ số đánh giá (metrics).

7.3 Đánh giá mối liên hệ đầu vào - chỉ số

Bảng 7.1 tóm tắt mối quan hệ giữa các biến đầu vào và tác động của chúng đến các chỉ số đánh giá.

Bảng 7.1: Mối liên hệ giữa các biến đầu vào và các chỉ số đánh giá

Biến đầu vào	Ảnh hưởng chính
Size_circuit	Tăng kích thước → depth tăng → fidelity giảm; makespan, utilization tăng.
Type_circuit	GHZ: shallow, entanglement cao; QAOA: deep, nhiều gate; Random: khó dự đoán → ảnh hưởng fidelity, throughput.
Num_jobs	Tăng số job → throughput tăng; đồng thời response time, turnaround time cũng tăng.
QPU	Backend khác nhau → ảnh hưởng constraint kết nối → tác động fidelity, overhead sampling.

7.4 Thử nghiệm

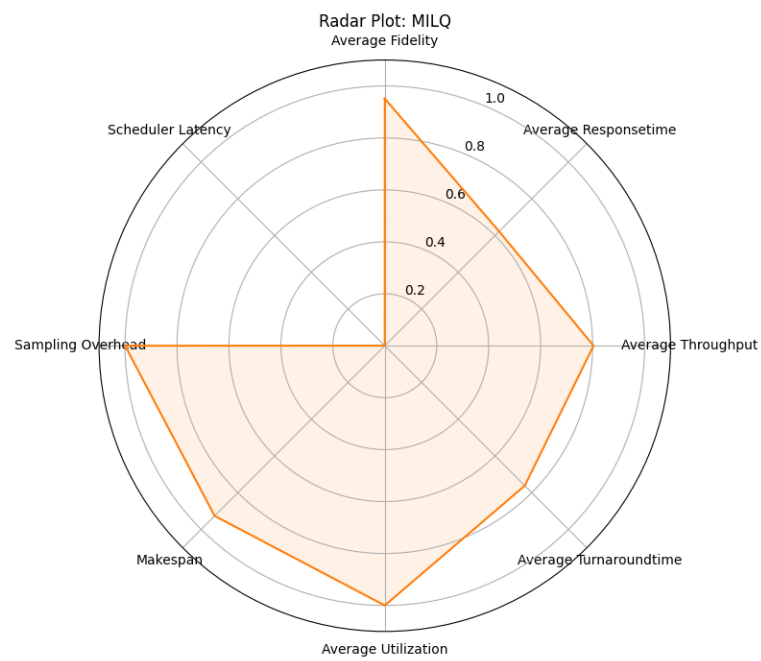
Để kiểm tra thực nghiệm chúng tôi thay đổi các tham số đầu vào, số lượng mạch trong batch (`num_jobs`) được tăng dần từ 2 đến 9 và số qubit trong mỗi circuit tăng từ 2 đến 10 trong khi các biến còn lại cố định:

- QPU: {ibmq_belem: 5 qubits, ibmq_quito: 5 qubits}).
- `type_circuit`: GHZ.
- `type machines`: multithreading

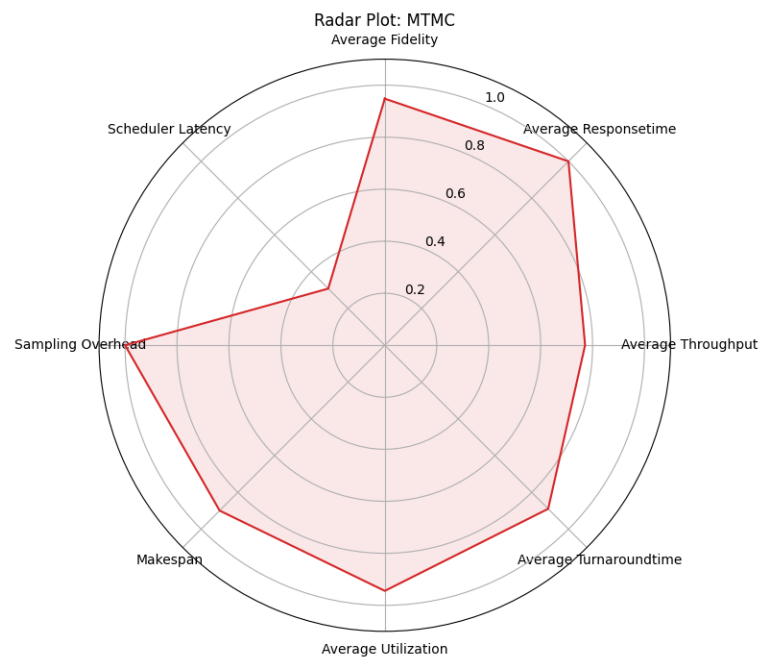
Sau khi thu thập dữ liệu, nhóm đồ án tính toán số liệu chuẩn hóa bằng tỉ lệ giá trị nhỏ nhất với cao nhất. Sau đó sẽ tính trung bình các giá trị chuẩn hóa lại để ra kết quả tổng thể. Biểu đồ chuẩn hóa



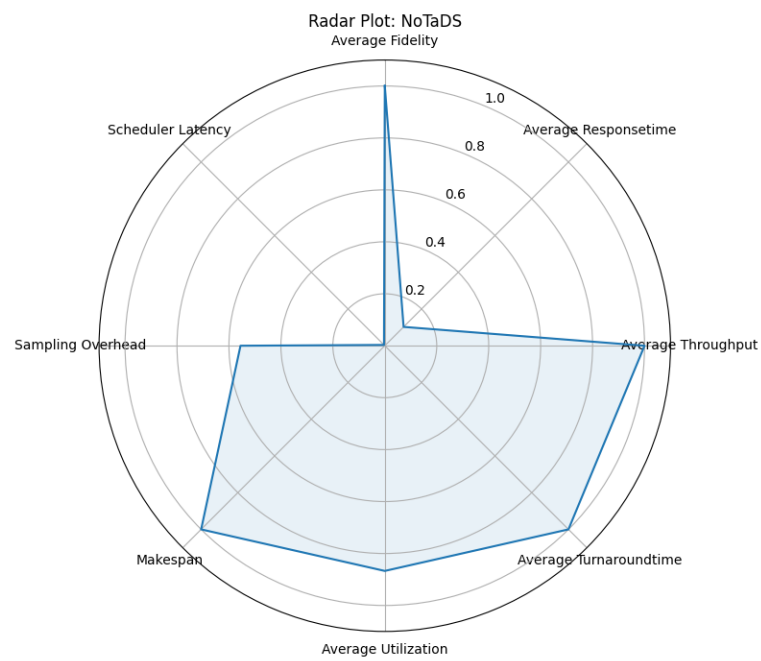
Hình 7.1: Biểu đồ chuẩn hóa của thuật toán FFD.



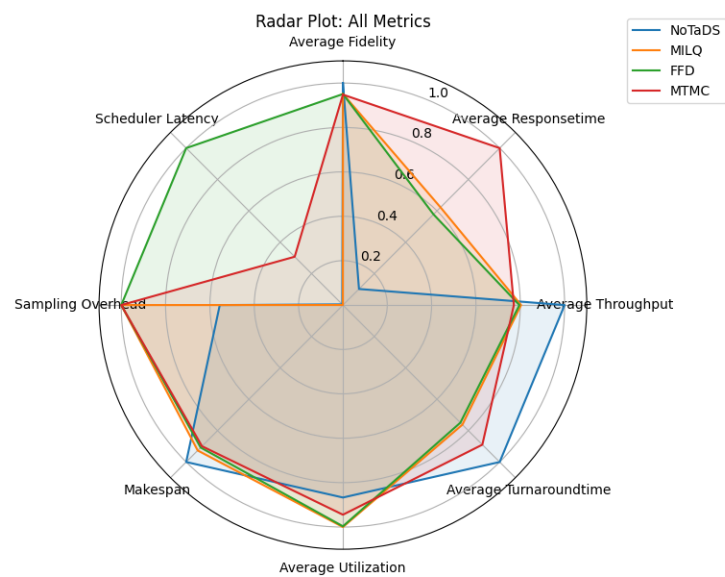
Hình 7.2: Biểu đồ chuẩn hóa của thuật toán MILQ.



Hình 7.3: Biểu đồ chuẩn hóa của thuật toán MTMC.



Hình 7.4: Biểu đồ chuẩn hóa của thuật toán NoTADS.



Hình 7.5: Biểu đồ chuẩn hóa tổng hợp kết quả các giải thuật.

Bảng 7.2: Xếp hạng thuật toán theo các tiêu chí

Thuật toán	Tiêu chí									
	fid	sch_latency	sam_ovh	mk.span	util	turnaround	throughput	respTime	total_score	overall_rank
FFD [11]	3	2	2	3	1	3	2	3	19	3
MILQ [9]	4	4	1	1	3	1	3	1	18	2
MTMC [12]	2	1	2	4	2	2	1	2	16	1
NoTADS [7]	1	3	4	2	4	4	4	4	26	4

7.5 Kết quả đánh giá

Kết quả tổng hợp được trình bày qua bảng số liệu và các biểu đồ trực quan. Các biểu đồ so sánh hiệu năng về fidelity, throughput, turnaround time, response time, makespan... cho thấy xu hướng:

- Fidelity giảm khi size_circuit tăng vì lý do khi kích thước mạch tăng lên dẫn tới độ chiếm dụng nhiều hơn trong máy. Vì vậy các qubit sẽ bị độ nhiễu cao từ các node không ổn định trong máy dẫn tới giá trị fidelity trung bình bị giảm.
- Throughput tăng tuyến tính với num_job do có nhiều mạch được xử lý song song hơn, tuy nhiên điều này đồng thời tạo áp lực lên hàng đợi xử lý, khiến response time trung bình tăng do phải chờ lâu hơn.
- ILP không khả thi khi num_job lớn hoặc size_circuit tăng cao, vì số biến và ràng buộc trong mô hình toán tăng nhanh theo quy mô bài toán, khiến thời gian giải tăng phi tuyến, thậm chí không hội tụ.
- Heuristic giữ hiệu năng ổn định hơn khi các input đầu vào thay đổi, vì các thuật toán heuristic thường sử dụng quy tắc đơn giản, không phụ thuộc mạnh vào quy mô bài toán hoặc cấu trúc cụ thể của mạch, giúp chúng phản ứng nhanh và duy trì hiệu suất ổn định ngay cả khi số lượng job hay kích thước mạch thay đổi.

7.6 Phân tích kết quả

Dựa trên kết quả phân tích và bảng xếp hạng, nhóm nghiên cứu nhận thấy đặc điểm nổi bật của từng giải thuật như sau:

- **Giải thuật MTMC** đạt **xếp hạng tổng thể cao nhất** (hạng 1), nhờ duy trì hiệu suất cân bằng trên hầu hết các tiêu chí. MTMC xếp hạng cao ở các tiêu chí quan trọng như *average_fidelity* (hạng 2), *average_throughput* (hạng 1), và *scheduling latency* (hạng 1). Điều này cho thấy MTMC phù hợp trong các môi trường cần cân bằng giữa độ chính xác, tốc độ xử lý và hiệu quả tài nguyên.
- **Giải thuật MILQ_extend** đạt **xếp hạng tổng thể thứ 2**, nổi bật ở tiêu chí *sampling_overhead* (hạng 1) và *makespan* (hạng 1). Tuy nhiên, MILQ_extend có *scheduling latency* và *average_utilization* thấp hơn MTMC, cho thấy khả năng quản lý thời gian và khai thác tài nguyên chưa tối ưu bằng MTMC.
- **Giải thuật FFD** đạt **xếp hạng tổng thể thứ 3**, thể hiện ưu điểm về *average_utilization* (hạng 1) và duy trì mức ổn định ở các tiêu chí khác. FFD có kết quả trung bình, phù hợp cho các bài toán yêu cầu độ phức tạp vừa phải, do bản chất là thuật toán heuristic đơn giản (first-fit) nên nhanh nhưng không đảm bảo tính tối ưu trong các cấu hình lớn hoặc phức tạp.
- **Giải thuật NoTADS** đạt **xếp hạng tổng thể thấp nhất** (hạng 4), với kết quả kém ở hầu hết các tiêu chí, đặc biệt là *average_fidelity* (hạng 1) nhưng lại thua kém đáng kể ở *makespan*, *average_throughput*, và *average_responseTime*. Điều này cho thấy NoTADS không phù hợp cho các ứng dụng yêu cầu cân bằng giữa nhiều tiêu chí.

Từ các kết quả cụ thể trên, nhóm nhận thấy rằng MTMC và MILQ có khả năng cân bằng hiệu quả giữa độ chính xác và hiệu suất tổng thể, trong khi FFD mang tính chất gần với heuristic cơ bản. Ngược lại, NoTADS cho thấy tính ngẫu nhiên cao, phù hợp hơn cho nghiên cứu lý thuyết thay vì ứng dụng thực tiễn.

7.7 Đánh giá tổng quan

Từ phân tích trên, nhóm nghiên cứu đề xuất:

- Ưu tiên sử dụng các giải thuật heuristic trong các hệ thống thực tế, nhờ khả năng cân bằng giữa độ chính xác và thông lượng.

- Giải thuật ILP nên được sử dụng trong các bài toán đặc thù yêu cầu độ chính xác cao, ít yêu cầu real-time.

7.8 Tóm tắt chương

Chương này đã trình bày quá trình thực nghiệm, bao gồm mô tả hệ thống, benchmark, kịch bản thử nghiệm, kết quả và phân tích. Những kết quả này làm nền tảng cho phần kết luận và định hướng phát triển trong chương tiếp theo.

Chương 8

Tổng kết

Trong chương 8, các nhận xét tổng hợp về các giải thuật và những hướng phát triển tiếp theo của đề án sẽ được trình bày.

8.1 Kết luận

Trong đề án này, nhóm đã tiến hành nghiên cứu, thiết kế và xây dựng một hệ thống mô phỏng nhằm đánh giá hiệu quả của các giải thuật định thời tài nguyên cho tính toán lượng tử, đặc biệt trong bối cảnh các thiết bị thuộc kỷ nguyên NISQ. Các kết quả chính đạt được bao gồm:

- Tổng hợp và hệ thống hóa kiến thức lý thuyết nền tảng về định thời lượng tử, kỹ thuật xử lý mạch lượng tử (circuit cutting, circuit knitting), các hệ mô phỏng lượng tử phổ biến và đưa ra các tiêu chí đánh giá.
- Triển khai thành công một hệ mô phỏng có khả năng kiểm tra các benchmark (batch circuit) đa dạng, mô phỏng backend QPU với các đặc điểm phần cứng khác nhau, tích hợp các giải thuật định thời (ILP, heuristic) và thu thập kết quả dựa trên tám chỉ số đánh giá (fidelity, throughput, response time, turnaround time, makespan, scheduling latency, utilization, cut sampling overhead).
- Thực hiện các kịch bản thử nghiệm với sự thay đổi về số lượng job, kích thước mạch và loại backend QPU; đồng thời phân tích ảnh hưởng của các yếu tố đầu vào đến kết quả thực nghiệm.

- Đưa ra phân tích so sánh, nhận định ưu nhược điểm giữa các giải thuật định thời, đồng thời phát hiện các vấn đề thực tiễn như độ ổn định của giải thuật trong môi trường nhiều biến động.

8.2 Hướng phát triển

Từ kết quả đã đạt được, nhóm đề xuất một số hướng phát triển tiếp theo:

- Mở rộng hệ mô phỏng để tích hợp thêm các giải thuật định thời dựa trên trí tuệ nhân tạo (AI-based scheduling) như reinforcement learning, graph neural network, nhằm cải thiện khả năng thích nghi và tối ưu đa mục tiêu.
- Kết nối trực tiếp hệ mô phỏng với backend phần cứng thực (IBM Quantum, IonQ, Rigetti) qua API để thu thập kết quả thực nghiệm song song với kết quả mô phỏng.
- Tối ưu hóa kiến trúc phần mềm của hệ thống để hỗ trợ chạy phân tán trên nhiều node, tăng khả năng xử lý các kịch bản benchmark lớn và phức tạp hơn.
- Khám phá tích hợp circuit cutting, circuit knitting cùng các chiến lược lập lịch thông minh nhằm khai thác tối đa tài nguyên trong môi trường multi-QPU.

Kết quả và hệ thống của đồ án có thể được sử dụng như một nền tảng tham chiếu cho các nghiên cứu tiếp theo về quản lý tài nguyên, tối ưu hóa lập lịch và kiểm thử giải thuật lập lịch lượng tử trong môi trường hạn chế tài nguyên.

Tài liệu tham khảo

- [1] F. Arute, K. Arya, R. Babbush, *et al.*, “Quantum supremacy using a programmable superconducting processor,” *Nature*, vol. 574, no. 7779, pp. 505–510, 2019.
- [2] L. Innocenti, “Machine-learning-assisted state and gate engineering for quantum technologies,” Ph.D. dissertation, Queen’s University Belfast, 2020.
- [3] J. Preskill, “Quantum computing in the nisq era and beyond,” *Quantum*, vol. 2, p. 79, 2018.
- [4] S. Bravyi, G. Smith, and J. A. Smolin, “Trading classical and quantum computational resources,” *Physical Review X*, vol. 6, no. 2, p. 021 043, 2016.
- [5] T. Peng, A. W. Harrow, M. Ozols, and X. Wu, “Simulating large quantum circuits on a small quantum computer,” *Physical review letters*, vol. 125, no. 15, p. 150 504, 2020.
- [6] K. Mitarai and K. Fujii, “Constructing a virtual two-qubit gate by sampling single-qubit operations,” *New Journal of Physics*, vol. 23, no. 2, p. 023 021, 2021.
- [7] D. Bhoulmik, R. Majumdar, A. Saha, and S. Sur-Kolay, “Distributed scheduling of quantum circuits with noise and time optimization,” *arXiv preprint arXiv:2309.06005*, 2023.
- [8] D. Bhoulmik, R. Majumdar, and S. Sur-Kolay, “Resource-aware scheduling of multiple quantum circuits on a hardware device,” *arXiv preprint arXiv:2407.08930*, 2024.
- [9] P. Seitz, M. Geiger, and C. B. Mendl, “Multithreaded parallelism for heterogeneous clusters of qpus,” in *ISC High Performance 2024 Research Paper Proceedings (39th International Conference)*, Prometeus GmbH, 2024, pp. 1–8.

- [10] A. Javadi-Abhari, M. Treinish, K. Krsulich, *et al.*, *Quantum computing with Qiskit*, 2024. arXiv: [2405.08810](https://arxiv.org/abs/2405.08810) [quant-ph].
- [11] D. S. Johnson, “Near-optimal bin packing algorithms,” Ph.D. dissertation, Massachusetts Institute of Technology, 1973.
- [12] J. Yao, J. Wang, F. Yue, J. Xu, and Z. Shan, “Mtmc: A scheduling framework of multi-tasking mapping on multi-chips,” 2022.
- [13] A. Barenco, D. Deutsch, A. Ekert, and R. Jozsa, “Conditional quantum dynamics and logic gates,” *Physical Review Letters*, vol. 74, no. 20, p. 4083, 1995.
- [14] L. Ruiz-Perez and J. C. Garcia-Escartin, “Quantum arithmetic with the quantum fourier transform,” *Quantum Information Processing*, vol. 16, pp. 1–14, 2017.
- [15] E. Şahin, “Quantum arithmetic operations based on quantum fourier transform on signed integers,” *International Journal of Quantum Information*, vol. 18, no. 06, p. 2 050 035, 2020.
- [16] L. K. Grover, “A fast quantum mechanical algorithm for database search,” in *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, 1996, pp. 212–219.
- [17] P. W. Shor, “Algorithms for quantum computation: Discrete logarithms and factoring,” in *Proceedings 35th annual symposium on foundations of computer science*, Ieee, 1994, pp. 124–134.
- [18] P. W. Shor, “Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer,” *SIAM review*, vol. 41, no. 2, pp. 303–332, 1999.
- [19] M. AbuGhanem, “Ibm quantum computers: Evolution, performance, and future directions,” *The Journal of Supercomputing*, vol. 81, no. 5, p. 687, 2025.
- [20] Y. Wu, W.-S. Bao, S. Cao, *et al.*, “Strong quantum computational advantage using a superconducting quantum processor,” *Physical review letters*, vol. 127, no. 18, p. 180 501, 2021.
- [21] Y.-T. Kao, J.-L. Liao, and H.-C. Hsu, “Solving combinatorial optimization problems on fujitsu digital annealer,” *arXiv preprint arXiv:2311.05196*, 2023.

- [22] D. Willsch, M. Willsch, C. D. Gonzalez Calaza, *et al.*, “Benchmarking advantage and d-wave 2000q quantum annealers with exact cover problems,” *Quantum Information Processing*, vol. 21, no. 4, p. 141, 2022.
- [23] A. C. Santos, “The ibm quantum computer and the ibm quantum experience,” *arXiv preprint arXiv:1610.06980*, 2016.
- [24] Microsoft, *Azure Quantum Development Kit*.
- [25] P. J. Karalekas, N. A. Tezak, E. C. Peterson, C. A. Ryan, M. P. da Silva, and R. S. Smith, “A quantum-classical cloud platform optimized for variational hybrid algorithms,” *Quantum Science and Technology*, vol. 5, no. 2, p. 024 003, Apr. 2020.
- [26] Amazon Web Services, *Amazon Braket*, 2020.
- [27] G. G. Guerreschi, J. Hogaboam, F. Baruffa, and N. P. Sawaya, “Intel quantum simulator: A cloud-ready high-performance simulator of quantum circuits,” *Quantum Science and Technology*, vol. 5, no. 3, p. 034 007, 2020.
- [28] C. Developers, *Cirq*. Zenodo, Apr. 2025.
- [29] B. Lu, Z. Chen, and Y. Wu, “Qsra: A qpu scheduling and resource allocation approach for cloud-based quantum computing,” *arXiv preprint arXiv:2411.05283*, 2024.
- [30] P. Seitz, M. Geiger, C. Ufrecht, *et al.*, “Scim milq: An hpc quantum scheduler,” in *2024 IEEE International Conference on Quantum Computing and Engineering (QCE)*, IEEE, vol. 2, 2024, pp. 292–298.
- [31] O. Sinnen, *Task scheduling for parallel systems*. John Wiley & Sons, 2007.
- [32] A. Vivas, A. Tchernykh, and H. Castro, “Trends, approaches, and gaps in scientific workflow scheduling: A systematic review,” *IEEE Access*, 2024.
- [33] M. Padberg, *Linear optimization and extensions*. Springer Science & Business Media, 2013, vol. 12.
- [34] C. A. Floudas and X. Lin, “Mixed integer linear programming in process scheduling: Modeling, algorithms, and applications,” *Annals of Operations Research*, vol. 139, pp. 131–162, 2005.

- [35] S. Desale, A. Rasool, S. Andhale, and P. Rane, “Heuristic and meta-heuristic algorithms and their relevance to the real world: A survey,” *Int. J. Comput. Eng. Res. Trends*, vol. 351, no. 5, pp. 2349–7084, 2015.
- [36] S. Brandhofer, I. Polian, and K. Krsulich, “Optimal partitioning of quantum circuits using gate cuts and wire cuts,” *IEEE Transactions on Quantum Engineering*, vol. 5, pp. 1–10, 2023.
- [37] W. Tang, T. Tomesh, M. Suchara, J. Larson, and M. Martonosi, “Cutqc: Using small quantum computers for large quantum circuit evaluations,” in *Proceedings of the 26th ACM International conference on architectural support for programming languages and operating systems*, 2021, pp. 473–486.
- [38] C. Piveteau and D. Sutter, “Circuit knitting with classical communication,” *IEEE Transactions on Information Theory*, vol. 70, no. 4, pp. 2734–2745, 2023.
- [39] R. Orús, “Tensor networks for complex quantum systems,” *Nature Reviews Physics*, vol. 1, no. 9, pp. 538–550, 2019.
- [40] J.-w. Pan and A. Zeilinger, “Greenberger-horne-zeilinger-state analyzer,” *Physical Review A*, vol. 57, no. 3, p. 2208, 1998.
- [41] S. Garnerone, A. Marzuoli, and M. Rasetti, “Quantum knitting,” *Laser Physics*, vol. 16, pp. 1582–1594, 2006.
- [42] C. Campbell, F. T. Chong, D. Dahl, *et al.*, “Superstaq: Deep optimization of quantum programs,” in *2023 IEEE International Conference on Quantum Computing and Engineering (QCE)*, IEEE, vol. 1, 2023, pp. 1020–1032.
- [43] B. Wescott, *Every computer performance book: How to avoid and solve performance problems on the computers you work with*. CreateSpace Independent Publishing Platform, 2013.
- [44] V. Bergholm, J. Izaac, M. Schuld, *et al.*, “PennyLane: Automatic differentiation of hybrid quantum-classical computations,” *arXiv preprint arXiv:1811.04968*, 2018.
- [45] T. Jones, A. Brown, I. Bush, and S. C. Benjamin, “Quest and high performance simulation of quantum computers,” *Scientific reports*, vol. 9, no. 1, p. 10 736, 2019.

- [46] D. Wierichs, J. Izaac, C. Wang, and C. Y.-Y. Lin, “General parameter-shift rules for quantum gradients,” *Quantum*, vol. 6, p. 677, 2022.
- [47] E. Giortamis, F. Romão, N. Tornow, and P. Bhatotia, “Qos: A quantum operating system,” *arXiv preprint arXiv:2406.19120*, 2024.
- [48] N. Quetschlich, L. Burgholzer, and R. Wille, “Mqt bench: Benchmarking software and design automation tools for quantum computing,” *Quantum*, vol. 7, p. 1062, 2023.
- [49] P. D. Nation and M. Treinish, “Suppressing quantum circuit errors due to system variability,” *PRX Quantum*, vol. 4, p. 010 327, 1 Mar. 2023.
- [50] S.-J. Gu, “Fidelity approach to quantum phase transitions,” *International Journal of Modern Physics B*, vol. 24, no. 23, pp. 4371–4458, 2010.
- [51] M. Cozzini, R. Ionicioiu, and P. Zanardi, “Quantum fidelity and quantum phase transitions in matrix product states,” *Physical Review B—Condensed Matter and Materials Physics*, vol. 76, no. 10, p. 104 420, 2007.
- [52] A. V. Goponenko, K. Lamar, C. Peterson, B. A. Allan, J. M. Brandt, and D. Dechev, “Metrics for packing efficiency and fairness of hpc cluster batch job scheduling,” in *2022 IEEE 34th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, IEEE, 2022, pp. 241–252.
- [53] H. P. Williams, “Integer programming,” in *Logic and integer programming*, Springer, 2009, pp. 25–70.
- [54] W. van Dam, M. Mykhailova, and M. Soeken, “Using azure quantum resource estimator for assessing performance of fault tolerant quantum computation,” in *Proceedings of the SC’23 Workshops of the International Conference on High Performance Computing, Network, Storage, and Analysis*, 2023, pp. 1414–1419.
- [55] P. D. Nation and M. Treinish, “Suppressing quantum circuit errors due to system variability,” *PRX Quantum*, vol. 4, no. 1, p. 010 327, 2023.
- [56] M. Bandic, L. Prielinger, J. Nüßlein, *et al.*, “Mapping quantum circuits to modular architectures with qubo,” in *2023 IEEE International Conference on Quantum Computing and Engineering (QCE)*, IEEE, vol. 1, 2023, pp. 790–801.

- [57] M. Y. Siraichi, V. F. d. Santos, C. Collange, and F. M. Q. Pereira, “Qubit allocation,” in *Proceedings of the 2018 international symposium on code generation and optimization*, 2018, pp. 113–125.
- [58] L. Li, P. Anand, K. He, and D. Englund, “Dynamic inhomogeneous quantum resource scheduling with reinforcement learning,” *arXiv preprint arXiv:2405.16380*, 2024.
- [59] S. Stein, S. Sussman, T. Tomesh, *et al.*, “Hetarch: Heterogeneous microarchitectures for superconducting quantum systems,” in *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture*, 2023, pp. 539–554.
- [60] Y. Li, J. Ma, Z. Xie, Z. Hu, X. Shen, and K. Zhang, “A scheduling method for heterogeneous signal processing platforms based on quantum genetic algorithm,” *Applied Sciences*, vol. 13, no. 7, p. 4428, 2023.
- [61] E. Rieffel and W. Polak, “An introduction to quantum computing for non-physicists,” *ACM Computing Surveys (CSUR)*, vol. 32, no. 3, pp. 300–335, 2000.
- [62] S. Mummadi and B. Rudra, “Fundamentals of quantum computation and basic quantum gates,” in *Handbook of Research on Quantum Computing for Smart Environments*, IGI Global, 2023, pp. 1–24.
- [63] K. Bharti, A. Cervera-Lierta, T. H. Kyaw, *et al.*, “Noisy intermediate-scale quantum algorithms,” *Reviews of Modern Physics*, vol. 94, no. 1, p. 015 004, 2022.
- [64] M. Schuld, “Supervised quantum machine learning models are kernel methods,” *arXiv preprint arXiv:2101.11020*, 2021.

Phụ lục A: Cài đặt và sử dụng môi trường

Toàn bộ mã nguồn có thể được truy cập tại: [GitHub](#)

8.3 Tải Về và Kiểm Tra Công Cụ

Làm theo các bước dưới đây để tải mã nguồn và thiết lập môi trường:

- Sao chép kho lưu trữ (clone repository):

```
1 git clone https://github.com/MagePro310/  
   Quantum_Simulation_Scheduling.git  
2 cd Quantum_Simulation_Scheduling
```

- Cài đặt các gói phụ thuộc:

```
1 pip install -r requirements.txt
```

- Kiểm tra cài đặt:

```
1 python simulator.py --help
```

Lệnh trên sẽ hiển thị hướng dẫn sử dụng công cụ nếu cài đặt thành công.

8.4 Ví Dụ Sử Dụng Cơ Bản

Dưới đây là ví dụ minh họa cách sử dụng thư viện trong một chương trình Python:

- Import lớp mô phỏng vào chương trình:

```
1 from quantum_simulator import QuantumSimulator
```

- **Khởi tạo trình mô phỏng với cấu hình mặc định:**

```
1 simulator = QuantumSimulator(config="default_config.json"  
    ")
```

- **Định nghĩa và thực thi các thao tác lượng tử:**

```
1 simulator.add_operation("H", qubit=0)  
2 simulator.add_operation("CNOT", control=0, target=1)  
3 simulator.run()
```

Lưu ý: Đảm bảo rằng file `default_config.json` tồn tại và chứa các thiết lập hợp lệ trước khi khởi tạo trình mô phỏng.

Phụ lục B: Danh sách các cổng lượng tử

Bảng 8.1: Các cổng lượng tử cơ bản và đặc điểm của chúng (Gates on One Target Qubit)

Ký hiệu	Tên cổng	Số qubit	Biểu diễn ma trận	Thuộc tính
H	Cổng Hadamard	1	$\frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$	Không vết (Traceless) Tự nghịch đảo (Involutory)
ID	Cổng đồng nhất	1	$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$	Cổng đơn vị (Identity) Tự nghịch đảo (Involutory)
X	Cổng Pauli-X	1	$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$	Không vết Tự nghịch đảo Thuộc nhóm Pauli
Y	Cổng Pauli-Y	1	$\begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}$	Không vết Tự nghịch đảo Thuộc nhóm Pauli
Z	Cổng Pauli-Z	1	$\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$	Không vết Tự nghịch đảo Thuộc nhóm Pauli

Bảng 8.2: Các cổng xoay và cổng pha trong tính toán lượng tử

Ký hiệu	Tên cổng	Số qubit	Ma trận biểu diễn	Thuộc tính
$RX(\theta)$	Cổng xoay quanh trục X	1	$\begin{bmatrix} \cos(\frac{\theta}{2}) & -i\sin(\frac{\theta}{2}) \\ -i\sin(\frac{\theta}{2}) & \cos(\frac{\theta}{2}) \end{bmatrix}$	Đơn vị đặc biệt (định thức = 1) Có tham số liên tục θ , chu kỳ 4π
$RY(\theta)$	Cổng xoay quanh trục Y	1	$\begin{bmatrix} \cos(\frac{\theta}{2}) & -\sin(\frac{\theta}{2}) \\ \sin(\frac{\theta}{2}) & \cos(\frac{\theta}{2}) \end{bmatrix}$	Đơn vị đặc biệt (định thức = 1) Có tham số liên tục θ , chu kỳ 4π
$RZ(\theta)$	Cổng xoay quanh trục Z	1	$\begin{bmatrix} e^{-i\frac{\theta}{2}} & 0 \\ 0 & e^{i\frac{\theta}{2}} \end{bmatrix}$	Đơn vị đặc biệt (định thức = 1) Có tham số liên tục θ , chu kỳ 4π
$R_1(\theta)$	Cổng pha toàn cục theo trục Z	1	$\begin{bmatrix} 1 & 0 \\ 0 & e^{i\theta} \end{bmatrix}$	Có tham số liên tục θ , chu kỳ 2π
S	Cổng pha S	1	$\begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix}$	Cổng pha cơ bản
S^\dagger	Cổng S nghịch đảo	1	$\begin{bmatrix} 1 & 0 \\ 0 & -i \end{bmatrix}$	Cổng pha nghịch đảo
T	Cổng pha T	1	$\begin{bmatrix} 1 & 0 \\ 0 & \frac{1+i}{\sqrt{2}} \end{bmatrix}$	Cổng pha $\pi/8$
T^\dagger	Cổng T nghịch đảo	1	$\begin{bmatrix} 1 & 0 \\ 0 & \frac{1-i}{\sqrt{2}} \end{bmatrix}$	Cổng pha $\pi/8$ nghịch đảo

Bảng 8.3: Các cổng điều khiển một qubit (có một qubit điều khiển và một qubit mục tiêu)

Ký hiệu	Tên cổng	Số qubit	Ma trận biểu diễn	Thuộc tính
CH	Cổng Hadamard điều khiển	2	$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ 0 & 0 & \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{pmatrix}$	Tự nghịch đảo (Involutory)
CX hoặc $CNOT$	Cổng NOT điều khiển (CNOT)	2	$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$	Tự nghịch đảo (Involutory)
CY	Cổng Y điều khiển	2	$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & -i \\ 0 & 0 & i & 0 \end{pmatrix}$	Tự nghịch đảo (Involutory)
CZ	Cổng Z điều khiển	2	$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix}$	Tự nghịch đảo (Involutory) Hoán đổi qubit điều khiển và qubit mục tiêu không thay đổi tác dụng

Bảng 8.4: Cổng hoán đổi và các cổng xoay điều khiển trong tính toán lượng tử

Ký hiệu	Tên cổng	Số qubit	Ma trận biểu diễn	Thuộc tính
SWAP	Cổng hoán đổi trạng thái hai qubit	2	$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$	Tự nghịch đảo (Involutory) Hoán đổi hai qubit mục tiêu không thay đổi tác dụng
$CRX(\theta)$	Cổng xoay quanh trục X có điều khiển	2	$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \cos(\frac{\theta}{2}) & -i\sin(\frac{\theta}{2}) \\ 0 & 0 & -i\sin(\frac{\theta}{2}) & \cos(\frac{\theta}{2}) \end{bmatrix}$	Đơn vị đặc biệt (định thức = 1) Tham số liên tục θ Chu kỳ 4π
$CRY(\theta)$	Cổng xoay quanh trục Y có điều khiển	2	$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \cos(\frac{\theta}{2}) & -\sin(\frac{\theta}{2}) \\ 0 & 0 & \sin(\frac{\theta}{2}) & \cos(\frac{\theta}{2}) \end{bmatrix}$	Đơn vị đặc biệt (định thức = 1) Tham số liên tục θ Chu kỳ 4π
$CRZ(\theta)$	Cổng xoay quanh trục Z có điều khiển	2	$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & e^{-i\frac{\theta}{2}} & 0 \\ 0 & 0 & 0 & e^{i\frac{\theta}{2}} \end{bmatrix}$	Đơn vị đặc biệt (định thức = 1) Tham số liên tục θ Chu kỳ 4π
$CR_1(\theta)$	Cổng pha toàn cục có điều khiển	2	$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & e^{i\theta} \end{bmatrix}$	Tham số liên tục θ Chu kỳ 2π Hoán đổi qubit điều khiển và qubit mục tiêu không thay đổi tác dụng