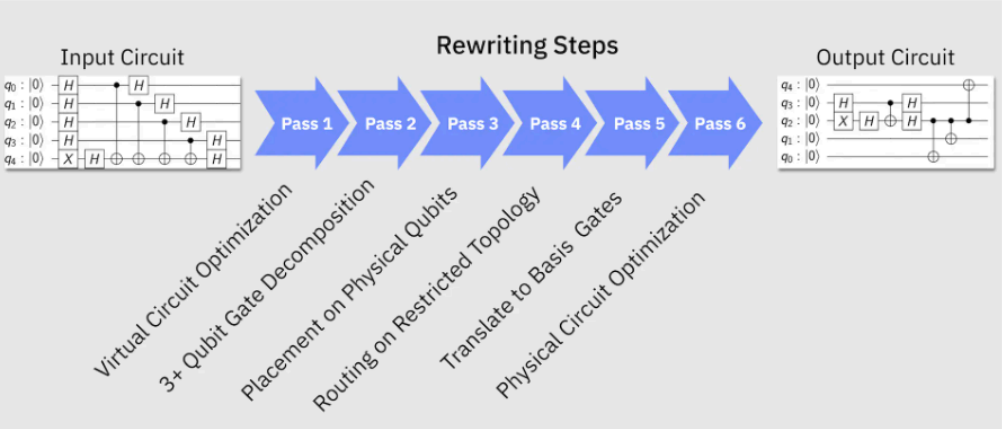
(지정주제 6) 전이조

배경

- Quantum transpile 과정



  - Init: 회로를 백엔드에 임베딩하기 전에 필요한 초기 작업을 실행
  - Layout: 회로의 가상 큐빗을 백엔드의 물리적 큐빗에 매핑하는 레이아웃을 적용
  - Routing: 레이아웃이 적용된 후 백엔드의 연결성에 맞게 게이트(예: swap)를 삽입
  - Translation: 회로의 게이트를 목표 백엔드의 기본 세트로 번역
  - Optimization: 회로 최적화를 반복적으로 실행
  - Scheduling: 하드웨어 인식 스케줄링 작업을 실행
- 각 stage마다 적용할 수 있는 여러 pass가 있음
  - https://docs.quantum.ibm.com/api/qiskit/transpiler_passes

목적: Fidelity 높이기

- Routing: 알고리즘 개선
- Init ~ Scheduling: 최적의 pass 조합 찾기

구체적인 방법

방법 1: Routing 알고리즘 개선 → "MetaSabre"

- 아이디어
  - Sabre 알고리즘은 어떤 swap 할 것인지 결정할 때 단순 heuristic 사용
  - SWAP 결정 단계에서, 그 swap 했을 때 미래의 sabre 얼마나 잘 작동하는지 조사하고 판단.
  - “잘"?: <실행 가능했었던 gate> / <탐색 깊이> 로 평가
- 결과

    - QFT transpilation 성능을 비교하기 위해서 qiskit transpilation process 에 직접 넣었으나 QFT transpilation routing 과정에 sabre가 사용되지 않음 (preprocessing 이 routing 해결)
    - Random: 작은 크기의 circuit에서만 실행가능했고, 대부분의 상황에서는 sabre와 swap 개수 비슷. 하지만 가끔 성능 개선이 이루어지기도 했음.
- 분석
    - recursion을 할 때 caller function에서의 context가 영향받지 않도록 deepcopy를 사용했는데, 이 부분이 특히 병목이 된 것 같다.
    - recursion대신 다른 방법으로 이 방식을 구현하면 좋을 것 같음 (롤백 직접 구현 등)

방법 2-1: 최적 조합 찾기 → Pass 조합 Genetic algorithm

- 아이디어
    - 어떤 Pass 조합이 좋은 성능을 보일까? 이를 자동으로 찾아보자.
- 결과: 2~14 큐비트 회로에 대해 qiskit O2랑 비교.
    - QFT: 유사한 성능
    - Random: (아마도) 비반복 패턴에 대해 1~4% 높은 성능 보임 (회로 Depth = 큐비트 수 + 1)
- 분석
    - QFT: 회로가 반복적인 패턴으로 구성돼서 O2가 이미 충분히 잘 하는 것 같다.
    - Random: 반복적인 패턴이 없어서 O2도 못하는 부분이 있는 것 같다. 따라서 genetic이 20% (20개 중 4개 개선) 회로에 대해 개선된 결과를 보이는 듯.

방법 2-2: 최적 조합 찾기 → Plugin 조합 Bruteforce

- 아이디어
    - 어떤 Plugin 조합이 좋은 성능을 보일까? 이를 전부 돌려서 비교해보자. (Plugin = Qiskit이 정의한 Pass 조합)
- 결과
    - QFT에 대해서 높아짐
- 분석
    - brute-force로 찾은 조합들은 전체적으로 O3보다 10~20% 좋은 성능을 보인다.
    - Transpilation 과정이 비결정적이기 때문에 점수가 높은 option들을 찾았어도 가끔 점수가 낮게 나올 때가 있었다.

소스코드

https://github.com/zzz845zz/Quantum-Korea-Hackathon-2024

- 방법 1: MetaSabre
    - 구현: meta_sabre 폴더
- 방법 2-1: Pass genetic 알고리즘
    - 구현: genetic 폴더

    - 결과: out 폴더
- 방법 2-2: Plugin bruteforce 알고리즘
    - 구현: score_machine 폴더
    - 결과: bruteforce_scores 폴더

역할분담

- 승민: 테스트 환경 구축 및 Genetic 알고리즘 개발
- 강욱: Routing 알고리즘, Bruteforce 알고리즘 개발
- 경민: 양자 전문가 아니므로 배우러 옴, 전체 프로젝트 지도, 결과 분석, Rust 엔지니어링 서포트